UAEM | Universidad Autónoma del Estado de México

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

Desarrollo de Algoritmos de Tratamiento de Imágenes en el Sector Agropecuario.

Tesis que presenta

Daniel Ayala Niño

Para obtener el Grado de

Maestro en Ciencias en Computación

Asesor de Tesis:

Dr. Jair Cervantes Canales

Texcoco, Estado de México. Octubre 2020

Declaration of Authorship

«No es el conocimiento, sino el acto de aprendizaje, y no la posesión, sino el acto de llegar allí, lo que concede el mayor disfrute.»

Carl Friedrich Gauss

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

Resumen

Maestría en Ciencias de la Computación

Desarrollo de Algoritmos de Tratamiento de Imágenes en el Sector Agropecuario.

by Daniel Ayala Niño

El reconocimiento de plantas a través de las hojas ha sido un campo de investigación muy estudiado. Los algoritmos actuales pueden perfectamente clasificar las hojas de diferentes especies, dado a que las hojas tienen la cualidad de ser diferentes entre sí. Haciendo la tarea de clasificación más fácil al tener rasgos diferenciables, como lo puede ser su color, morfología y textura. Sin embargo, para estos algoritmos es difícil clasificar hojas de diferentes variedades que pertenecen a la misma especie. Estas variedades pueden ser muy similares entre sí, llegando a ser un gran reto para expertos botánicos, donde se requiere de práctica para especializarse en una especie de plantas. Siendo la clasificación de este tipo de plantas el principal enfoque de la investigación. Como se mencionó anteriormente, ya existen métodos en la clasificación de plantas de diferentes especies. El más notable es el uso de Deep Learning (DL, Aprendizaje Profundo), cuya simplicidad de aprender a partir de "datos crudos" hace que esta sea una tarea sencilla. No obstante, en el estado del arte esta técnica no se ha explorado en la clasificación de hojas complejas. Lo que se hizo en esta tesis fue utilizar algoritmos de Deep Learning para la clasificación de plantas por medio de hojas de durazno con 6 especies diferentes. La arquitectura de Deep Learning utilizada fueron las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés). Obteniendo así una precisión del 92.8571 % en comparación de otros modelos relacionados en el estado del arte.

Índice general

Acknowledgements VII

Resumen IX

Índice general XI

Índice de figuras XIII

Índice de cuadros XV

List of Abbreviations XVII

1. Introducción **1**

1.1. Problemática . . . 2
1.2. Justificación . . . 3
1.3. Objetivos y Alcances del Proyecto . . . 4
1.3.1. Objetivos Generales . . . 4
1.3.2. Objetivos Específicos . . . 4
1.4. Hipótesis . . . 5
1.5. Estado del Arte . . . 5

2. Preliminares **9**

2.1. Deep Learning . . . 10
2.1.1. El Perceptrón . . . 10
La función de costo del perceptrón . . . 10
El algoritmo del perceptrón . . . 11
2.1.2. Feedforward Networks (Fully Connected Networks) . . . 14
2.1.3. El algoritmo Backpropagation . . . 21
Variantes del Gradiente Descendiente . . . 28
2.1.4. Optimización . . . 29
2.1.5. Evaluación del Desempeño . . . 30

2.2. Redes Convolucionales CNN 33
2.2.1. La Operación Convolucional 34
2.2.2. ReLU y Softmax 39
2.2.3. Pooling 41
2.2.4. Regularización e Inicialización de una CNN 43
Regularización 43
Inicialización 45
2.2.5. La Arquitectura de una CNN 46
2.2.6. Arquitectura Siamese *SNN* 47

3. Metodología **51**
3.1. Conjunto de Datos 52
3.1.1. Recolección 53
3.1.2. Pre-procesamiento y Conjunto Final 53
3.2. Modelos Propuestos 53

4. Resultados y Conclusiones **59**
4.1. Certificado de Aceptación y Capítulo de Libro 59
4.2. Resultados 62
4.3. Conclusiones 63

A. Código Modelos **65**
A.1. Modelo *224X224* 65
A.2. Modelo *416X416* 67

B. Arquitecturas **71**
B.1. Modelo *224X224* 71
B.2. Modelo *416X416* 72

C. Gráficas AUC-ROC y Matrices de Confusión **75**
C.1. Modelo *224X224* 75
C.2. Modelo *416X416* 75

Bibliografía **77**

Índice de figuras

2.1. Interpretación geométrica perceptrón 13
2.2. Arquitectura básica perceptrón 14
2.3. Clases forman poliedro 15
2.4. Ejemplo de Neuronas 16
2.5. Mapeo de la primera capa oculta 17
2.6. Mapeo de los patrones en 2D 18
2.7. Ejemplo Red Feed-Forwardl final 19
2.8. Función sigmoidea 22
2.9. Función tanh 23
2.10. Función no convexa 24
2.11. Conexión de una neurona 25
2.12. Ejemplo Momentum 30
2.13. Terminología básica y compleja de la capa de una CNN 35
2.14. Ejemplo Convolución 36
2.15. Ejemplo Filtro de Bordes 37
2.16. Ejemplo Filtros de kernel de una CNN 38
2.17. Función ReLU 40
2.18. Ejemplo Activación ReLU 41
2.19. Ejemplo Max Pooling 42
2.20. Ejemplo Max Pooling sobre Imagen 43
2.21. Arquitectura General de una CNN 47
2.22. Arquitectura General de una SNN 49

3.1. Recolección de datos. 55
3.2. Muestra de las 6 clases de Durazno 56
3.3. Modelo Early Fusion 56
3.4. Modelos Propuestos 57

B.1. Modelo 224×224 71

B.2. Modelo 416×416 73

C.1. AUC-ROC Modelo 224×224 75
C.2. AUC-ROC Modelo 416×416 76

Índice de cuadros

3.1. Distribución final del conjunto de datos. 54
3.2. Modelo 224×224. 54
3.3. Modelo 416×416. 55

4.1. Comparación de los resultados experimentales de los modelos propuestos con otros artículos 62
4.2. Comparación de los CNN's propuestos en [8] con los modelos propuestos en esta investigación. 63

C.1. Matriz de Confusión Modelo 224×224. 76
C.2. Matriz de Confusión Modelo 416×416. 76

List of Abbreviations

DL	**D**eep **L**earning
CNN	**C**onvolutional **N**eural **N**etworks
ML	**M**achine **L**earning
ColPos	**Col**egio de **Pos**tgraduados
INIFAP	**I**nstituto **N**acional de **I**nvestigaciones **F**orestales **A**grícolas y **P**ecuarias
IDSC	**I**nner-**D**istance **S**hape **C**ontext
MMC	**M**ove **M**edian **C**enter
LVQ	**L**earning **V**ector **Q**uantization
RBF	**R**adial **B**ase **F**unction
HOG	**H**istograms of **O**riented **G**radients
SVM	**S**upport **V**ector **M**achine
CTD	**C**urvelet **T**ransform **D**escriptors
LBP	**L**ocal **B**inary **P**attern
GLCM	**G**rey **L**evel **C**o-occurrence **M**atrix
NCFS	**N**eighbourhood **C**omponent **F**eature **S**election
PDA	**P**enalized **D**iscriminant **A**nalysis
RF	**R**andom **F**orest
DN	**D**econvolutional **N**etworks
CAE	**C**onvolutional **A**uto**E**ncoder
NN	**N**eural **N**etwork
SGD	**S**tochastic **G**radient **D**escent
Adam	**Ada**ptative **M**oment Estimation
TP	**T**rue **P**ositive
FP	**F**alse **P**ositive
TN	**T**rue **N**egative
FN	**F**alse **N**egative
AUC	**A**rea **U**nder the **C**urve
ROC	**R**eceiver **O**perating **C**haracteristic
OCR	**O**ptical **C**haracter **R**ecognition

ReLU	Rectified Linear Unit
DNN	**D**eep **N**euranl **N**etwork
SNN	**S**iamese **N**eural **N**etwork

Capítulo 1

Introducción

Hoy en día, la tecnología ha demostrado ser de vital importancia para el desarrollo de diversas áreas. Las mejoras más importantes han sido las aplicadas por el aprendizaje automático, ya que puede imitar eficazmente las "tareas humanas". Las áreas en las que se ha desarrollado son diversas, como la agricultura donde se aplica big data y máquinas de aprendizaje (ML, por sus siglas en inglés) en la protección de cultivos contra plagas [1], en medicina [2] [3] y botánica [4], siendo de vital importancia para el desarrollo de la economía, del medio ambiente, puede ayudarnos en la investigación y creación de nuevos fármacos, y mejorar la producción agrícola.

Por lo que es importante poder clasificarlos. Por ejemplo, los botánicos utilizan las características de la hoja como herramienta comparativa para el estudio de las plantas, esto se debe a que después de ser recolectados duran varios meses. Mientras que las flores y los frutos no duran tanto [4]. Características como la textura, el color y las estructuras morfológicas juegan un papel importante en la distinción de especies vegetales [5]. La mayoría de estas características se identifican de manera manual, lo que requiere de experiencia y tiempo para clasificar una gran variedad de plantas. Esto trajo el primer problema, un experto en una especie o familia puede no estar familiarizado con otra, lo que se conoce como ”el impedimento taxonómico” [6].

Este ha sido un problema que la literatura ha abordado a lo largo de los años. Se han creado varios sistemas de visión para clasificar estas hojas. Esto se ha hecho aplicando diferentes técnicas, como extraer características de las hojas usando diferentes algoritmos para obtener la mayor cantidad posible de descriptores [2] [7] [8] [5] [9] y luego usar estas características para alimentar un clasificador. Hoy en día con la creciente popularidad del Aprendizaje Profundo (DL, por sus siglas en inglés) y las Redes Neurales Convolucionales (CNN por sus siglas en inglés) debido a su efectividad con la clasificación de imágenes, han demostrado que no es necesario extraer manualmente

las características antes de su clasificación [10]. Este tipo de modelos se entrenan utilizando grandes conjuntos de datos, y es de conocimiento común que cuanto mayor sea el conjunto de datos, mejor será el rendimiento del modelo [11]. Artículos relacionados han trabajado en grandes conjuntos de datos que clasifican diferentes tipos de plantas [12] [13], pero no hay muchos artículos relacionados enfocados en la clasificación de hojas complejas utilizando DL. En [8], se demostró que es posible clasificar hojas complejas, utilizando un Algoritmo Genético para seleccionar las características que mejor describen una planta, luego este conjunto de características se utilizó para entrenar un clasificador, obteniendo altas tasas de precisión. En el mismo trabajo, se utilizaron dos modelos de CNN para clasificar 6 variedades de durazno. Los resultados obtenidos no fueron alentadores, ya que la mejor precisión de la CNN fue de 62 %, y con el Algoritmo Genético, la precisión más alta fue del 83.937 %.

Nuestro propósito en esta investigación es crear un modelo que pueda clasificar diferentes variedades de durazno y aprendiendo de una hoja completa, no solo de una de sus partes, sino de su anverso y reverso; cosa que el estado del arte no ha trabajado. Para lograr esto, hemos creado un nuevo conjunto de datos en el que las dos caras de la hoja se consideran un elemento del conjunto de datos, e ingresando estos pares en nuestros nuevos modelos de redes neuronales convolucionales híbridas, cuyas primeras capas convolucionales se comparten, este enfoque ha superado a [8] con una precisión del 92.8571 %. Los modelos de CNN utilizados en esta investigación se basaron en los modelos expuestos en [14] y [15].

El documento está organizado de la siguiente manera. En este capítulo mostraremos la problemática de la investigación, su justificación, objetivos, hipótesis y estado del arte relacionado con la clasificación de hojas complejas. Luego, en el Capítulo 2, describimos los preliminares, que es toda la teoría relacionada para clasificar las hojas de durazno. En el Capítulo 3 mostramos la metodología aplicada en esta investigación. Mostramos los resultados obtenidos con el modelo descrito en el Capítulo 4. Finalmente, nuestras conclusiones relacionadas con los resultados obtenidos se describen en el Capítulo 5.

1.1. Problemática

La hoja o filoma es un órgano lateral y a veces terminal que brota del tallo o de las ramas; tiene forma generalmente laminar y estructura dorsiventral; se origina a partir de los rudimentos foliares del ápice vegetativo caulinar, los cuales a medida que se

separan del extremo se vuelven triangulares y adquieren el aspecto de una vaina, diferenciándose posteriormente en dos zonas, una superior que corresponderá al limbo y una inferior que corresponderá al peciolo. Las hojas se hallan en Magnoliophyta y Pteridófitos. En esta investigación, se trabajó con hojas de la especie "Prunus persica L. Batsch" (mejor conocida como Durazno), del tipo Magnoliophyta.

Una de las mejores formas de identificar una planta es por medio de las hojas, esto se debe a que la hoja, bajo las condiciones adecuadas puede durar meses, hasta cientos de años. Es importante su clasificación ya que esto ayuda a la creación de nuevas variedades de plantas. Estas nuevas variedades son mejoradas genéticamente, para que estas seas más resistentes a plagas, y/o el clima. Esto a su vez favorece a la producción agropecuaria. Ya que con estos beneficios se puede acelerar la producción y mejorar la calidad de un bien alimenticio.

La nueva variedad, resultado de la investigación experimental, va a ser muy similar a las demás variedades de esa misma especie, a esto se le conoce como hoja compleja. Esto hace que su clasificación sea aún más difícil por sus similitudes, y un taxónomo puede estar no familiarizado con esta especie y con la nueva variedad. Resultando en un mayor tiempo de espera en lo que el taxónomo se especializa en una especie en específico. Los datos recuperados por el taxónomo se recopilan de forma manual, lo que hace esta tarea aún más lenta y propensa a un margen de error más grande.

En este trabajo se propone desarrollar un modelo hibrido convolucional, para poder clasificar hojas complejas. Las variedades con las que se trabajaron pertenecen a la especie del Durazno. Estas variedades son objeto de investigación en el Colegio de Postgraduados en Montecillos.

1.2. Justificación

La agricultura y la ganadería pertenecen al sector primario de la economía de México. El sector primario está compuesto por todas las actividades donde los recursos naturales se aprovechan tal y como se obtienen de la naturaleza, ya sea para alimento o para generar materias primas. La principal fuente de alimentos de nuestro país es la agricultura y los productos pecuarios derivados de ésta. Gracias a la gran diversidad de climas, en México es posible cultivar una gran variedad de especies, esto a su vez ayuda a aumentar la economía de la región.

El Durazno tiene una gran demanda en México, tan solo en 2019 se produjeron 158,942.1 toneladas [16], y en 2018 México ocupó el 7mo lugar a nivel mundial en la

producción de durazno. [17]. En la actualidad hay dos institutos dedicados a la investigación y mejoramiento del Durazno, que son El Colegio de Postgraduados (ColPos) y el Instituto Nacional de Investigaciones Forestales Agrícolas y Pecuarias (INIFAP). Cuentan con una amplia gama de variedades y la elección de las variedades por plantar está íntimamente ligada a su requerimiento de frío invernal [18].

Es conveniente señalar que estos centros de investigación han definido el frío como requerimiento fundamental para todas las variedades de durazno y que se han realizado múltiples estudios para calcular la acumulación de frío que ocurre en el sitio que queremos plantar, para evitar el establecer variedades que no se ajusten a este requerimiento; una variedad mal seleccionada puede quedarse sin florecer o hacerlo anticipadamente durante el período de heladas con lo cual se pierde la futura cosecha [18]. De aquí la importancia de tener un control adecuado de cada variedad.

En el estado del arte existen varios métodos para poder clasificar plantas por medio de las hojas. Estos métodos se enfocan en clasificar plantas de diferentes especies, que hacen más simple la tarea para el algoritmo. Ya que estas hojas son fáciles de diferenciar gracias a las grandes diferencias que tienen. A diferencia de estos algoritmos, lo que buscamos es poder clasificar hojas complejas. Las hojas complejas son hojas que pertenecen a la misma especie, pero son de diferente variedad. Esto lo vamos a lograr por medio de algoritmos de ML, utilizando CNN-Hibridas.

1.3. Objetivos y Alcances del Proyecto

1.3.1. Objetivos Generales

Clasificar hojas complejas, utilizando un modelo CNN-Híbrido. Para poder tener una herramienta que identifique seis variedades de Durazno.

1.3.2. Objetivos Específicos

- 1. Analizar algoritmos de DL en el área de Redes Convolucionales, distinguidas por su gran desempeño en la clasificación de imágenes.
- 2. Implementar modelos Convolucionales en la clasificación de hojas complejas.
- 3. Analizar y discutir los resultados obtenidos de los modelos seleccionados.

1.4. Hipótesis

Se espera que, con el desarrollo de algoritmos de tratamiento de imágenes en DL, exista una mejora en la clasificación de hojas complejas de la especie del Durazno, reduciendo errores como son los falsos positivos y/o falsos negativos, así mismo siendo esta una alternativa que reduce el tiempo de evaluación.

1.5. Estado del Arte

En esta sección se describirá el estado del arte relacionado a la clasificación de hojas. En general, en la literatura existen dos métodos para clasificar hojas utilizando algoritmos computacionales. El primero, está relacionado con la extracción de características por medio de un algoritmo, para después obtener un vector de características que al final se utilizará para entrenar algún clasificador. Y el segundo, utiliza algoritmos de Deep Learning, como las CNN, para que de forma automática se extraiga de las imágenes la información más relevante para su clasificación. En algunos casos es necesario aplicar algún tipo de preprocesamiento en las imágenes, esto es algo inusual porque en muchos de los casos las CNN entrenan con "imágenes crudas", esto quiere decir que la imagen con la que entrena el modelo no recibe ningún tipo de preprocesamiento.

El acercamiento por medio del uso de extracción de características ha sido ampliamente utilizado, debido a que todas las hojas tienen características físicas únicas que las distinguen de las demás. Hay varias características que se pueden usar para describir la morfología de la hoja como el tamaño de su elongación, elipse [19]. Zhao et al. [13] propuso las características I-DSC, un nuevo descriptor de forma basado en el conteo, que puede reconocer hojas simples y compuestas. Aakif y Khan [12] extrajeron características morfológicas, descriptores de Fourier [20] y una nueva característica llamada definición de forma. Todas estas características se introdujeron en una red neuronal obteniendo una precisión del 96 %. Otra investigación donde hacen uso de características morfológicas de la hoja es el trabajo de Du et al. [21], quien propuso un clasificador de 20 tipos de plantas mediante la extracción de características morfológicas de la hoja, esto incluiría características geométricas de la hoja y sus momentos invariantes, dando como resultado un total de 15 características con las que se entrenaría por medio del algoritmo Move Median Centers (MMC).

Otra forma en la que se podría clasificar una hoja es por su textura. Este procedimiento se centra en extraer información sobre la textura de la hoja, utilizando los píxeles

de la imagen. Una forma de hacerlo es aplicando la teoría fractal, como Backes et al. [22], logrando así una precisión del 90 % en 10 tipos de hojas. Rashad et al. [23] clasificó las hojas por textura, combinando clasificadores como la cuantificación vectorial de aprendizaje (LVQ) junto con la función de base radial (RBF). Naresh y Nagendraswamy [24] utilizaron Patrones Binarios Locales Modificados para extraer características de la textura de la hoja, para luego ser clasificadas por vecinos cercanos. Olsen et al. [25] utilizó características de histogramas de gradientes orientados (HOG) para poder representar las texturas en la imagen de una hoja.

Las hojas también se pueden clasificar por su color, esto se puede hacer a partir de la comparación de sus histogramas [26]. Esta técnica tiene un problema, ya que la cromaticidad de la hoja no es estática, es variable. Factores como el tiempo, el clima y otros afectan el estado de la hoja. Una alternativa es utilizar las características texturales de la hoja ya que brindan información sobre las intensidades de los colores en la imagen, su forma geométrica, tamaño y la forma de la región previamente segmentada y cromática que hacen referencia a la intensidad del color de una región segmentada [27].

Como se ha visto existen varias técnicas para clasificar las hojas. Incluso hay investigaciones que utilizan todas las técnicas mencionadas anteriormente, creando un gran vector de características que describe la hoja. Siendo esta una desventaja para el clasificador que se utiliza, ya que pueden existir varias variables que generen ruido provocando confusión en el clasificador. Como Caglayan et al. [9] que extrajo características de forma y color para clasificarlas usando algoritmos como K-Neighbour, SVM, Naïve Bayes y Random Forest, obteniendo una precisión del 96 % trabajando junto con el conjunto de datos Flavia. También Elnemr [28], propuso un sistema que consta de cinco pasos, la cual primero se preprocesa la imagen, luego se aplica la segmentación de la imagen, para después extraer las características como descriptores de transformación de curva (CTD), patrón binario local (LBP) y textura de matriz de coocurrencia a nivel de grises (GLCM), luego la selección de características se realiza utilizando la técnica de selección de características de componentes de vecindario (NCFS), seleccionando las características altamente discriminatorias, obteniendo una precisión del 98 % . Los enfoques descritos anteriormente requieren mucho tiempo, ya que el preprocesamiento de la imagen, extracción y selección de características debe aplicarse a cada elemento del conjunto de datos, y para las hojas nuevas se debe realizar el mismo proceso. La mayoría de las veces se utilizan todas las características extraídas, pero no todas contribuyen de manera significativa, por lo que eso significa que se deben mantener

otros algoritmos para seleccionar las mejores características, esto significa más tiempo dedicando a seleccionar características [7] [8]. Un trabajo más reciente es el método propuesto por Cervantes et al. [7], propuso un método para resolver el rendimiento de algoritmos de aprendizaje automático en la clasificación de hojas complejas, utilizando diferentes técnicas de selección de características para mejorar los resultados. En esta investigación se establece un enfoque diferente, ya que en el trabajo de [7] el conjunto de datos de hojas complejas estaba compuesto por hojas similares físicamente, pero todas de diferentes especies. Para esta investigación es importante saber a qué tipo de variedad pertenece una hoja específica. Ya que sólo un experto en esta especie podría clasificarla.

Las CNN's se han convertido en el algoritmo más popular para tareas de visión por computadora, debido a su excelente rendimiento en diferentes campos como clasificación de imágenes, reconocimiento de objetos y segmentación de imágenes. Y ha mostrado un gran desempeño en la tarea de clasificación de plantas. Como Guillermo et al. [10], donde se clasificaron tres especies de leguminosas diferentes (frijol blanco, frijol rojo y soja), aquí se destacan diferentes niveles de detalles en la venación, usando segmentación de venas, usando dos tipos de configuraciones, alcanzando una precisión del 96,9 %, en comparación con un enfoque similar, realizado por Larese et al. [29] donde se alcanzó un 95,1 % de precisión, aquí, en lugar de usar una CNN, se extrajeron algunas características para alimentar una SVM, PDA o RF. Lee et al. [30], usó una CNN para clasificar las plantas del conjunto de datos MalayaKew, que consta de imágenes de hojas de 44 clases de especies y reutilizando una CNN ILSVRC2012 preentrenada. Aquí utilizaron redes deconvolucionales (DN) para averiguar en qué parte de las hojas se centra la CNN, obteniendo una precisión del 99,5 %. Las CNN también se utilizan para crear nuevas funciones como el trabajo de Ramos et al. [31], donde utilizó un Autoencoder Convolucional (CAE) se utiliza como extractor de características de las imágenes de la hoja, y este vector final se utiliza para alimentar una SVM, obteniendo una precisión del 94,74 %. Bing Wang y Dian Wang [15], propusieron un método de aprendizaje de Pocos-Disparos aplicando la Red Siamés, esto para crear un espacio métrico para la clasificación de hojas, donde muestras similares están cerca unas de otras y las muestras diferentes están lejos, esto se debe al pequeño conjunto de datos que se utilizó (Flavia, Swedish y Leafsnap), obtuvieron una precisión del 95,32 %. Lee et al. [14], también usó DNN para averiguar qué está aprendiendo una CNN de las hojas, y propuso modelos híbridos de CNN, estos modelos híbridos tienen dos entradas, una entrada es la imagen de la hoja, y la segunda entrada es un segmento de la misma hoja,

esto con el objetivo de enfocar la venación de la hoja. En este trabajo se describieron tres modelos híbridos diferentes, obteniendo una precisión del 96,3 % utilizando el modelo híbrido Early fusion (conv-sum).

Algo a destacar del estado del arte descrito anteriormente es que los modelos no se enfocaron en clasificar hojas complejas, que son hojas que pertenecen a la misma especie, pero son de diferentes variedades. Las hojas utilizadas en el estado del arte no cumplen con este requisito, siendo este nuestro objetivo, que consiste en clasificar 6 variedades de durazno. Además, la mayoría de las aplicaciones que usan CNN en la literatura no prestan atención a qué lado de la hoja están aprendiendo, ambos lados de las hojas se utilizan como diferentes muestras del conjunto de datos, en nuestro enfoque queremos que el modelo aprenda de una hoja completa, que consiste en su anverso y reverso, esto se logra con nuestros modelos híbridos convolucionales.

Capítulo 2

Preliminares

Machine Learning se ha convertido en uno de los tópicos de investigación más populares en las últimas décadas, y esta va en aumento[32]. Las áreas en las que se puede aplicar son variadas, entre ellas se encuentran: recuperación de conceptos multimedia, clasificación de imágenes, recomendación de videos, análisis de redes sociales, minería de textos, etc. Dentro de estos algoritmos de Machine Learning, se encuentra Deep Learning, que es muy popular dentro de estas áreas ya mencionadas. La gran cantidad de datos disponible, y el acelerado avance del hardware han provocado nuevos estudios en Deep Learning. Estos avances traen consigo la creación de modelos más complejos que obtienen una precisión alta, aunque no todas las veces una gran cantidad de datos y una arquitectura compleja resulta en algo positivo[33].

Deep Learning tiene sus raices en las Redes Neuronales (NN, por sus siglas en inglés). Una NN consta de muchos procesadores simples conectados llamados neuronas, cada uno produciendo una secuencia de activaciones de valor real. Las neuronas de entrada se activan a través de sensores que perciben el entorno, otras neuronas se activan a través de conexiones ponderadas de neuronas activas. Algunas neuronas pueden influir en el entorno desencadenando acciones. El aprendizaje o la asignación de créditos se trata de encontrar pesos que hagan que la NN muestre el comportamiento deseado, como puede ser conducir un coche. Según el problema y cómo están conectadas las neuronas, tal comportamiento puede requerir largas cadenas causales de etapas computacionales, donde cada etapa transforma (a menudo de forma no lineal) la activación agregada de la red. Deep Learning se trata de asignar pesos en muchas de estas etapas[34].

En este Capítulo se comenzará con DL (Sección 2.1), se hablará sobre temas principales de esta área que tal vez hoy en día pueden ser de conocimiento común para alguien especializado en esta área, o pueden ser sólo un tema del que conocen su existencia y los utilizan sin la necesidad de saber su porque y para que, para alguien que tiene

conocimiento de librerías especializadas. Aún así, es importante tener conocimiento de estos temas. Después se hablará sobre las CNN's (Sección 2.2), al igual que la Sección 2.1, se trata de incluir los temas principales para su entendimiento. Tal vez no se pueda profundizar tanto como uno quiere, ya que estas áreas de estudio son muy extensas y complejas, se requeriría más tiempo de estudio para poder comprenderlas con claridad. En caso de que se requiera una mayor profundidad se recomienda la lectura de los siguientes libros [35] y [36].

2.1. Deep Learning

2.1.1. El Perceptrón

Nuestro punto de inicio será un problema simple de dos clases linealmente separables (ω_1, ω_2). En otras palabras se nos estará brindando un conjunto de datos de entrenamiento, (y_n, x_n), donde $n = 1, 2, \dots, N$, con $y_n \in \{-1, +1\}$, y $x_n \in \mathbf{R}^l$ y se asume hay que hay un hiperplano:

$$\theta^T x = 0 \tag{2.1}$$

Tal que:

$$\theta^T x > 0, \text{ sí } x \in \omega_1$$
$$\theta^T x < 0, \text{ sí } x \in \omega_2$$

Este hiperplano, clasificará correctamente todos los puntos del conjunto de entrenamiento. Ahora, nuestro objetivo será desarrollar un algoritmo que iterativamente calcule el hiperplano que clasifique correctamente todos los patrones de ambas clases. Para lograr esto, una función de costo es necesaria.

La función de costo del perceptrón

Sea θ la estimación vectorial disponible en el paso de iteración actual de los parámetros desconocidos. Entonces habrían dos posibilidades: La primera es que todos los puntos sean clasificados correctamente; esto quiere decir que la solución ha sido encontrada. La otra alternativa es que θ clasifique correctamente alguno de los puntos, dejando los demás con una clasificación errónea. Sea γ el conjunto de datos clasificados erróneamente. El costo del perceptrón estará definido como:

$$J(\theta) = - \sum_{n:x_n \in \gamma} y_n \theta^T x_n \tag{2.2}$$

Donde:

$$y_n = \begin{cases} +1, \text{ sí } x \in \omega_1 \\ -1, \text{ sí } x \in \omega_2 \end{cases} \tag{2.3}$$

La función de costo no es negativa. De hecho, dado que la suma se aplica sobre los puntos mal clasificados, sí $x_n \in \omega_1(\omega_2)$, entonces $\theta^T x_n \leq (\geq) 0$, renderizando el producto $-y_n \theta^T x_n \geq 0$. Se tiene una solución cuando no hay puntos clasificados erroneamente, esto es, $\gamma = 0$. Por conveniencia, se podría decir que $J(\theta) = 0$.

La función de costo del perceptrón no es diferenciable en todos los puntos. Es una función lineal continua por partes. De hecho, escribámoslo de una manera ligeramente diferente:

$$J(\theta) = \Big(- \sum_{n:x_n \in \gamma} y_n x_n^T \Big) \theta \tag{2.4}$$

Esta es una función lineal con respecto a θ, siempre y cuando número de datos mal clasificados se mantenga estable. Sin embargo, conforme existan pequeños cambios en θ, el cual corresponde a un cambio en la posición del hiperplano respectivo, habrá un punto donde el número de elementos mal clasificados en γ cambiará; este será el momento en que un elemento en el conjunto de entrenamiento cambia su posición relativa con respecto al hiperplano (en movimiento) y como consecuencia el conjunto γ es modificado. Después de este cambio, $J(\theta)$ se ajustará a una nueva función.

El algoritmo del perceptrón

Se puede mostrar que, comenzando a partir de un punto arbitrario, $\theta^{(0)}$, la siguiente actualización de la iteración, converge después de un número finito de pasos. A continuación se muestra la regla del perceptrón.

$$\theta^{(i)} = \theta^{(i-1)} + \mu_i \sum_{n:x_n \in \gamma} y_n x_n \tag{2.5}$$

A parte de la ecuación 2.5, existe otra versión del algoritmo que considera un elemento por iteración de manera cíclica, esto se realiza hasta que el algoritmo converja. Denotemos por $y_{(i)}, x_{(i)},\ (i) \in 1, 2, \ldots, N$, el par de entrenamiento que se presenta en el

algoritmo en el i-ésimo paso de iteración. Entonces la actualización de cada iteración estaría dada como:

$$\theta^{(i)} = \begin{cases} \theta^{(i-1)} + \mu_i y_{(i)} x_{(i)}, \text{ sí } x_{(i)} \text{ es mal clasificado por } \theta^{(i-1)} \\ \theta^{(i-1)}, \text{ de otra manera.} \end{cases} \tag{2.6}$$

En otras palabras, comenzando a partir de una estimación inicial, por ejemplo, una inicialización aleatoria de $\theta^{(0)}$, se aplica un test a cada una de las muestras, x_n, $n = 1, 2, \ldots, N$. Cada vez que una muestra es clasificada erróneamente, se realiza una acción para su corrección. De otra manera, ni una acción será necesaria. Una vez todas las muestras fueron consideradas, se dice que una *época* ha finalizado. Si no converge, todos los elementos de la muestra se consideran para una segunda época, y así sucesivamente. Esta versión se conoce como un esquema *patrón a patrón*.

Después de una sucesión finita de épocas, el algoritmo garantiza la convergencia. Note que para la convergencia, la secuencia μ_i debe ser escogida de manera apropiada. Aún así, para el caso del algoritmo del perceptrón, la convergencia está garantizada aunque μ_i sea una constante positiva, $\mu_i = \mu > 0$.

La formulación en 2.6 le da al algoritmo del perceptrón la filosofía de aprendizaje de *recompensa y castigo*. Si la estimación tiene éxito al predecir la clase del patrón respectivo, ni una acción se ejecuta (recompensa). De otra manera, el algoritmo se castiga para realizar una actualización.

La figura 2.1 nos brinda una interpretación geométrica de la regla del perceptrón. Asumamos que la muestra x se clasifica erróneamente por el hiperplano, $\theta^{(i-1)}$ (línea roja). $\theta^{(i-1)}$ corresponde al vector que es perpendicular al hiperplano que está definido por este vector. x se encuentra en la parte $(-)$ del hiperplano y no está clasificada correctamente, pertenece a la clase ω_1. Por lo tanto, asumiendo $\mu = 1$, la corrección aplicada por el algoritmo es:

$$\theta^{(i)} = \theta^{(i-1)} + x$$

Su efecto será girar el hiperplano a la dirección de x, para colocarse en la parte $(+)$ del nuevo hiperplano, el cual está definido por la actualización definida por $\theta^{(i)}$

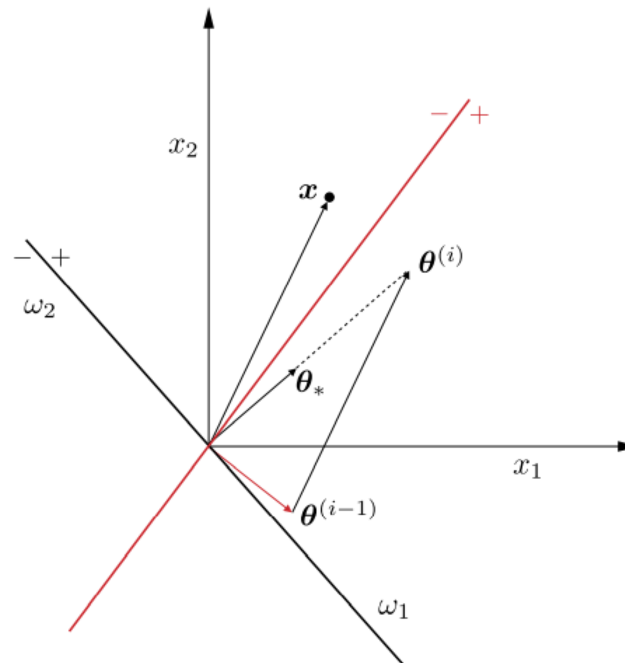

FIGURA 2.1: Interpretación Geométrica del Perceptrón[35]

El algoritmo del perceptron es una operación de tipo patrón por patrón, a continuación se resume este:

Algoritmo 1: Algoritmo del perceptrón

Inicialización:
$\theta^{(0)}$ =valor aleatorio pequeño;
/* Por lo general μ es igual a 1 */
μ=1;
i=1;
/* Cada iteración es una época */
do
  /* Cuenta el número de actualizaciones por época */
  counter=0;
  for $n = 1$ **to** N **do**
    if $y_n x_n^T \theta^{(i-1)} \leq 0$ **then**
      $\theta^{(i)} = \theta^{(i-1)} + \mu y_n x_n$;
      i=i+1;
      counter = counter +1;
    end
  end
while *counter!=0*;

Una vez el algoritmo se ejecute y converja, tendremos los pesos, $\theta_i, i = 1, 2, \ldots, l$, de la sinapsis de la neurona/perceptrón asociada(o), así como el término del bias θ_0. Esta ahora se puede usar para clasificar patrones desconocidos. La figura 2.2**A** nos muestra la correspondiente arquitectura de una neurona básica. Las características x_i, $i = 1, 2, \ldots, l$, son introducidos en los nodos de entrada. A su vez, cada característica es multiplicada por la sinapsis (peso) correspondiente, y después se suma el bias en la combinación lineal. La salida de esta operación pasa a través de una función no linear, f, conocida como la *función de activación*. Dependiendo de la forma de la no-linealidad, diferentes tipos de neuronas se activan. La forma más básica, conocida como la neurona de McCulloch–Pitts, la función de activación es la Heaviside, que es:

$$y_n = \begin{cases} 1, \text{ sí } z > 0 \\ 0, \text{ sí } x \leq 0 \end{cases} \tag{2.7}$$

Usualmente, la sumatoria y la no-linealidad se combinan para formar un nodo en la figura 2.2**B** se puede ver este caso. Entonces, el modelo básico de una NN, comprende la concatenación de (a) una combinación linear, (b) un valor de umbral (bias), y (c) la no-linealidad. Note que por la existencia de la no-linealidad, la salida de la neurona indica la clase de la que se origina el patrón de entrada. Para la no-linealidad de Heaviside, la salida es 1 para patrones de la clase ω_1 y 0 para los pertenecientes de la clase ω_2.

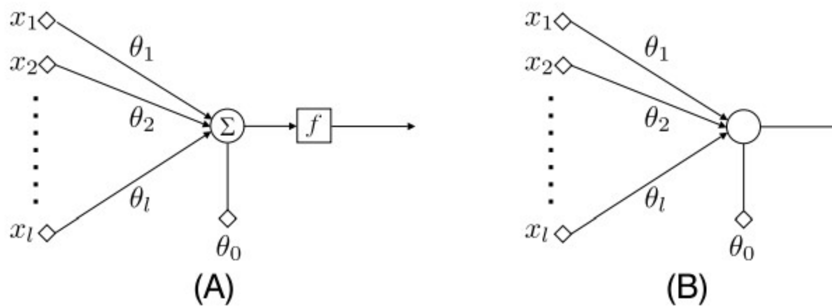

FIGURA 2.2: Arquitectura básica de un perceptrón[35]

2.1.2. Feedforward Networks (Fully Connected Networks)

Una simple neurona está asociada con un hiperplano en el espacio de las entradas (las características).

$$H : \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_l x_l + \theta_0 = 0 \tag{2.8}$$

Además, la clasificación se realiza a partir de la no-linealidad, del cual se obtendrá un 1 o se mantendrá en cero, dependiendo en que parte de H se encuentre el punto.

Como punto de partida, consideramos el caso en donde las clases en el espacio de características están formadas por las uniones de regiones poliédricas. Esto se muestra en la figura 2.3, para el caso de dos dimensiones. Estas regiones poliédricas están formadas por intersecciones de otros sub-espacios, cada uno asociado con un sub-espacio, En la figura 2.3, hay tres hiperplanos (líneas rectas en $\mathbf{R}^2$), indicadas como H_1, H_2 y H_3, dando lugar a 7 regiones poliédricas. Para cada hiperplano, las partes $(+)$ y $(-)$ (sub-espacios), están indicados. Cada una de las regiones se etiqueta mediante un triplete de números binarios, según de qué lado se ubique con respecto a H_1, H_2, H_3. Por ejemplo, la región etiquetada como (101) se encuentra en el lado $(+)$ de H_1, el lado $(-)$de H_2 y el lado $(+)$ de H_3.

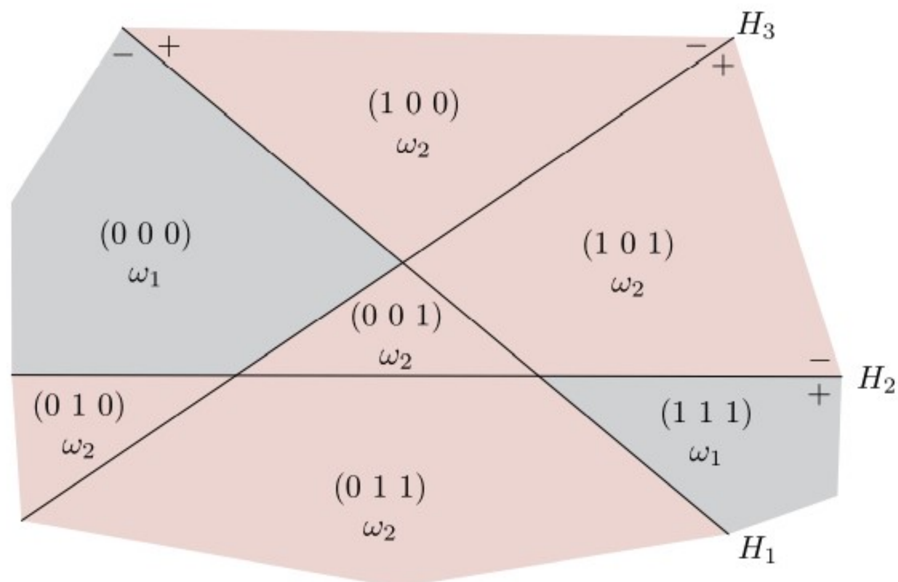

FIGURA 2.3: Las clases están formadas por uniones de regiones poliédricas.[35]

La figura 2.4, muestra tres neuronas clasificando tres hiperplanos, H_1, H_2 y H_3, mostrados en la figura 2.3. Las respectivas salidas, denotadas como y_1, y_2 y y_3, forman la etiqueta de la región de donde el patrón de entrada pertenece. De hecho, sí los pesos de las sinapsis fueron apropiadamente ajustados, entonces si un patrón se origina de la

región, digamos, (010), entonces la primer neurona de la izquierda introducirá un cero ($y_1 = 0$), la de en medio un uno ($y_2 = 1$) y el último un cero ($y_3 = 0$). En otras palabras, combinando las salidas de las tres neuronas juntas, habríamos logrado un mapeo del espacio de los datos de entrada al espacio tridimensional. Más específicamente, el mapeo se ejecuta en los vértices del cubo en $\mathbf{R}^3$, como se muestra en la figura 2.5. Cada región del espacio de los datos de entrada corresponden únicamente a un vértice del cubo. En el caso más general, donde se usan p neuronas, el mapeo va a realizarse en los vértices en la unidad del hipercubo en $\mathbf{R}^p$. Esta capa de neuronas comprende la primer *capa oculta* de la red.

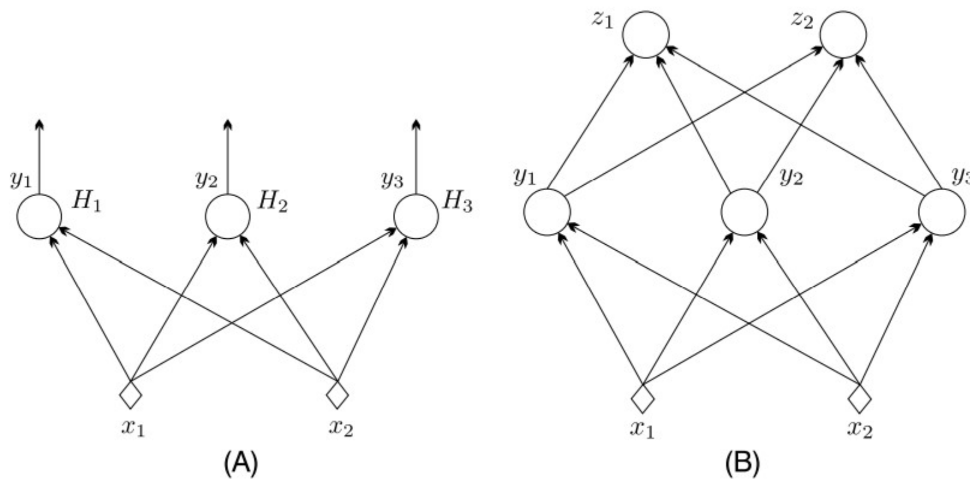

FIGURA 2.4: Ejemplo de 3 neuronas clasificando tres hiperplanos .[35]

Una forma alternativa de ver este mapeo es como una nueva representación de los patrones de entrada en términos de palabras de código. Para tres neuronas/hiperplanos podemos formar 2^3 códigos binarios, cada uno correspondiendo un vértice del cubo unitario, el cual puede representar $2^3 - 1 = 7$ regiones. Tenga en cuenta, sin embargo, que este mapeo codifica información en relación con alguna estructura de los datos de entrada; es decir, la información relativa a cómo los patrones de entrada son agrupados en el espacio de características en diferentes regiones.

Ahora usaremos esta nueva representación, ya que es proporcionada por las salidas de las neuronas de la primera capa oculta, como entrada que alimenta las neuronas de una segunda capa oculta, que se construye de la siguiente manera. Elegimos todas las regiones que pertenecen a una clase. Por el bien de nuestro ejemplo, en la figura 2.3, seleccionaremos las dos regiones que corresponden a la clase w_1, que son (000) y (111).

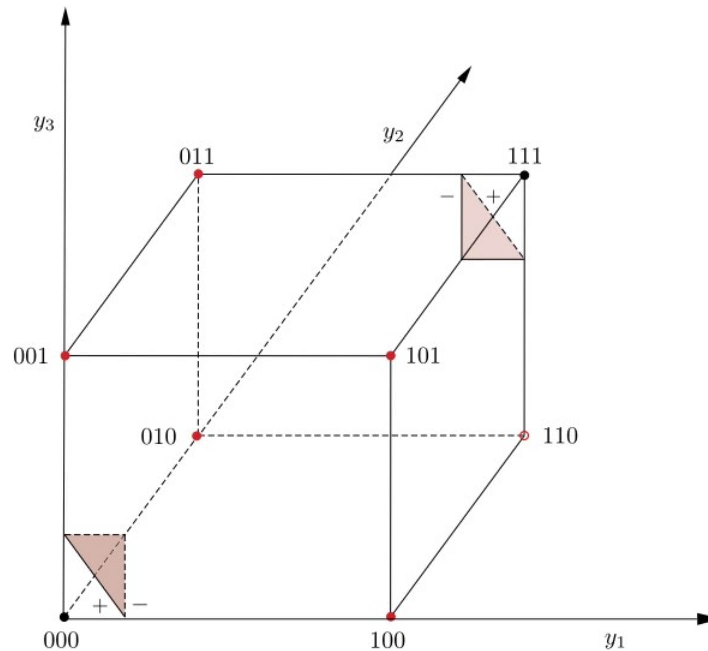

FIGURA 2.5: Las neuronas de la primera capa oculta realizan un mapeo desde el espacio de la entidad de entrada hasta los vértices de un hipercubo unitario.[35]

Recuerde que todos los puntos de estas regiones se asignan a los vértices respectivos del cubo unitario en el $\mathbf{R^3}$. Sin embargo, en este nuevo espacio transformado, cada uno de los vértices es *linealmente separable* del resto. Esto significa que podemos usar una neurona/perceptron en el espacio transformado, que colocará un único vértice en el lado $(+)$ y el resto en el $(-)$ del hiperplano asociado. Esto se muestra en la figura 2.5, donde los dos planos se muestran, los cuales separan los respectivos vértices de los demás. Cada uno de estos planos se trabaja por una neurona, operando en $\mathbf{R^3}$, como se ve el la figura 2.4**B**, donde una segunda capa de neuronas *escondidas* se han agregado.

Note que la salida z_1 de la neurona de la izquierda va a disparar un 1 sólo sí el patrón de entrada se origina de la región (000) y será un cero para todos los demás patrones. Para la neurona de la derecha, la salida z_2 será un uno sí todos los patrones vienen de la región (111) y cero, sí proviene de las restantes. Hay que ver que esta segunda capa de neuronas ejecuta un segundo mapeo, esta vez en los vértices del rectángulo unitario en $\mathbf{R^2}$. Este mapeo nos da una nueva representación de los patrones de entrada, y esta representación codifica información relacionada a las regiones de las clases. La figura 2.6 muestra el mapeo a los vértices del rectángulo unitario en el espacio z_1, z_2.

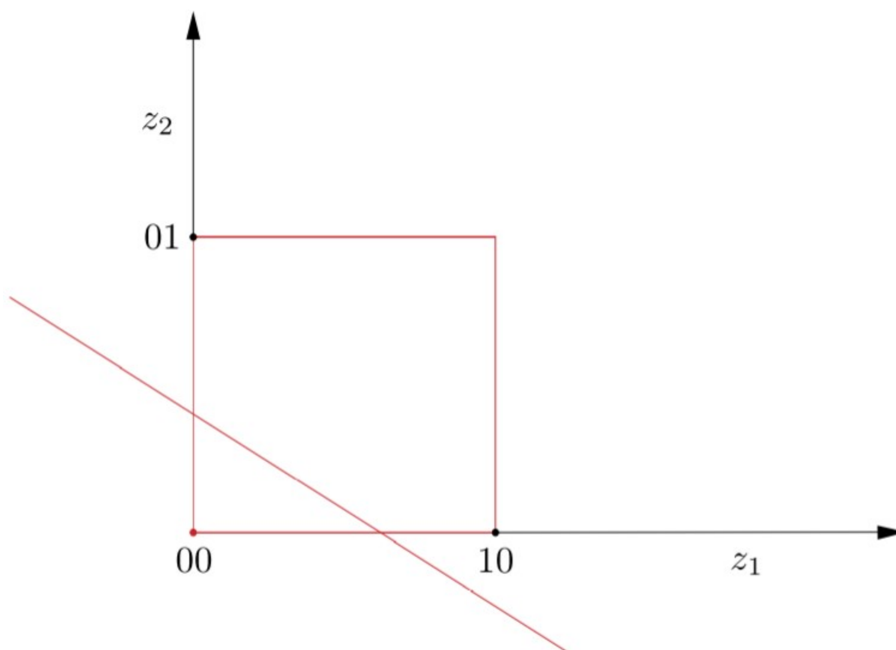

FIGURA 2.6: Los patrones de la clase w_1 son mapeados ya sea en (01) o (10) y los patrones de la clase w_2, todos serán mapeados en (00).[35]

Note que todos los puntos originados por la clase w_2, son mapeados en (00), y los puntos de la clase w_1 son mapeados ya sea en (10) o (01). Esto es interesante, hemos transformado nuestra tarea no-linear y separable, a una que es linealmente separable, estro por medio de los mapeos sucesivos. De hecho, el punto (00) puede ser linealmente separado de (01) y (10), por medio de una neurona extra operando en el espacio de (z_1, z_2). Este es conocido como la *neurona de salida*, porque esta nos brinda la decisión final del modelo. La neurona final se muestra en la figura 2.7. Esta red se llama *feed-forward*, porque la información fluye hacia adelante a partir de la capa de la entrada hacia la capa de la salida. Este modelo comprende la capa de entrada, que no es de procesamiento, dos capas ocultas (el término ”oculto” se explica por sí mismo) y una capa de salida. A esta red neuronal, la llamamos red de tres capas, esto sin contar la capa de la entrada ya que no procesa datos.

Hemos demostrado que una red feed-forward de 3 capas, puede resolver cualquier tarea de clasificación, donde todas las clases están conformadas por uniones de regiones de poliedros. A pesar de que nos enfocamos en un caso de dos clases, la generalización a multiclases es sencilla, esto se lograría aplicando más neuronas de salida dependiendo del número de clases. Note que en algunos casos, una capa oculta de nodos podría

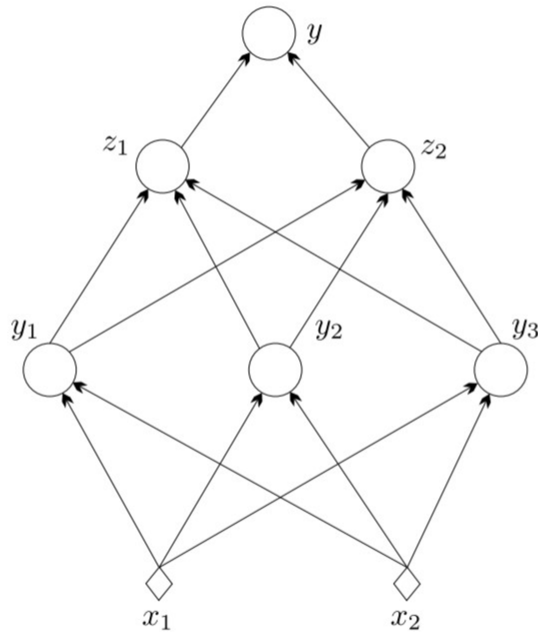

FIGURA 2.7: Modelo neuronal feed-forward de 3 capas.[35]

ser suficiente. Esto depende de si los vértices en los que se asignan las regiones se asignan a clases para que sea posible la separabilidad lineal. Por ejemplo, este sería el caso sí la clase w_1 fuera la unión de las regiones (000) y (100). Entonces, estos dos vértices podrían ser separados del resto por medio de un simple plano y una segunda capa oculta de neuronas no sería necesaria. Además, en la vida real, las clases no están necesariamente formadas por la unión de regiones poliédricas y las clases más importantes se superponen. Por lo tanto, es necesario diseñar un procedimiento de capacitación basado en una función de costos y un conjunto de datos de entrenamiento.

Nuestro enfoque se centrará en buscar formas de estimar los pesos desconocidos de las sinapsis y los sesgos de las neuronas. Sin embargo, desde un punto de vista conceptual, debemos recordar que cada capa realiza un mapeo en un nuevo espacio, y cada mapeo proporciona una representación diferente, con suerte, más informativa, de los datos de entrada, hasta la última capa, donde la tarea se ha transformado en uno que es fácil de resolver.

Las redes feed-forward, también conocidas como *fully connected networks*, son redes donde cada una de sus las neuronas/nodos en cualquier capa están conectados directamente a cada nodo de la capa anterior. Los nodos de la primera capa oculta están

completamente (fully, en inglés) conectados a los de la capa de entrada. En otras palabras, cada neurona está asociada a un vector de parámetros, cuya dimensión es igual al número de nodos de la capa anterior (entrada). Las operaciones algebraicas que se realizan son por medio del producto punto.

Para resumir de una manera más formal el tipo de operaciones que tienen lugar en una red completamente conectada, centrémonos en, digamos, la *r*-ésima capa de una red neuronal multicapa y supongamos que comprende k neuronas. El vector de entrada a esta capa, consiste de la salida de los nodos de la capa previa, denotado como y^{r-1}. Sea θ_j^r el vector de pesos, incluyendo el bias, asociado con la *j*-ésima neurona de la *r*-ésima capa, donde $j = 1, 2, \ldots, k_r$. La dimensión respectiva es $k_{r-1} + 1$, dónde k_{r-1} es el número de neuronas previas de la capa previa $r - 1$, e incrementa por uno por el bias. Después, las operaciones realizadas previo a la no-linealidad, son los productos punto.

$$z_j^r = \theta_j^{rT} y^{r-1}, j = 1, 2, \ldots, k_r \tag{2.9}$$

Recolectando todos los valores de salida en un vector, $z^r = [z_1^r, z_2^r, \ldots, z_{k_r}^r]$, y uniendo todos los vectores sinápticos como filas, uno debajo del otro, en una matriz Θ, que, podemos escribir colectivamente:

$$z^r = \Theta y^{r-1}, \text{donde } \Theta := [\theta_1^r, \theta_2^r, \ldots, \theta_{k_r}^r]^r \tag{2.10}$$

El vector de la salida de la *r*-ésima capa oculta, después de empujar cada z_i^r a través de la no-linealidad f, es finalmente dada por:

$$y^r = \begin{bmatrix} 1 \\ f(z^r) \end{bmatrix} \tag{2.11}$$

Donde la notación de la ecuación 2.11, significa que f actúa individualmente sobre cada uno de los componentes del vector, y la extensión del vector por uno es para contar el término bias.

Para redes largas, con muchas capas y nodos por capa, este tipo de conectividad termina siendo muy costosa en términos del número de parámetros (pesos), el cual es del orden $k_r k_{r-1}$. Una gran cantidad de parámetros hace que una red sea vulnerable al sobre-entrenamiento.

2.1.3. El algoritmo Backpropagation

Una red neuronal feed-forward consiste de un número de capas y neuronas, donde cada neurona tiene su propio conjunto de pesos de sinapsis y su bias. Desde este punto de vista, una red neuronal realiza a una función paramétrica no-lineal, $\hat{y} = f_\theta(x)$, donde θ representa todos los pesos y bias presentes en la red. Por lo tanto, entrenar una red neuronal no parece ser tan diferente al entrenamiento de cualquier modelo paramétrico de predicción. Todo lo que se necesita es, (a) un conjunto de datos de entrenamiento, (b) una función de costo, $\mathcal{L}(y, \hat{y})$ y (c) un esquema iterativo, por ejemplo, el gradiente descendiente, para realizar la optimización asociado a la función de costo:

$$J(\theta) = \sum_{n=1}^{N} \mathcal{L}(y_n, f_\theta(x_n)) \tag{2.12}$$

La dificultad de entrenar redes neuronales se encuentra en su estructura multicapa que complica los cálculos de la obtención del gradiente, el cual consta de su optimización. Además, la neurona de McCulloch–Pitts, se basa en la función discontinua Heaviside, la cual no es diferenciable. Un primer paso para el desarrollo de un algoritmo práctico para entrenar una red neuronal, es por medio del remplazo de la función de activación Heaviside, con otra función aproximada a esta.

La neurona *sigmoidea logística*: Una posibilidad es usar la función logística sigmoidea, que es:

$$f(z) = \sigma(z) := \frac{1}{1 + exp(-az)} \tag{2.13}$$

La gráfica de la función se muestra en la figura 2.8. Note que entre más largo el valor del parámetro a sea, lo más cercano la gráfica será a la función Heaviside. Otra posibilidad sería usar:

$$f(z) = a\tanh(\frac{cz}{2}) \tag{2.14}$$

Donde c y a son parámetros de control. La gráfica de esta función se muestra en la figura 2.9. Note que en contraste con la función logística sigmoidea, esta es una función antisimétrica, que es, $f(-z) = -f(z)$. Ambas son conocidas. Ambas también se conocen como funciones de aplastamiento porque limitan la salida a un rango finito de valores.

Una dificultad cuando se trata de minimizar la función de costo, como la ecuación 2.12, en el marco de las redes neuronales, es su *no convexidad*. Si la función no es convexa, un punto estacionario puede pertenecer a una de las siguientes tres categorías: (a)

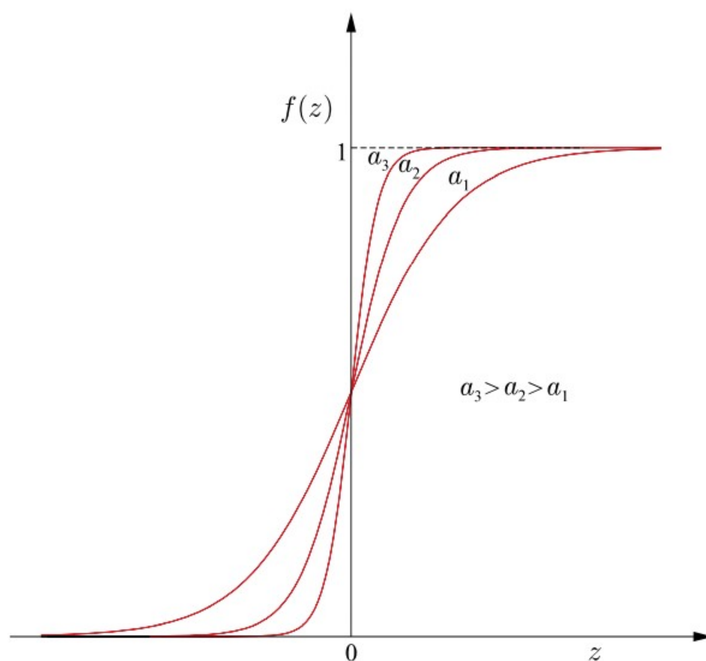

FIGURA 2.8: Función sigmoidea.[35]

ser un mínimo (máximo) local, (b) un mínimo (máximo) global, o (c) un punto silla; vea la figura 2.10 como ejemplo de una función no convexa en el espacio unidimensional. Todos estos puntos estacionarios son de importancia en redes neuronales. Un mínimo local es el punto donde el valor del costo $J(\theta_g) \leq J(\theta)$, $\forall\theta \in \mathbf{R}$. Un punto silla, no es ni un mínimo ni máximo, sin embargo, la derivada (gradiente en general) es igual a cero.

Cuándo se adopta un esquema de gradiente descendiente para minimizar una función de costo no convexa, el algoritmo puede que converja en un mínimo local o global. En problemas reales, que abarcan espacios multidimensionales, el número de mínimos locales puede ser muy largo, haciendo posible que el algoritmo converja en un mínimo local. Sin embargo, estas no son malas noticias. Si este mínimo local es lo suficientemente profundo, eso es, si el valor de la función de costo, por ejemplo, $J(\theta_l)$, no es tan largo como el obtenido en el mínimo global, por ejemplo, $J(\theta_g)$, la convergencia a tal mínimo local puede corresponder a una solución viable. En la practica, uno tiene que ser cuidadoso en como inicializar un algoritmo cuando se trata con espacios no convexos.

Una vez teniendo una función de activación diferenciable, estamos listos para desarrollar nuestro esquema iterativo del gradiente descendiente. Sea (y_n, x_n), $n = 1, 2, \ldots, N$, el número de muestras de entrenamiento. Note que hemos asumido que hay múltiples

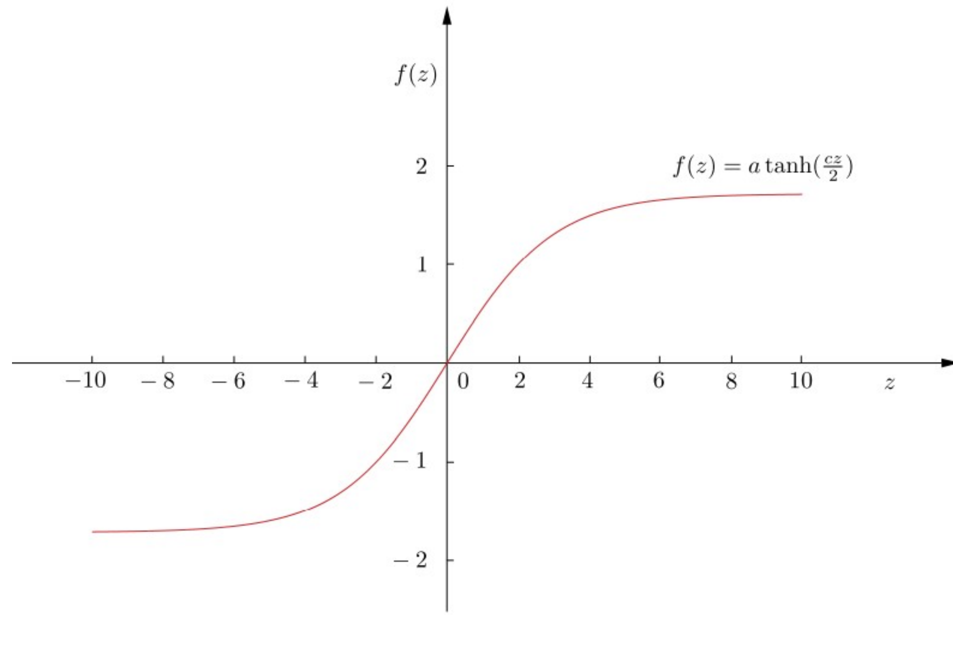

FIGURA 2.9: Función tanh.[35]

variables de salida, estos estarán ensamblados en un vector. Asumimos que la red comprende L capas; $L-1$ capas ocultas y una capa de salida. Cada capa va a consistir de k_r, $r = 1, 2, \ldots, L$, neuronas. Entonces, los vectores "deseados" de salida son:

$$y_n = [y_{n1}, y_{n2}, \ldots, y_{nk_L}]^T \in \mathbf{R}^{k_L}, n = 1, 2, \ldots, N \tag{2.15}$$

Por el bien de las derivaciones matemáticas, también denotamos el número de nodos de entrada como k_0; es decir, $k_0 = l$, donde l es la dimensionalidad del espacio de características de entrada.

Sea θ_j^r el vector de pesos sinápticos asociado con la j-ésima neurona en la r-ésima capa, con $j = 1, 2, \ldots, k_r$ y $r = 1, 2, \ldots, L$, donde el bias se incluye en θ_j^r, eso es:

$$\theta_j^r := [\theta_{j0}^r, \theta_{j1}^r, \ldots, \theta_{jk_{r-1}}^r]^T \tag{2.16}$$

Los pesos sinápticos unen la respectiva neurona con todas las neuronas en la capa k_{r-1} (vea la figura 2.11). El paso iterativo básico del gradiente descendiente es:

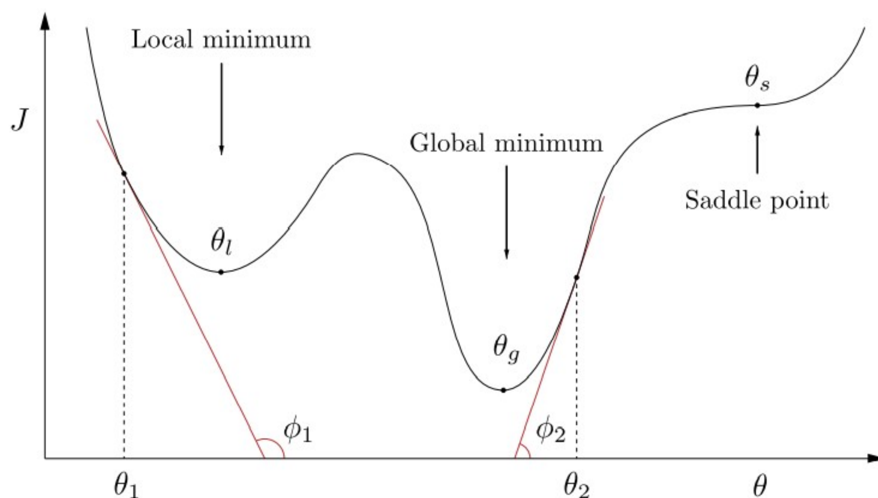

FIGURA 2.10: Ejemplo de función no convexa.[35]

$$\theta_j^r(nuevo) = \theta_j^r(viejo) + \delta\theta_j^r \tag{2.17}$$

$$\delta\theta_j^r := -\mu \frac{\partial J}{\partial \theta_j^r}\bigg|_{\theta_j^r(viejo)} \tag{2.18}$$

El parámetro μ es el tamaño del paso, definido por el usuario (también puede ser dependiente a las iteraciones) y J denota la función de costo.

La actualización de las ecuaciones 2.17 y 2.18 comprenden el par del esquema de descenso de gradiente para la optimización. Como se dijo anteriormente, la dificultad en las feed-forward networks surge de su estructura de múltiples capas. En orden de obtener el gradiente en la ecuación 2.18, para todas las neuronas en todas las capas, uno tiene que seguir dos pasos.

- *Cálculos hacia adelante*: Para un vector dado de entrada x_n, $n = 1, 2, \ldots, N$, se utilizan las estimaciones actuales de los parámetros (pesos sinápticos) (θ_j^r(viejo)) y se calculan todas las salidas de todas las neuronas en todas las capas, denotadas como y_{nj}^r.

- *Cálculos hacia atrás*: Usando las salidas del paso de arriba junto con los valores conocidos de las salidas, y_{nk}, de la capa de salida, calcula los gradientes de la función de costo. Esto involucra L pasos, esto es, tantas como el número de capas. La secuencia de los pasos del algoritmo se brindan más adelante:

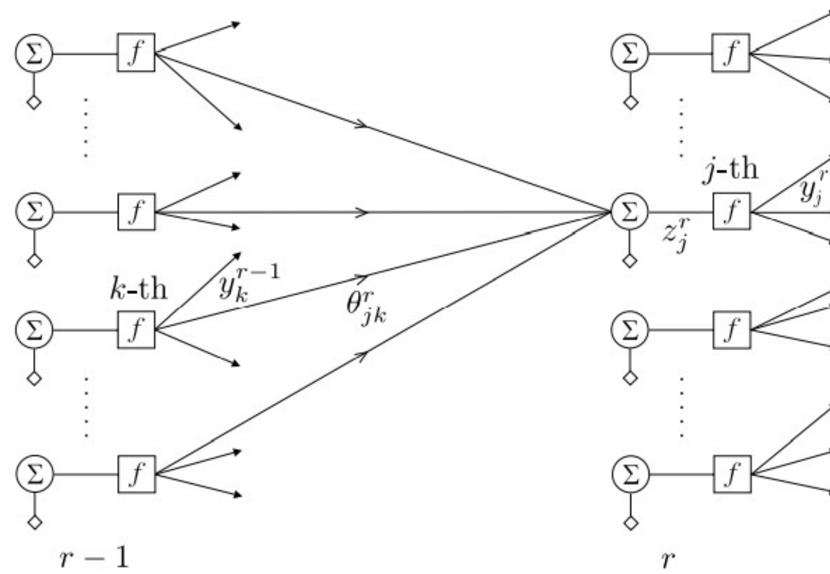

FIGURA 2.11: Ejemplo de la conexión de una neurona con las demás.[35]

- Calcula el gradiente de la función de costo con respecto a los parámetros de las neuronas de la última capa. $\frac{\partial J}{\partial \theta_j^L}$, $j = 1, 2, \ldots, k_r$.

- ***for*** $r = L - 1$ ***to*** 1, ***do***:
  - Calcular el gradiente con respecto a los parámetros asociados con las neuronas de la r-ésima capa, por ejemplo: $\frac{\partial J}{\partial \theta_k^r}$, $k = 1, 2, \ldots, k_r$, basado en todos los gradientes $\frac{\theta J}{\theta_j^{r+1}}$, $j = 1, 2, \ldots, k_{r+1}$, con respecto a los parámetros de la capa $r + 1$ que han sido calculados en el paso anterior.

- ***END for***

Los cálculos del paso *hacia atrás*, es una aplicación directa de la regla de la cadena, y comienza con el cálculo de las derivadas asociadas con la última capa (la salida). Después, el algoritmo ”fluye” hacia atrás por medio de la jerarquía de capas. Esto se debe a la naturaleza de la red multicapa, donde las salidas, capa tras capa, se forman como funciones de funciones. De hecho, vamos a enfocarnos en la salida y_k^r de la k-ésima neurona en la capa r. Entonces tendremos:

$$y_r^k = f\left(\theta_k^{r^T} y^{r-1}\right), k = 1, 2, \ldots, k_r \tag{2.19}$$

Donde y^{r-1} es el vector (extendido) que comprende todas las salidas de la capa anterior, $r-1$, y f denota la no-linealidad. Basado en lo visto arriba (ecuación 2.19), la salida de la j-ésima neurona en la siguiente capa esta dado como:

$$y_j^{r+1} = f\left(\theta^{r+1^T} y^r\right) = f\left(\theta^{r+1^T} \begin{bmatrix} 1 \\ f(\Theta^r y^{r-1}) \end{bmatrix}\right) \tag{2.20}$$

Donde $\Theta^r := [\theta_1^r, \theta_2^r, \ldots, \theta_{kr}]^T$ denota la matriz que tiene como filas el vector de pesos de la fila r. Esta estructura función-sobre-función-sobre-función es el subproducto de la naturaleza multicapa de las redes neuronales, y es una operación altamente no lineal que da lugar a la complicación para calcular los gradientes.

Sin embargo, uno puede detectar fácilmente que calcular los gradientes con respecto a los parámetros que definen la capa de salida no presenta ninguna dificultad. De hecho, la salida de la j-ésima neurona de la última capa (que en realidad es la estimación de salida de corriente respectiva) se escribe como:

$$\hat{y}_j := y_j^L = f(\theta_j^{L^T} y^{L-1}) \tag{2.21}$$

Como se conoce y^{L-1}, después de los cálculos durante el paso hacia adelante, tomar la derivada con respecto a θ^{L_j} es sencillo; aquí no interviene ninguna operación de función sobre función. Es por eso que comenzamos desde la capa superior y luego nos movemos hacia atrás.

A continuación daremos un resumen del algoritmo de backpropagation:

Algoritmo 2: Algoritmo backpropagation

Inicialización:

/* Inicializaremos todos los pesos sinápticos y bias de forma aleatoria con valores pequeños, pero no tan pequeños */

/* Seleccionaremos el tamaño del paso μ */

/* $j = 1, 2, \dots, k_0 := l$, $n = 1, 2, \dots, N$ */

$y^0_{nj} = x_{nj}$;

/* Cada repetición es una época */

do
- **for** $n = 1$ **to** N **do**
  - **for** $r = 1$ **to** L **do**
    - **for** $j = 1$ **to** k_r **do**
      - Calcular z^r_{nj} de la ecuación 2.22 ;
      - Calcular $y^r_{nj} = f(z^r_{nj})$;
    - **end**
  - **end**
  - **for** $j = 1$ **to** k_L **do**
    - /* cálculos backward (capa de salida) */
    - Calcular δ^L_{nj} de la ecuación 2.24;
  - **end**
  - **for** $r = L$ **to** 2 **do**
    - /* cálculos backward (capas escondidas) */
    - **for** $j = 1$ **to** k_r **do**
      - Calcular δ^{r-1}_{nj} de la ecuación 2.25;
    - **end**
  - **end**
- **end**
- **for** $r = 1$ **to** L **do**
  - /* actualizar los pesos */
  - **for** $j = 1$ **to** k_r **do**
    - Calcular $\Delta\theta^r_j$ de la ecuación 2.26;
    - $\theta^r_j = \theta^r_j + \Delta\theta^r_j$
  - **end**
- **end**

while *Hasta que la condición de paro se alcance*;

Las ecuaciones usadas en el algoritmo 2, se en listan a continuación:

- *Cálculo del gradiente:* sea z_{nj}^{r} denota la salida de la combinación lineal de la j-ésima neurona en la r-ésima capa en el instante n, cuando el patrón x_n es aplicado en el nodo de entrada (vea la figura 2.11). Entonces lo podremos escribir como:

$$z_{nj}^{r} = \sum_{m=1}^{k_{r-1}} \theta_{jm}^{r} y_{nm}^{r-1} + \theta_{j0}^{r} = \sum_{m=0}^{k_{r-1}} \theta_{jm}^{r} y_{nm}^{r-1} = \theta_j^{r^T} y_n^{r-1} \tag{2.22}$$

  Donde por definición:

$$y_n^{r-1} := [1, y_{n1}^{r-1}, \dots, y_{nk_{r-1}}^{r-1}]^T \tag{2.23}$$

  Y $y \equiv 1$, $\forall r$, n y θ_j^r han sido definidas en la ecuación 2.16. Para las neuronas en la capa de salida, $r = L$, y_{nm}^{L}, $m = 1, 2, \dots, k_L$, y para $r = 1$, tenemos $y_{nm}^{0} = x_{nm}$, $m = 1, 2, \dots, k_0$; eso es, y_{nm}^{0} es igual al conjunto de características de entrada.

- f' denota la derivada de f y e_{nj} es el error asociado con la j-ésima variable de salida en el tiempo n

$$\delta_{nj}^{L} = (\hat{y}_{nj} - y_{nj}) f'(z_{nj}^{L}) = e_{nj} f'(z_{nj}^{L}),\ j = 1, 2, \dots, k_L \tag{2.24}$$

- $$\delta_{nj}^{r-1} = e_{nj}^{r-1} f'(z_{nj}^{r-1}) \tag{2.25}$$

- $$\Delta\theta_j^{r} = -\mu \sum_{n=1}^{N} \delta_{nj}^{r} y_n^{r-1},\ r = 1, 2, \dots, L \tag{2.26}$$

Variantes del Gradiente Descendiente

Existen tres variantes del gradiente descendiente, estas difieren en la cantidad de datos usados para calcular el gradiente de la función de costo. Dependiendo de la cantidad de datos, hacemos un intercambio entre la precisión de la actualización de los parámetros y el tiempo que lleva realizar una actualización [37]. Estas son:

- Batch Gradient Descent: Calcula el gradiente de la función de costo con respecto a los parámetros θ, para el conjunto (completo) de datos de entrenamiento. Puede ser muy lento ya que una actualización de los pesos implica una época, y es

intratable para el caso de datos que no caben en memoria. Esta variante garantiza la convergencia a un mínimo global en una superficie no convexa, y un mínimo local para superficie convexas.

- Stochastic Gradient Desecnt (SGD): En contraste con el punto anterior, la actualización de los parámetros se aplica para cada elemento $x^{(i)}$ y $y^{(i)}$ del conjunto de entrenamiento. Por lo tanto, es usualmente más rápido, aunque las actualizaciones pueden variar, provocando que la función objetivo fluctúe mucho.
- Mini-batch Gradient Descent: Esta variante toma lo mejor de los puntos anteriores, realizando una actualización cada *mini-batch* de n ejemplares de entrenamiento. De esta manera se reduce la varianza de la actualización de los parámetros, la cual puede resultar en una convergencia más estable. Puede hacer uso de optimizaciones matriciales altamente optimizadas comunes a las bibliotecas de aprendizaje profundo de última generación que hacen que el cálculo del gradiente con respecto a un mini-batch sea muy eficiente. Un tamaño común de n para el mini-batch esta entre 50 y 36, este valor puede variar dependiendo de las aplicaciones.

2.1.4. Optimización

El esquema básico del gradiente descendiente hereda todas las ventajas (una baja demanda computacional por paso de iteración) y todas sus desventajas (una tasa de convergencia lenta). Para acelerar la tasa de convergencia, se ha invertido mucho esfuerzo de investigación y se ha propuesto una gran cantidad de variantes del esquema básico del gradiente descendiente-backpropagation. Su objetivo no sólo es minimizar la función de costo encontrando un valor óptimo para los pesos, sino que también se debe asegurar que el algoritmo generalice bien los datos. Existen diversos algoritmos variantes del esquema básico, como pueden ser:

- Momentum [38]: Es un método que ayuda a acelerar el SGD en la dirección relevante y amortigua las oscilaciones, como se puede ver en la figura 2.12
- Adagrad[39]: Es un algoritmo para la optimización basada en gradientes que hace precisamente esto: adapta la tasa de aprendizaje a los parámetros, realizando actualizaciones más grandes para actualizaciones poco frecuentes y más pequeñas para parámetros frecuentes. La principal debilidad de Adagrad es su acumulación de gradientes cuadrados en el denominador: dado que cada término agregado es positivo, la suma acumulada sigue creciendo durante el entrenamiento.

Esto, a su vez, hace que la tasa de aprendizaje se reduzca y, finalmente, se vuelva infinitesimalmente pequeña, momento en el que el algoritmo ya no puede adquirir conocimientos adicionales.

- Adadelta[40]: Es una extensión de Adagrad que busca reducir su tasa de aprendizaje agresiva y monótonamente decreciente. En lugar de acumular todos los gradientes cuadrados pasados, Adadelta restringe la ventana de gradientes pasados acumulados a un tamaño fijo w.

- Adaptative Moment Estimation (Adam)[41]: Es otro método que calcula las tasas de aprendizaje adaptativo para cada parámetro. Además de almacenar un promedio en declive exponencial de gradientes pasados al cuadrado v_t como Adadelta, Adam también mantiene un promedio en declive exponencial de gradientes pasados m_t, similar al Momentum.

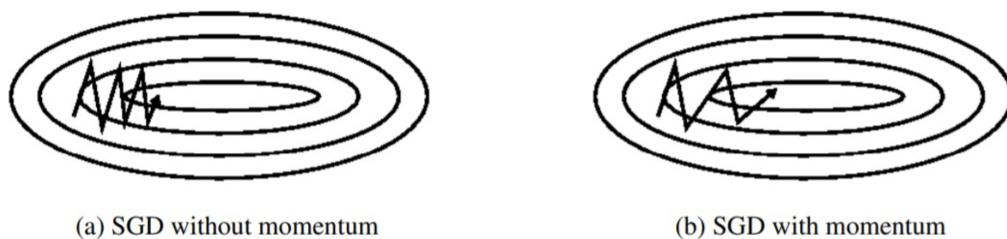

FIGURA 2.12: Ejemplo de la oscilación con y sin algoritmo Momentum.[37]

En [37], se establece que el mejor optimizador, dentro de los estudiados en dicho artículo, es el algoritmo Adam. Como se mencionó al inicio de este apartado, desarrollar nuevos algoritmos de optimización es importante para la obtención de mejores resultados (de convergencia) en poco tiempo. Por el momento el algoritmo Adam es el más usado, pero es cuestión de tiempo para que se desarrollen mejores algoritmos que superen a este, como por ejemplo, en el año en que se escribe esta tesis, [42] propone un nuevo algoritmo que dice supera al algoritmo Adam, este aplicado en CNN's.

2.1.5. Evaluación del Desempeño

Seleccionar una métrica es crucial cuando se evalúa un modelo de ML, es un paso necesario porque esta métrica será una guía para las futuras acciones a tomar sobre el algoritmo. Hay que tomar en cuenta que no siempre es posible alcanzar un error

de 0. El error de Bayes define la tasa de error mínima que se puede esperar lograr, incluso si tiene datos de entrenamiento infinitos y puede recuperar la distribución de probabilidad real. Esto se debe a que sus características de entrada pueden no contener información completa sobre la variable de salida, o porque el sistema puede ser intrínsecamente estocástico. También estará limitado por tener una cantidad finita de datos de entrenamiento.

La cantidad de datos de entrenamiento puede ser limitada por varias razones. Cuando su objetivo es crear el mejor producto o servicio posible, normalmente puede recopilar más datos, pero debe determinar el valor de reducir aún más el error y sopesar esto con el costo de recopilar más datos. La recopilación de datos puede requerir tiempo, dinero o sufrimiento humano (por ejemplo, si su proceso de recopilación de datos implica la realización de pruebas médicas invasivas). Cuando su objetivo es responder una pregunta científica sobre qué algoritmo funciona mejor en un punto de referencia fijo, la especificación del punto de referencia generalmente determina el conjunto de entrenamiento y no se le permite recopilar más datos.

Por lo general, en el entorno académico, tenemos algunas estimaciones de la tasa de error que se puede alcanzar en función de los resultados obtenidos en otros artículos. Una vez que se haya determinado su tasa de error deseada realista, las decisiones de diseño se guiarán por alcanzar esta tasa de error. Otra consideración importante además del valor de la métrica de rendimiento es la elección de la métrica que se utilizará. Se pueden usar varias métricas de rendimiento diferentes para medir la efectividad de una aplicación completa que incluye componentes de aprendizaje automático. A continuación se dará una explicación sobre las métricas utilizadas en esta investigación.

- Matriz de Confusión: Esta no es una métrica, pero es un concepto clave para observar el desempeño del clasificador. Es una visualización tabular de las predicciones del modelo frente a las etiquetas de *verdad fundamental*. Cada fila de la matriz de confusión representa las instancias en una clase predicha y cada columna representa las instancias en una clase real.

- Accuracy: Es una de las métricas más simples, y está definida como el número de predicciones correctas dividido entre el número total de predicciones.

$$acc = \frac{TP}{n} \tag{2.27}$$

- Precisión: Hay varios casos en las que la métrica $accuracy$ no es un buen indicador para medir el desempeño del modelo. Uno de estos casos aplica cuando el

conjunto de datos no esta balanceado. Por lo tanto, también debemos analizar las métricas de rendimiento específicas de la clase. Precisión es una de las métricas que se encarga de esto, y se obtiene con la siguiente fórmula:

$$precision = \frac{TP}{TP + FP} \tag{2.28}$$

- Recall: Esta es otra métrica importante, que se define como la fracción de muestras de una clase que el modelo predice correctamente. Más formalmente:

$$recall = \frac{TP}{TP + FP} \tag{2.29}$$

- F1-Score: Dependiendo de la aplicación, es posible que desee dar mayor prioridad a la $recall$ o la métrica de precisión. Pero hay muchas aplicaciones en las que tanto $recall$ como precisión son importantes. Por lo tanto, es natural pensar en una forma de combinar estas dos en una sola métrica. Una métrica popular que las combina se llama F1-Score, que es la media armónica de precisión y $recall$ definida como:

$$F1 - Score = 2\frac{precision \times recall}{precision + recall} \tag{2.30}$$

- Proporción de verdaderos positivos (*TP rate / Sensitivity*): Esta es otra métrica popular, como su nombre lo dice, nos da la proporción de verdaderos positivos. La fórmula es la siguiente.

$$TPrate = \frac{TP}{TP + FN} \tag{2.31}$$

- Porción de falsos positivos (*FP rate / Specificity*): Al igual que el punto anterior, es una métrica popular que nos brinda la proporción de falsos positivos, su fórmula es la siguiente.

$$FPrate = \frac{TN}{TN + FP} \tag{2.32}$$

- Curva AUC-ROC: Esta métrica, cuenta con la unión de dos términos, el primer término es la curva *ROC* (Receiver Operating Characteristic) y el segundo es *AUC* (Area Under the Curve). A continuación se explicará cada uno.
  - ROC: Es un gráfico que muestra el rendimiento de un clasificador binario en función de su umbral de corte. Esencialmente, muestra la proporción de

verdaderos positivos contra la proporción de falsos positivos, para varios valores de umbral.

- AUC: Calcula el área debajo de la curva *ROC*, este es un valor entre 0 y 1. Una forma de interpretar el AUC es como la probabilidad de que el modelo clasifique un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio. Por lo tanto, cuanto mayor sea el AUC de un modelo, mejor será el desempeño del modelo.

2.2. Redes Convolucionales CNN

Una alternativa se desarrolló a finales de los 80's, cuando la fase de generación de características se integró como parte del entrenamiento de una NN. La idea era aprender las características de los datos junto con los parámetros de la red neuronal y no de forma independiente. Esta red neuronal se llamaría *Red Neuronal Convolucional* (CNN), y su primer éxito fue el reconocimiento de números (OCR) [43]. Las CNN's son un modelo especializado de redes neuronales para el procesamiento de datos con una topología de cuadrícula. El nombre *"Red Neuronal Convolucional"*, indica que la red utiliza una operación matemática conocida como *convolución*. La convolución es un tipo especializado de operación lineal. Las redes convolucionales son simplemente redes neuronales que utilizan la convolución en lugar de la multiplicación de matrices general en al menos una de sus capas.

En la práctica no podemos "alimentar" una red neuronal con datos en crudo, como puede ser una imagen, (primero se debería preprocesar la imagen para que de esta forma se generen características). El entrenamiento de redes neuronales con imágenes no es plausible, porque la imagen vectorizada para este modelo, involucraría millones de parámetros, este podría ser el caso para una imagen de 256×256 y con una primera capa con 1000 nodos. Y, sí el modelo entrenara con imágenes de alta resolución de una dimensión, el valor de cada parámetro sufriría una explosión numérica, sería lo mismo para el caso de una imagen de tres canales de color (RGB). Además, conforme se vayan agregando capas ocultas, el número de parámetros incrementaría. Esto implica que la generalización del modelo se vería afectada. Este tipo de modelos requieren de una gran cantidad de datos de entrenamiento, para evitar el sobre entrenamiento.

A parte de la explosión numérica que sufriría el modelo, la vectorización de una imagen daría como consecuencia perdida de información. Esto se debe a que con esta acción, se desecha información sobre como los píxeles están relacionados en el área de

la imagen. De hecho, el objetivo de las diversas técnicas de generación de características que se han desarrollado a lo largo de los años es exactamente ese. Es decir, extraer información que cuantifique correlaciones u otras dependencias estadísticas que relacionen valores de píxeles dentro de la imagen. De esta manera, se puede ”codificar” de manera eficiente la información relacionada con el aprendizaje que reside en los datos sin procesar.

Una capa de una CNN, consiste de tres etapas (ver figura 2.13). En la primera etapa, la capa realiza varias convoluciones en paralelo para producir un conjunto de activaciones lineales. En la segunda etapa, cada activación lineal se ejecuta a través de una función de activación no lineal, como la función *ReLU*. Esta etapa a veces se denomina etapa de detección. En la tercera etapa, usamos una función de agrupación (*Pooling*) para modificar aún más la salida de la capa.

Hay dos conjuntos de terminología de uso común para describir estas capas. (figura 2.13 izquierda) En esta terminología, la CNN se ve como una pequeña cantidad de capas relativamente complejas, y cada capa tiene muchas ”etapas”. (figura 2.13 derecha) En esta terminología, la CNN se ve como un número mayor de capas simples; cada paso del procesamiento se considera una capa por derecho propio.

La lectura del artículo [44] brindará conocimiento adicional para entender como trabaja y aprende una CNN.

2.2.1. La Operación Convolucional

De forma general, la convolución es una operación de dos funciones de valores tipo $\mathbf{R}$. Estaría dada de la siguiente manera:

$$g(y) = \int_{-\infty}^{\infty} x(a)w(t-a)da \tag{2.33}$$

En términos de una CNN, el primer argumento, la función x, va a ser la entrada de esta red, y la función w, será el **kernel**, la salida será el **mapa de características**. Ahora, si asumimos que x y w están definidas como enteros, podremos definir la convolución discreta:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.34}$$

Finalmente, a menudo usamos convoluciones en más de un eje a la vez (multidimensional). Por ejemplo, si usamos una imagen bidimensional I como nuestra entrada,

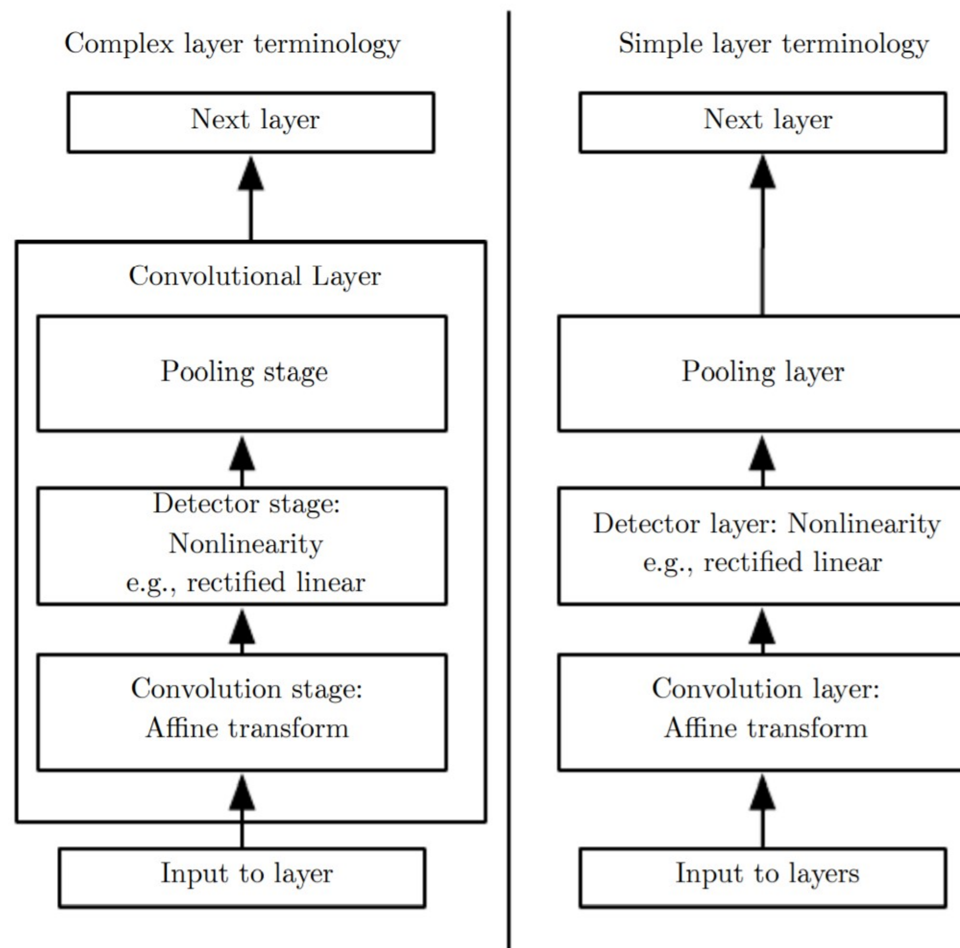

FIGURA 2.13: Terminología básica y compleja de la capa de una CNN.[36]

probablemente también queramos usar un kernel bidimensional, que denotaremos por h, será la capa oculta (**h**idden layer):

$$s(i,j) = (h * I)(i,j) = \sum_m \sum_n I(m,n)h(i-m, j-n) \tag{2.35}$$

En la figura 2.14 se muestra un ejemplo de convolución 2-D. Restringimos la salida solo a posiciones en las que el kernel se encuentra completamente dentro de la imagen, lo que se denomina convolución ”válida” en algunos contextos. Los cuadros con flechas indican cómo se forma el elemento superior izquierdo del tensor de salida aplicando el kernel a la región superior izquierda correspondiente del tensor de entrada.

Al realizar convoluciones, en lugar de realizar el producto de matrices, se tienen los

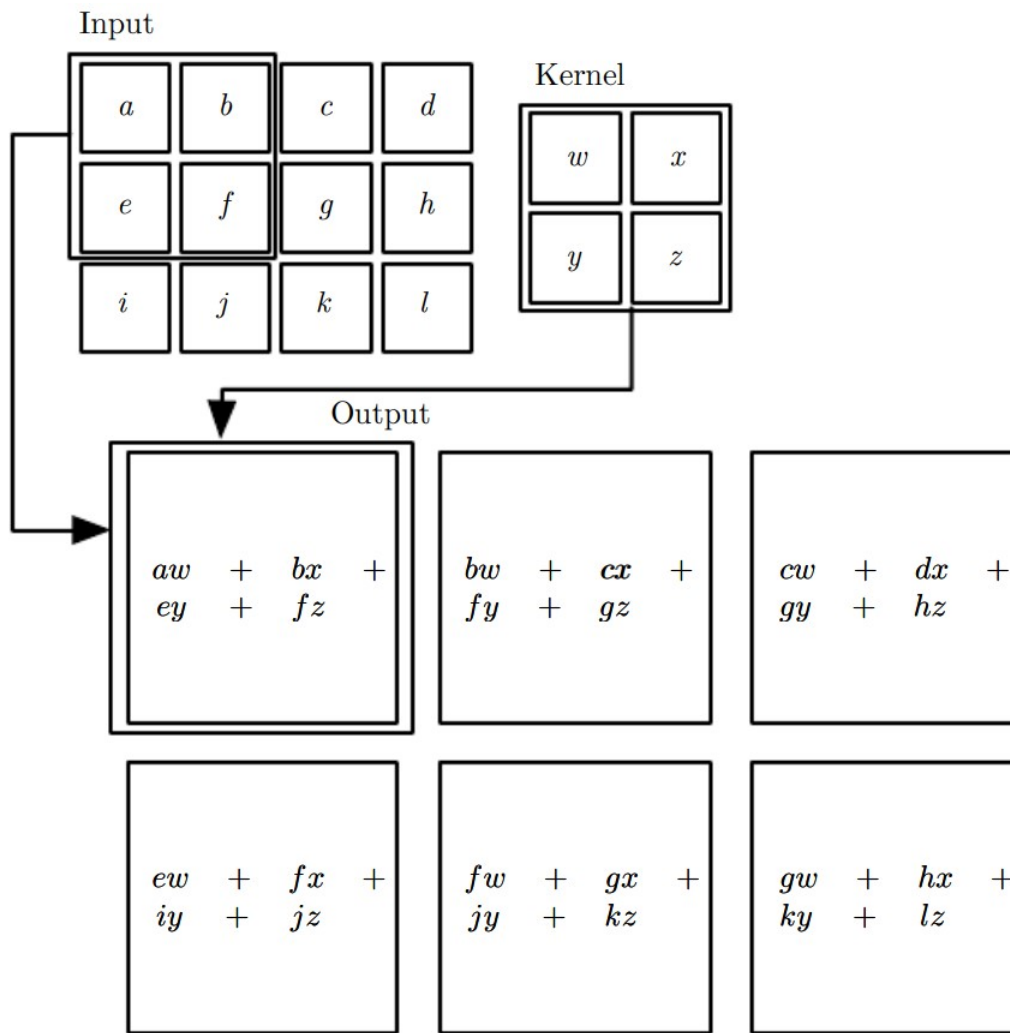

FIGURA 2.14: Ejemplo de convolución 2-D.[36]

siguientes resultados: (a) los parámetros que comprenden la capa oculta son compartidos por todos los píxeles de entrada y no tenemos un conjunto dedicado de parámetros por elemento de entrada (píxel), y (b) las salidas de la capa oculta codifican información de correlación de vecindad local de las diversas áreas dentro de la imagen de entrada. además, dado que la salida de la capa oculta también es una matriz de imágenes, se puede considerar como la entrada a una segunda capa oculta y de esta manera construir una red con muchas capas, cada una realizando convoluciones.

De hecho, estas operaciones de ”filtrado” se han utilizado tradicionalmente para generar características a partir de imágenes. La diferencia fue que los elementos de la matriz de filtros fueron preseleccionados. Tomen como ejemplo la siguiente matriz:

$$H = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.36}$$

El filtro mostrado en 2.36, se conoce como detector de bordes. Convolucionando una imagen I, con ese filtro, detectará los bordes de una imagen, como se puede observar en la figura 2.15. Donde la figura 2.15**A**, muestra la imagen antes de la operación de convolución, y la figura 2.15**B**, muestra el resultado después de esta operación.

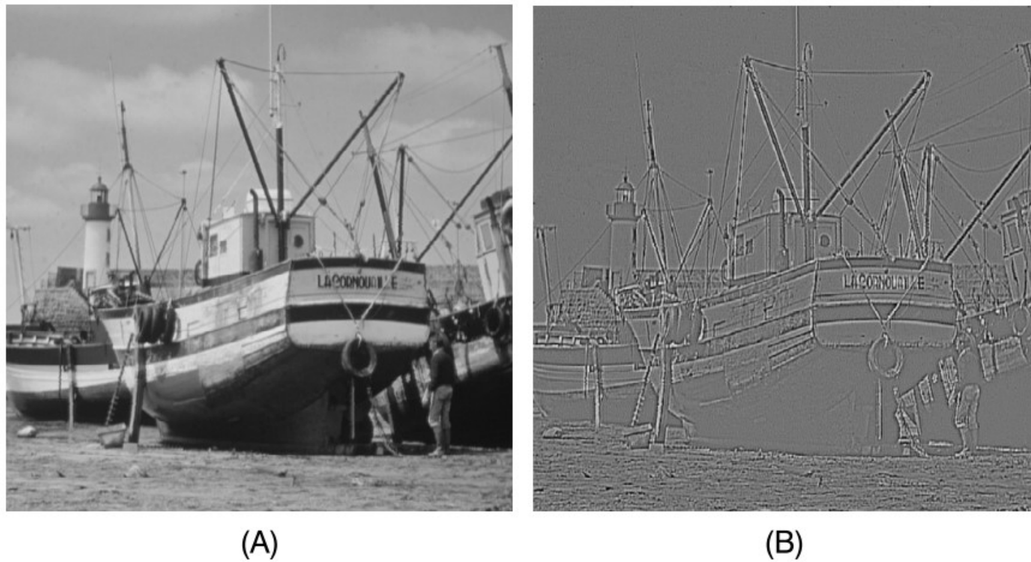

FIGURA 2.15: Ejemplo de aplicación de filtro de detección de bordes.[35]

La detección de bordes es de gran importancia para la comprensión de imágenes. Además, cambiando apropiadamente los valores en H, se pueden detectar bordes en diferentes orientaciones, por ejemplo, diagonal, vertical, horizontal; en otras palabras, cambiando los valores de H uno puede generar diferentes tipos de características. Así, nos acercamos a la idea detrás de las CNN.

- En lugar de utilizar una matriz de filtro / kernel fija, como en el ejemplo del detector de bordes, deje el cálculo de los valores de la matriz H, a la fase de entrenamiento. En otras palabras, hacemos que H sea adaptable a los datos y no preseleccionado.
- En lugar de utilizar una única matriz de filtro, se utiliza más de una. Cada uno de ellos generará un tipo diferente de características. Por ejemplo, uno puede generar

bordes diagonales, el otro horizontal, etc. Por tanto, cada capa oculta comprenderá más de una matriz de filtro. Los valores de los elementos de cada una de las matrices de filtro se calcularán durante la fase de entrenamiento, optimizando algún criterio. En otras palabras, cada capa oculta de una CNN genera un conjunto de características de manera óptima.

La figura 2.16 nos muestra la entrada de una imagen a la primera capa oculta de una CNN. La capa oculta consiste de un kernel de tres capas, llamados H_1, H_2 y H_3. Cada mapa de características, es el resultado de una convolución aplicada por cada una de las capas del kernel. Es por eso que cuanto más filtros se usen, mayor será la cantidad de características que se obtendrán y como consecuencia, el rendimiento del modelo mejoraría. Sin embargo, entre más filtros se utilicen, mayor será la cantidad de parámetros que el modelo necesitará aprender, aumentando el costo computacional y un posible sobre entrenamiento. Una característica importante de las CNN's es que son invariantes a las traslaciones.

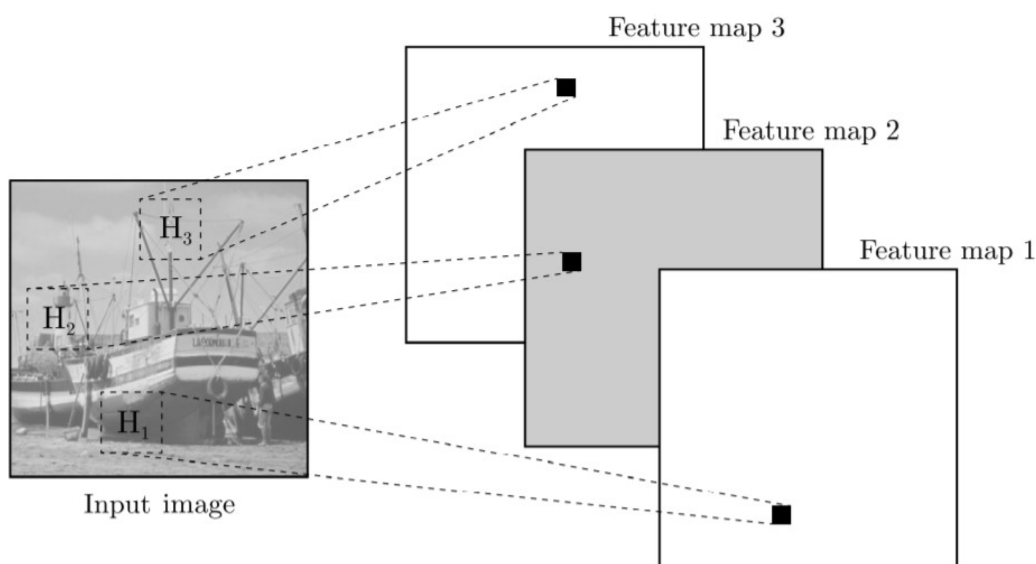

FIGURA 2.16: Ejemplo de extracción de características de un kernel de tres capas.[35]

A continuación, explicaremos términos importantes de un modelo convolucional.

- Profundidad (*Canales*): La profundidad de una capa es el número de filtros que se están aplicando en esta capa. Este término no se debe confundir con el de una NN, el cual corresponde al número de capas ocultas que este tiene.

- Campo receptivo: Cada píxel en una matriz de mapa de características de salida resulta como un promedio ponderado de los píxeles dentro de un área específica de la matriz de imágenes de entrada (o de la salida de la capa anterior). El área específica que corresponde a un píxel se conoce como su campo receptivo.

- Paso (*stride*): En la práctica, en lugar de deslizar la matriz del filtro un píxel a la vez, se puede deslizar, digamos, s píxeles. Este valor se conoce como paso. Para valores de $s > 1$, se obtienen matrices de mapas de características que tienen un tamaño más pequeño.

- Sin relleno (*Zero padding*): A veces, se utilizan ceros para rellenar la matriz de entrada alrededor de los píxeles del borde. De esta forma aumenta la dimensión de la matriz. Si la matriz original tiene dimensiones $l \times l$, después de expandirla con p columnas y filas, las nuevas dimensiones se convierten en $(l+2p) \times (l+2p)$.

- Término bias: Después de cada operación de convolución que genera un píxel de mapa de características, se agrega un término bias, b. El valor de este término también se calcula durante el entrenamiento. Tenga en cuenta que se utiliza un término de sesgo común para todos los píxeles en el mismo mapa de características. Esto está en consonancia con la lógica del reparto del peso; de la misma manera que todos los parámetros de una matriz de filtro son compartidos por todos los píxeles de la matriz de imágenes de entrada, se utiliza el mismo término bias para todas las ubicaciones de píxeles.

El tamaño del mapa de características se puede ajustar, mediante el ajuste del valor s (stride), y el número de columnas y filas extras de ceros. Esto se puede verificar, sí $I \in \mathbf{R}^{l\times l}$, $H \in \mathbf{R}^{m\times m}$, s es el stride y p es el número de columnas y filas extras (*padding*), entonces el mapa de características tiene una dimensión $k \times k$, donde:

$$k = \left\lfloor \frac{l+2p-m}{s} + 1 \right\rfloor \tag{2.37}$$

2.2.2. ReLU y Softmax

Una vez realizada la convolución y el término bias ha sido agregado en todos los valores de los mapas de características, el siguiente paso a aplicar es la no linealidad (función de activación), esto se aplica a cada uno de los píxeles de los mapas de características.

Actualmente, la función de activación más utilizada es *ReLu* (Rectified Linear Units) [45], definida de la siguiente manera:

$$f(z) = max(0, z) \tag{2.38}$$

El uso de ReLU en las capas ocultas puede acelerar significativamente el tiempo de entrenamiento. Esta función de activación no sufre de saturación y su derivada es igual a uno cuando la neurona opera en su región activa ($z > 0$) . Por lo tanto, es deseable establecer el bias de las neuronas, durante la inicialización, en un pequeño valor positivo, por ejemplo, $\theta_0 = 0{,}1$, para aumentar la probabilidad de que la entrada a la activación sea positiva. En la figura 2.17 se muestra la función ReLU.

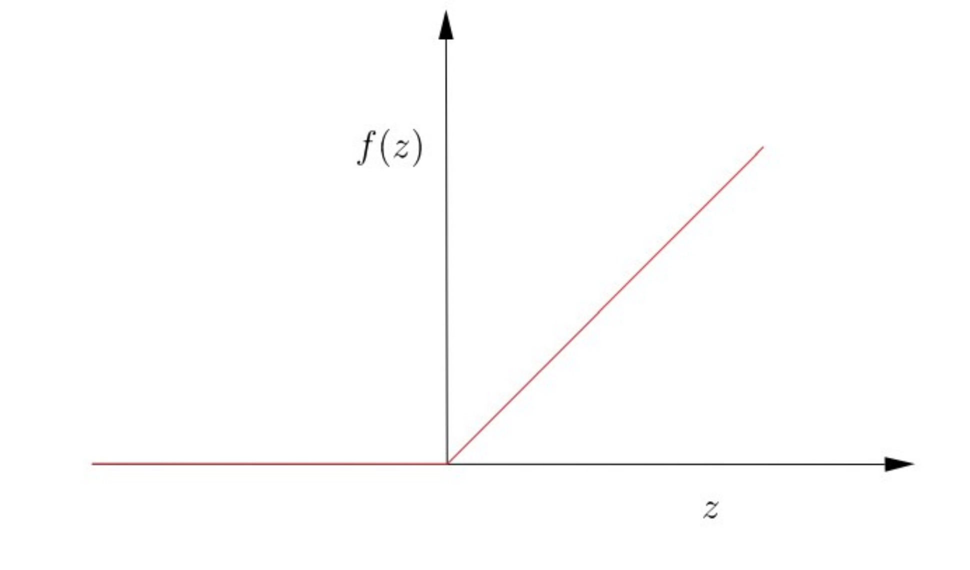

FIGURA 2.17: Función ReLU.[35]

La figura 2.18**A**, nos muestra la imagen del bote después de que el filtro de detección de bordes se utilizara, y la figura 2.18**B**, nos muestra el resultado obtenido después la aplicación de la función ReLU sobre cada uno de los pixeles.

Convencionalmente, ReLU se utiliza como función de activación (en las capas ocultas) en una DNN, con la función *Softmax* como función de clasificación (en la última capa). La función softmax se usará cada vez que queramos representar una distribución probabilística sobre una variable discreta con n valores posibles. Esta se puede ver como una generalización de la función sigmoidea, la cual era usada para representar la distribución probabilística de una variable binaria, formalmente la función es la siguiente.

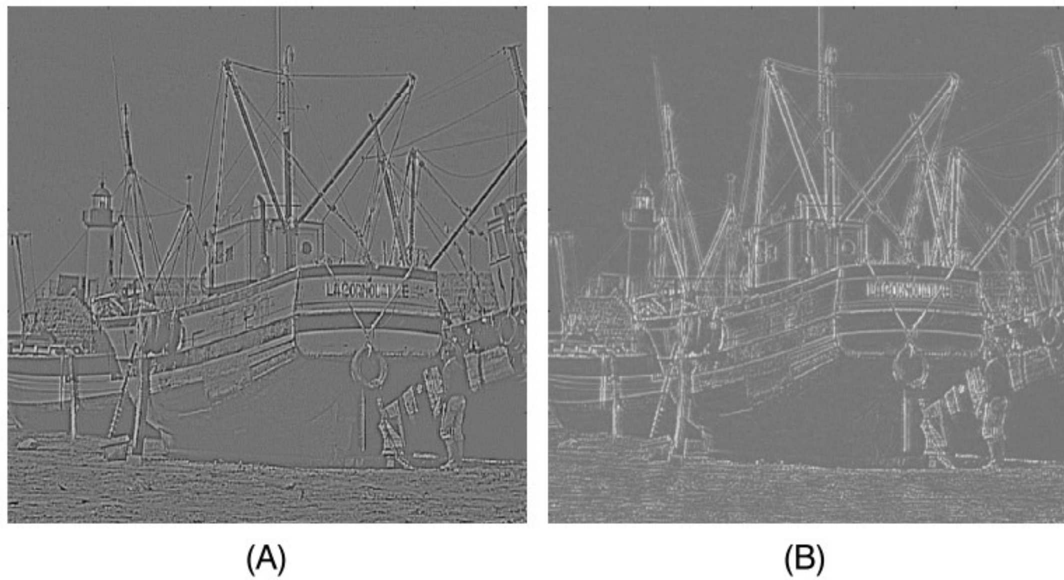

FIGURA 2.18: Función ReLU.[35]

$$softmax(Z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2.39}$$

2.2.3. Pooling

El propósito de este paso es reducir la dimensión de cada mapa de características. Para ello, se define una ventana y se desliza sobre la matriz correspondiente. El deslizamiento se puede realizar adoptando un valor para el parámetro de *stride* respectivo, s.

La operación pooling consiste en escoger un sólo valor que represente todos los píxeles que se encuentren dentro de esta *ventana*. La operación más común es *max pooling* [46]; consiste en seleccionar el elemento de mayor tamaño dentro de la ventana. En todos los casos, pooling ayuda a que la representación sea aproximadamente invariante a pequeñas traslaciones de la entrada. La invarianza a la traslación significa que si trasladamos la entrada en una pequeña cantidad, los valores de la mayoría de las salidas agrupadas no cambian. Por ejemplo, al determinar si una imagen contiene una cara, no necesitamos saber la ubicación de los ojos con una precisión perfecta de píxeles, solo necesitamos saber que hay un ojo en el lado izquierdo de la cara y otro ojo en el lado derecho. En otras palabras, si una pequeña traslación no trae a la ventana un nuevo

elemento más grande y tampoco elimina el elemento más grande sacándolo fuera de la ventana, entonces el máximo no cambia

Debido a que la operación pooling resume las respuestas en todo un vecindario, es posible utilizar menos unidades de pooling que unidades detectoras, informando estadísticas resumidas para las regiones de agrupación separadas por k píxeles en lugar de 1 píxel. Esto mejora la eficiencia computacional de la red porque la siguiente capa tiene aproximadamente k veces menos entradas para procesar. Por ejemplo, la figura 2.19**A** es la matriz de entrada a la que se le aplica un max pooling de 2×2 y un stride $s = 2$, dando como resultado la matriz de la figura 2.19**B**. Cuando el número de parámetros en la siguiente capa es una función de su tamaño de entrada (como cuando la siguiente capa está completamente conectada y se basa en la multiplicación de matrices), esta reducción en el tamaño de entrada también puede resultar en una mayor eficiencia estadística y menores requisitos de memoria para almacenar los parámetros. Otro ejemplo se puede observar en la figura 2.20, esta muestra el efecto de aplicar pooling a la imagen que se muestra a la izquierda. Sin duda, los bordes se vuelven más gruesos, pero la información relacionada con los bordes aún se puede extraer. Se debe tener en cuenta que después de la operación, el tamaño de la matriz de imágenes se reduce.

2	3	**7**	1	4	5
4	**5**	0	6	**7**	1
6	2	1	3	2	3
4	5	6	**8**	4	**5**
1	3	2	1	2	4
4	2	1	**8**	**6**	3

(A)

5	7	7
6	8	5
4	8	6

(B)

FIGURA 2.19: Ejemplo Max Pooling con $s = 2$ y una ventana de 2×2.[35]

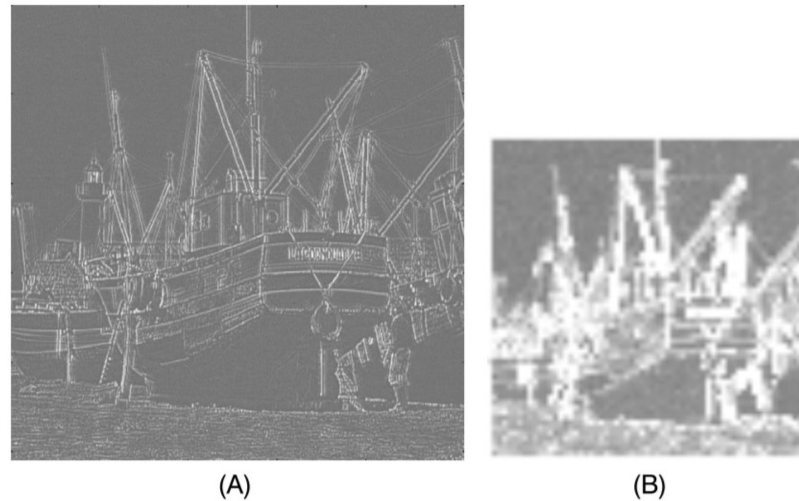

FIGURA 2.20: Ejemplo Max Pooling en una imagen con $s = 2$ y una ventana de 8×8.[35]

2.2.4. Regularización e Inicialización de una CNN

Regularización

Un problema central en machine learning es cómo hacer un algoritmo que funcione bien, no solo en los datos de entrenamiento, sino también en las nuevas entradas. Muchas estrategias utilizadas en machine learning están diseñadas explícitamente para reducir el error de prueba, posiblemente a expensas de un mayor error de entrenamiento. Estas estrategias se conocen colectivamente como regularización. Hay muchas formas de regularización disponibles para un profesional de DL. De hecho, el desarrollo de estrategias de regularización más efectivas ha sido uno de los principales esfuerzos de investigación en el campo. A continuación daremos una breve introducción a las más comunes.

- L^2 (*Weight decay*): Esta regularización se basa en el supuesto de que un modelo con pesos "pequeños" es más simple que un modelo con pesos largos. Esto mediante la penalización del cuadrado de los pesos en la función de costo, al tener pesos grandes. Esto se refiere a una regularización de función de costo típica a través de la norma euclidiana cuadrada de los pesos. En lugar de minimizar una función de costo, $J(\theta)$, se usa su versión regularizada, de modo que:

$$J'(\theta) = J(\theta) + \lambda||\theta||^2 \tag{2.40}$$

- L^1: Mientras que la regularización L^2 es la forma más común de ”achicar” los pesos, existen otras maneras de penalizar el tamaño de los parámetros. Otra opción es la regularización L^1, de forma general la función sería:

$$J'(\theta) = J(\theta) + \lambda h(\theta) \tag{2.41}$$

  Existen muchas variantes de la función $h(\theta)$, por lo general, se debe asignar un límite θ_h, sí cualquier peso $\theta_k < \theta_h$ (donde k, es el número total de pesos utilizados en el modelo), se penaliza el peso y tiende a cero. Y para valores donde $\theta_k > \theta_h$, no se penalizará el peso. De esta manera, los pesos menos significativos son ”empujados” hacia el cero.

- Parada anticipada (*early stopping*): Esta es una alternativa muy básica, que resulta útil en la práctica. La idea es detener el entrenamiento cuando el error comienza a aumentar en el conjunto test.

- *Dropout*[47] El término "Droput"se refiere a eliminar unidades/nodos (en las capas ocultas y de entrada) en una red neuronal. En cada iteración el algoritmo de entrenamiento removerá un número de nodos. Los parámetros de los nodos restantes se actualizan según la regla de actualización. En otras palabras, en cada paso de la iteración solo se actualiza un subconjunto de los parámetros, mientras que el resto (los asociados con los nodos eliminados) se congelan en sus estimaciones disponibles actualmente de la iteración anterior. El subconjunto que consta de los nodos restantes define una subred del original más grande. La eliminación de los nodos se realiza de forma probabilística. Es decir, cada nodo se retiene con probabilidad P. Por lo general, el valor de P es igual a $0{,}5$ para las capas ocultas y se establece igual a $0{,}8$ para los nodos de la capa de entrada.

- Batch Normalization[48]: Batch Normalization evita la explosión de la activación corrigiendo repetidamente todas las activaciones para que sean de media cero y desviación estándar unitaria. Con esta “precaución de seguridad”, es posible entrenar redes con grandes tasas de aprendizaje, ya que las activaciones no pueden crecer de manera incontrolable ya que sus medias y variaciones están normalizadas. Considerando Batch Normalization para una CNN. Dónde su salida y entrada de una capa Batch Normalization son tensores de cuatro dimensiones. Donde la entrada será $I_{b,c,x,y}$ y la salida $O_{b,c,x,y}$. Las dimensiones correspondientes a los

ejemplos dentro de un lote b, canal c y dos dimensiones espaciales x, y respectivamente. Para las imágenes de entrada, los canales corresponden a los canales RGB. Batch Normalization aplica la misma normalización para todas las activaciones en un canal determinado:

$$O_{b,c,x,y} = \gamma_c \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta_c, \forall b, c, x, y \quad (2.42)$$

Aquí, Batch Normalization extrae la media $\mu_c = \frac{1}{|B|} \sum_{b,x,y} I_{b,c,x,y}$ de todas las activaciones de las entradas en el canal c, donde B contiene todas las activaciones en el canal c, a través de todas las características b en todo el mini-bacth y en todas las dimensiones de x y y. Posteriormente, Batch Normalization divide la activación centrada por la desviación estándar σ_c (más ϵ para la estabilidad numérica). γ_c y β_c, son parámetros que se aprenden durante el entrenamiento.

Inicialización

El objetivo de la inicialización de los pesos es evitar que las salidas de activación de capa exploten o desaparezcan durante el curso de un pase directo a través de una DNN. Si ocurre alguna de las dos cosas, los gradientes de la función de costo serán demasiado grandes o demasiado pequeños para fluir hacia atrás de manera beneficiosa, y la red tardará más en converger, si es que es capaz de hacerlo.

Si se inicializan los pesos con ceros o con números aleatorios, pueden ocurrir algunos de estos casos: 1) Si los pesos en una red comienzan demasiado pequeños, entonces la señal se reduce a medida que pasa por cada capa hasta que es demasiado pequeña para ser útil. 2) Si los pesos en una red comienzan demasiado grandes, la señal crece a medida que pasa por cada capa hasta que es demasiado masiva para ser útil. A continuación se explicarán dos métodos de inicialización de pesos muy comunes en DL.

- Inicialización Xavier[49]: La inicialización de Xavier asegura que los pesos sean ”los correctos”, manteniendo la señal en un rango razonable de valores a través de muchas capas. La inicialización de los pesos en una red extrayéndolas de una distribución con media cero y una varianza específica, se daría de la siguiente manera:

$$Var(W) = \frac{2}{n_{in} + n_{out}} \quad (2.43)$$

Donde W es la distribución de inicialización de la neurona en cuestión, y n_{in} es el número de neuronas que la alimentan, y n_{out} es el número de neuronas de las que se alimenta el resultado.

- Inicialización He[50]: La creación de la inicialización He se da porque la inicialización Xavier provocaba la explosión de los pesos cuando se utilizaba en conjunto con la activación ReLU. Con la inicialización Xavier, se asume que las activaciones lineales aceptan entradas de promedio 0. Retomemos de la ecuación 2.38, que esto no es posible. La inicialización He se da de la siguiente manera:

$$Var(W) = \frac{2}{n_{in}} \tag{2.44}$$

Lo que tiene sentido: ReLU es cero para la mitad de su entrada, por lo que necesita duplicar el tamaño de la variación de peso para mantener constante la variación de la señal.

2.2.5. La Arquitectura de una CNN

La forma típica de una arquitectura CNN consiste de una secuencia de capas convolucionales, cada una comprimiendo los tres temas ya vistos (Sub-sección 2.2.1, 2.2.2 y 2.2.3), que son, la convolución, la no linealidad (ReLU) y pooling. Dependiendo de la aplicación, se pueden apilar tantas capas convolucionaes como sean necesarias, donde la salida de una capa se vuelve la entrada de la siguiente. La arquitectura general se muestra en la figura 2.21. En la primera capa, se emplean varios volúmenes de filtro (canales) para realizar convoluciones seguidas por la operación ReLU. Luego, la etapa de la operación pooling toma el control para reducir la altura y el ancho de cada volumen de salida, que luego se usa como entrada para la segunda capa, y así sucesivamente. Finalmente, se vectoriza el volumen de salida de la última capa. A veces, esto también se conoce como operación de *aplanamiento*. La vectorización puede tener lugar a través de varias estrategias. De hecho, el vector obtenido forma el vector de características que finalmente se ha generado a través de las diversas transformaciones que las convoluciones implementan capa tras capa. Este vector de características se utilizará luego como entrada para un clasificador, por ejemplo, para una capa full connected (parte inferior de la figura 2.21) o para cualquier otro predictor.

La estrategia general es seguir reduciendo la altura y el ancho mientras aumenta la profundidad de los volúmenes. Una mayor profundidad corresponde a más filtros por

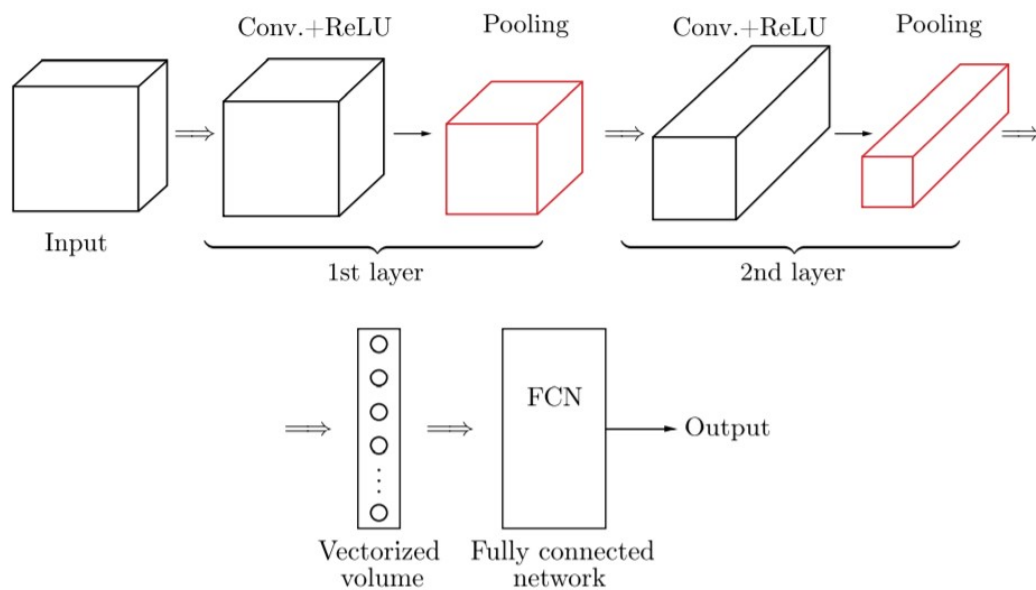

FIGURA 2.21: Arquitectura general de una CNN[35]

etapa, lo que se traduce en más características. Tanto el número de capas convolucionales como el número de capas de la full connected dependen en gran medida de la aplicación y, hasta ahora, no existe un método formal para determinar automáticamente el número de capas, así como el número de filtros o nodos por capa. La elección es una cuestión de "ingeniería" y evaluación de diferentes combinaciones para seleccionar la mejor. Una buena práctica es seleccionar una arquitectura existente que se haya utilizado antes en una aplicación relacionada y comenzar desde allí.

El entrenamiento de una red convolucional sigue un fundamento similar al de Backpropagation, que se ha discutido en la Sub-sección 2.1.3. Hay más detalles que se deben tomar en cuanta para lograr esto, para tener un mejor entendimiento leer [51].

2.2.6. Arquitectura Siamese *SNN*

En esta sub-sección se explicará la arquitectura Siamese, iniciando con dos conceptos importantes, que son el "*One-Shot Learning*" y "*Weight Sharing*", para después explicar la arquitectura.

- One-Shot Learning: Una de las desventajas que presenta DL en el estado del arte es que necesitan de grandes cantidades de datos para su buen funcionamiento. One-Shot Learning se refiere a problemas de DL en los que el modelo solo recibe

una instancia para los datos de entrenamiento y debe aprender a volver a identificar esa instancia en los datos de prueba. La idea es entrenar el modelo para diferenciar entre pares iguales y diferentes y luego generalizar estas ideas para evaluar nuevas categorías. El papel de DL en este proceso es aplicar una serie de funciones que convierten una representación de alta dimensión como matrices de imágenes en representaciones, generalmente de mucha menor dimensión, que son fácilmente separables entre sí. En la actualidad se ha aplicado para la verificación de rostros [52].

- Weight Sharing: Weight Sharing es una de las características principales de una CNN. Tenemos que el número de filtros dentro de una capa indica la dimensión de profundidad del volumen de salida de los mapas de activación/características, que son creados por la capa convolucional como entrada a la siguiente capa. Cada uno de estos filtros tiene un ancho y una altura establecidos, que corresponde al campo receptivo local de una sola unidad dentro de la capa. Los filtros que actúan sobre los datos de entrada crean la salida de una capa convolucional, el mapa de características. Los valores de peso dentro de los filtros se pueden aprender durante la fase de entrenamiento de una CNN. La dimensión de salida de la capa convolucional tiene un componente de profundidad, si dividimos cada segmento de la salida obtendremos un plano $2D$ de un mapa de características. El filtro utilizado en un solo plano $2D$ contiene un peso que se comparte entre todos los filtros utilizados en el mismo plano. La ventaja de esto es que mantenemos el mismo detector de características que se utiliza en una parte de los datos de entrada en otras secciones de los datos de entrada y además el número de parámetros es menor. Ahora, este mismo concepto se puede aplicar sobre capas completas, por ejemplo; se pueden tener dos capas convolucionales que tengan los mismos pesos, por ende, cuando estas dos capas estén en fase de entrenamiento, la actualización de los pesos será la misma para ambas.

Como se vio en la Sub-sección 2.2.5, la arquitectura básica de una CNN, consiste de una entrada (una imagen), que pasará por n capas convolucionales, y alfinal se vecotrizará para que un clasificador realice su tarea (ver la figura 2.21). Este podría ser la función Softmax, que nos brindará un vector de probabilidades para cada clase, la clase con mayor probabilidad es la ganadora. También se mencionó que el vector de características se puede utilizar para que otros métodos (no una NN) se pudieran aplicar. Esto será importante para una SNN.

La arquitectura consta de dos entradas, las dos entradas recibirían una imagen diferente. Dado que esta es una arquitectura para asistir la tarea de *verificación*, la imagen x_j tendría que pertenecer a la clase verdadera, mientras la imagen x_i es la que se va a verificar si pertenece o no a la clase de la imagen x_j, como se puede observar en la figura 2.22.

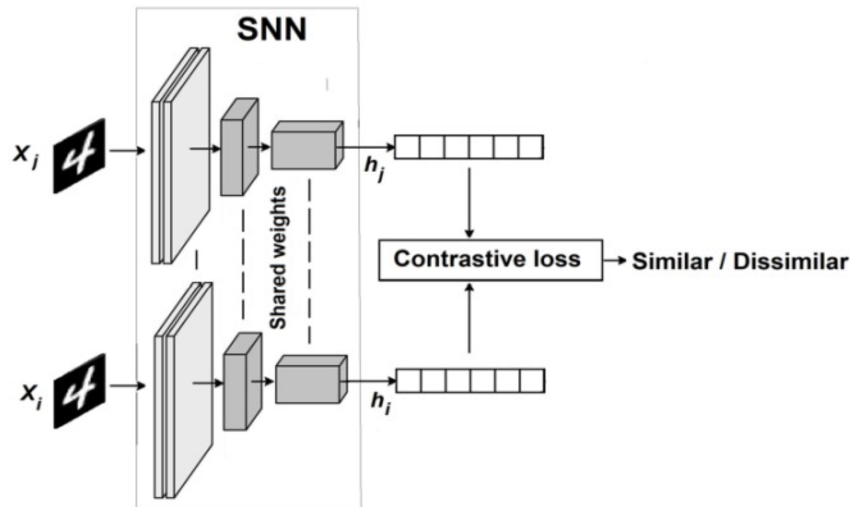

FIGURA 2.22: Arquitectura general de una SNN[53]

Como se puede observar en la figura 2.22, los pesos de ambas capas son compartidas (weight sharing). Lo cual tiene sentido, se requiere que ambas capas sean la "misma" para poder comparar los dos vectores de características. Las CNN que usan dos capas idénticas para dos entradas diferentes se llaman Siamese NN. Como la última capa no consta de un clasificador como lo es Softmax, se utilizará otro método. Entonces, el vector resultante de la última capa, será nuestra codificación de la imagen $f(x_{(i)})$. Se tienen que aprender parámetros para que:

- Si $x_{(i)}$ y $x_{(j)}$, pertenecen a la misma clase, entonces $||f(x_{(i)}) - f(x_{(j)})||^2$ es pequeño.
- Si $x_{(i)}$ y $x_{(j)}$, son de diferentes clases, entonces $||f(x_{(i)}) - f(x_{(j)})||^2$ es grande.

La primera vez que se utilizó este tipo de arquitectura fue en 1993 por Bromley et al. [54] y fue para verificar firmas, donde en vez de utilizar una CNN se utilizó una NN, y al final se obtenía la distancia del vector de características de ambas firmas. Una métrica que se puede utilizar es la norma Gaussiana L^2.

Capítulo 3

Metodología

Un proyecto de ML se puede desarrollar de diferentes formas, todo va a depender de los datos con los que se cuenta y del algoritmo seleccionado para modelar estos datos, por lo general a esto se le llama *end-to-end pipeline*, y sus variantes son:

- Tradicional: Este tipo de desarrollo es útil cuando la cantidad de datos es mínima. Por lo general se llama *pipeline*, consiste de diversos pasos que se aplican sobre los datos antes del entrenamiento del modelo. Estos pasos pueden ser:
    - Recolección de los datos: Este es un problema que no se presenta muy a menudo en la actualidad, aunque existen ocasiones en las que los datos para resolver un problema en específico no existen o son muy escasos. Para estos casos es necesaria la recolección, no existe una métrica que especifique la cantidad de datos necesaria, esto también va a depender del modelo a aplicar.
    - Visualización y preparación de los datos: Consiste en revisar el estado de los datos, que sean útiles (en su mayoría).Se pueden aplicar ciertos filtros para eliminar datos innecesarios que pueden causar ruido en el entrenamiento, como pueden ser los *outliers*. También se pueden aplicar algunos métodos de inferencia para poder revisar que los datos sean aceptables y el modelo pueda generalizarlos.
    - Pre-procesamiento de los datos: Este paso dependerá del tipo de datos con el que se esté trabajando. Un ejemplo puede ser la normalización de los datos para que se trabajen con datos entre 0 y 1, o si se tratan de imágenes, pasarlas a blanco y negro, para que se les puedan aplicar otros métodos de extracción de características.

  - Extracción de características: Una vez pre-procesados los datos, se les puede aplicar algún algoritmo de extracción de características, que al final resultaría en un vector de características que se va a utilizar para entrenar el modelo.
  - Selección del modelo y entrenamiento: Se selecciona el modelo dependiendo de los datos que se tenga y se entrena con los vectores de características obtenidos, también se selecciona la métrica y el objetivo a alcanzar.
  - Afinamiento del modelo: Dependiendo de la métrica seleccionada para evaluar el modelo, se va a ajustar o modificar el modelo, esto hasta que se alcance la meta deseada.

- End-to-end DL: Esta es la simplificación de un proceso o sistema de aprendizaje mediante la aplicación de DL. A diferencia del método tradicional, aquí se requiere de una gran cantidad significativa de datos para que el modelo funcione, además, ya no será necesaria la extracción de características, esto se hará automáticamente. Tal vez se requiera aplicar algún pre-procesamiento antes del entrenamiento, y la selección de una métrica siempre será necesaria para cualquier algoritmo de ML.

Como se explico en el Capítulo 2, Sección 2.1, DL es un algoritmo de aprendizaje automático que consta de múltiples capas, lo que permite la abstracción de las representaciones de múltiples datos. Su propósito principal es extrapolar nuevas características a partir de representaciones de entrada sin procesar, esto se hace sin decirle al modelo que características utilizar y cómo extraer las características. Siendo esta la importancia de utilizar las CNN's, por su capacidad de clasificar y extraer las características de las imágenes sin necesidad de decirle cómo hacerlo. En este capítulo describiremos la metodología utilizada (end-to-end DL) y el modelo híbrido convolocional que se utilizó.

3.1. Conjunto de Datos

Para esta investigación se recolectaron imágenes de seis diferentes variedades de Durazno. Esta recolección se efectuó en el ColPos campus Montecillo, con el apoyo del Doctor Guillermo Calderón Zavala, encargado de la investigación del Durazno, en el departamento de Fruticultura.

3.1.1. Recolección

La recolección de las hojas consistió de los siguientes pasos:

- Se recortaron ramas que tuvieran hojas en buen estado (sin ningún signo de enfermedad y enteras), se almacenaron en bolsas de papel (Divididas por clases figura 3.1**A**) para después guardarlas en una hielera de unicel (figura 3.1**B**), para su transportación.
- Se recolectaron las hojas de cada rama y se apilaron una sobre otra para que no perdieran la forma (figura 3.1**C** y **D**), una vez recolectadas se refrigeraron para que no perdieran su forma y color (figura 3.1**E**). Cada muestra puede mantener su estado natural en refrigeración, no más de 4 días.
- Se le tomó una foto a cada hoja en un *set* con luz artificial (para evitar sombras) y fondo blanco (figura 3.1**F**). Esto se realizó con una cámara *Canon EOS Rebel T3*, la dimensión de las imágenes fue de 720×480. En esta investigación, un elemento del conjunto de datos consiste de las dos caras de las hojas, ver figura 3.2.

3.1.2. Pre-procesamiento y Conjunto Final

El único pre-procesamiento que se aplicó sobre las imágenes fue la escala de estas mismas para poder utilizarlas en los modelos propuestos. El primer paso fue hacer las imágenes cuadradas, para ello se agregaron valores de color blanco en las orillas deseadas, obteniendo imágenes de 720×720. Esto se realizó usando Python, en el entorno Spyder. Finalizando esto, se crearon dos conjuntos de datos, uno consiste de imágenes de tamaño 224×224 y el otro conjunto de imágenes de 416×416.

Es importante recalcar que cada par de hojas esta apuntando a la misma dirección, como se puede observar en la figura 3.2, esto se hace por conveniencia ya que nuestro modelo trabaja con capas con pesos compartidos. La distribución final del conjunto de entrenamiento y test se encuentra en el cuadro 3.1. Se tomó el $10\,\%$ de cada clase para el conjunto de test, el restante para entrenamiento.

3.2. Modelos Propuestos

Los modelos se basaron en [14] y [15]. La arquitectura de [15] se explicó en la Subsección 2.2.6, que es el modelo Siamese. El modelo *Early Fusion* [14], consta de dos

Clase y Nombre	Número de Pares	Conjunto Entrenamiento	Conjunto de Test
1. CP-03-06	101	90	11
2. Oro Azteca	207	186	21
3. Oro San Juan	204	179	25
4. Cardenal	229	209	20
5. Colegio	205	190	15
6. Robin	319	285	34
Total	1265	1139	126

CUADRO 3.1: Distribución final del conjunto de datos.

entradas. La primer entrada recibirá una cara de la hoja, y la siguiente entrada recibe una sección recortada de nuestra primer entrada, esta sección será la parte central de la hoja, ver figura 3.3. Este modelo se llama así porque realiza una sumatoria ”temprana” en la última capa convolucional, para después pasar por una capa full connected.

Los modelos propuestos utilizaron el principio del modelo Siamese de pesos compartidos para que este entrene con las dos caras de las hojas, así un solo flujo de capas convolucionales aprenderán de ambas caras. Y una vez se tengan los dos tensores (cara frontal y trasera), se sumarán, para crear un nuevo tensor de características de las dos caras de la hoja. En la figura 3.4 se muestra un esquema de los modelos, donde una unidad (*Unit#*) consta de una capa convolucional, Batch Normalization, una activación ReLU y Max-Pooling. Más detalles sobre los modelos se describen en los cuadros 3.2 y 3.3. Se utilizó dropout en las unidades 4 y 5 del modelo 224×224, para la unidad 4 se utilizó un dropout de $0{,}05$ y para la unidad 5, un dropout de $0{,}1$. Para el modelo 416×416, se utilizaron los mismos valores de dropout, pero se aplicaron en las últimas dos unidades, 6 y 7. Para ambos modelos, los pesos de las capas convolucionales se inicializaron con He, esto se debe a que se utilizó la activación ReLU, de igual manera, para todas las capas convolucionales. Para la capa full connected, los pesos se inicializaron con la inicialización Xavier, esto porque se utilizó la función de activación Softmax, esto aplica para ambos modelos. En el Apéndice A se muestra el código utilizado para ambos modelos, y en el Apéndice B se muestra gráficamente los modelos con más detalles.

Unit1 Conv	MaxPool	Unit2 Conv	MaxPool	Unit3 Conv	MaxPool	Unit4 Conv	MaxPool	Unit5 Conv	MaxPool	FC
$5 \times 5 \times 18$	3×3	$5 \times 5 \times 30$	3×3	$5 \times 5 \times 60$	3×3	$5 \times 5 \times 100$	2×2	$5 \times 5 \times 90$	2×2	720

CUADRO 3.2: Modelo 224×224.

Unit1 Conv	MaxPool	Unit2 Conv	MaxPool	Unit3 Conv	MaxPool	Unit4 Conv	MaxPool	Unit5 Conv	MaxPool	Unit6 Conv	MaxPool	Unit7 Conv	MaxPool	FC
$5\times5\times18$	3×3	$5\times5\times30$	3×3	$5\times5\times60$	3×3	$5\times5\times100$	2×2	$5\times5\times90$	2×2	$5\times5\times90$	2×2	$5\times5\times180$	2×2	720

CUADRO 3.3: Modelo 416×416.

FIGURA 3.1: Recolección de datos.

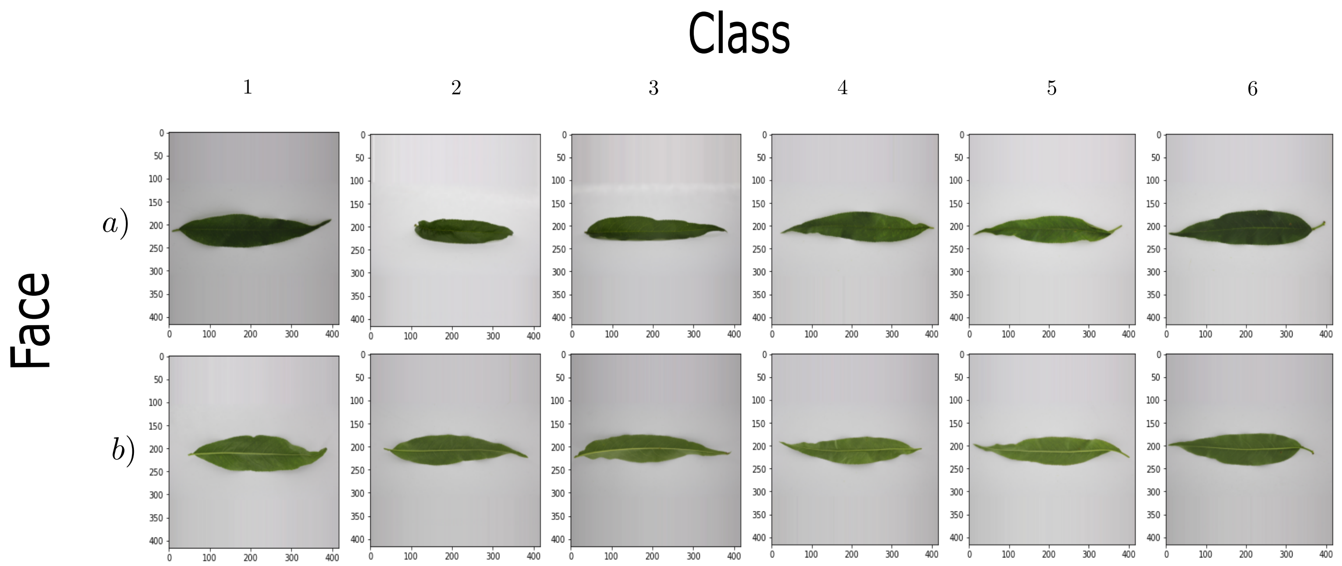

FIGURA 3.2: Muestra de las 6 clases de Durazno.

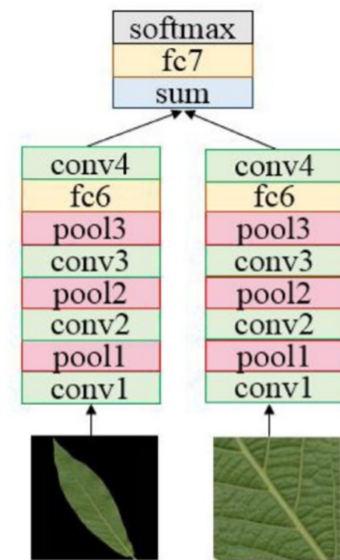

FIGURA 3.3: Modelo Early Fusion[14]

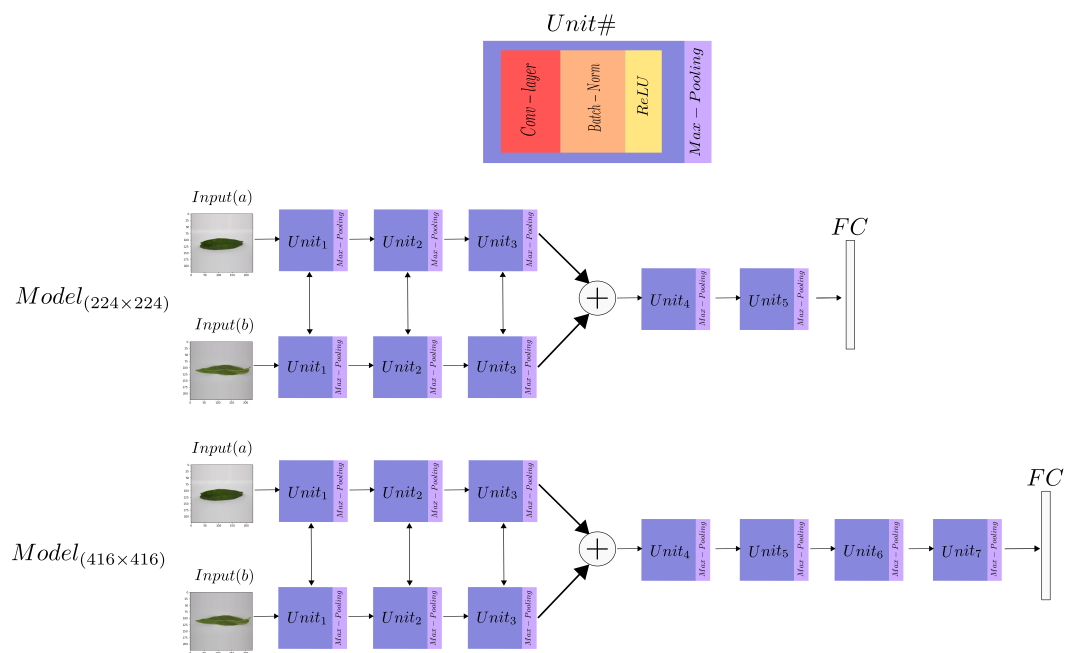

FIGURA 3.4: Modelos Propuestos

Capítulo 4

Resultados y Conclusiones

En este capítulo se muestra el producto obtenido de esta investigación, que consta de un capítulo de libro aceptado en el *Internationl Conference on Intelligent Computing* 2020. Y se expuso de manera virtual la investigación en dicha conferencia el sábado 10 de Octubre del 2020.

Cabe destacar que se creó un repositorio con las imágenes utilizadas, que son las seis variedades de durazno. Está disponible en Google Drive. La liga también está disponible en el resumen de la Sección 4.1.

En el Apéndice C, se muestra la gráfica AUC-ROC de ambos modelos, así como sus respectivas matrices de confusión.

4.1. Certificado de Aceptación y Capítulo de Libro

2020 International Conference on Intelligent Computing, October 2-5, 2020, Bari, Italy
http://ic-ic.tongji.edu.cn/2020/index.htm

Attendance Certificate

August 15, 2020

Paper ID: **164**
Authors: **Daniel Ayala Niño, Jair Cervantes Canales, Farid García Lamont, Guillermo Calderón Zavala and Joel Ayala de la Vega**
Paper Title: **A Hybrid Convolutional Neural Network for Complex Leaves Identification**

Dear Daniel Ayala Niño, Jair Cervantes Canales, Farid García Lamont, Guillermo Calderón Zavala and Joel Ayala de la Vega ,

This is to certify that your above mentioned paper has been officially accepted by our 2020 International Conference on Intelligent computing (ICIC 2020), and you have registered this paper. This year due to COVID-19 outbreak, our ICIC 2020 will be organized as virtual meeting by a virtual meeting platform.

Thank you for your attention.

Sincerely Yours

ICIC 2020 Orgnization Committee

A Hybrid Convolutional Neural Network for Complex Leaves Identification

Daniel Ayala Niño[1(✉)], Jair Cervantes Canales[1], Farid García Lamont[1], Joel Ayala de la Vega[1], and Guillermo Calderón Zavala[2]

[1] UAEMEX (Universidad Autónoma del Estado de México), Jardín Zumpango s/n, Fraccionamiento El Tejocote, 56259 Texcoco, CP, Mexico
{danielayalanio, joelayala2001}@yahoo.com.mx, {jcervantesc, fgarcial}@uaemex.mx

[2] ColPos (Colegio de Postgraduados), Km. 36.5, Montecillo, 56230 Texcoco, CP, Mexico
cazagu@colpos.mx

Abstract. The classification of leaves has gained popularity through the years, and a great variety of algorithms has been created to target these tasks, among those is the Deep Learning approach, which simplicity of learning from raw imputed data makes this task easy to target. However, not all methods are into the complex leaves classification task. In this work we propose a different approach in the way the leaf's pictures are used to train the models, this is done by using the front and back face of a leaf as one element of the dataset. These pairs will be inputted into two shared convolutional layers, making the models to learn from a complete leaf. The results obtained in this work overpassed the accuracy obtained in related works. For this, we created a new complex leaves dataset, that consists of 6 different kinds of peach varieties, the dataset is available in this link (https://drive.google.com/drive/folders/1rWCr9DrknoK0HKFhNRavCVgZ5UKjU3hi).

Keywords: Complex leaves identification · Convolutional Neural Networks · Computer vision

1 Introduction

Plants identification has always played an important role in different disciplines, such as medicine in the development of new drugs [1, 2, 15], botany [3], agriculture [4, 9, 32]. This last one is important for food production. It is known that by 2050, global food production will increase by 60%. Thus, knowing which variety of plant is better under certain conditions, will be a clue point to improve and increase the agricultural production [5].

Since the necessity of identifying plants has increased for different reasons exposed above, the techniques for classifying them have also improved. Botanists use different leaf's characteristics (features like texture, color, morphological, venation, among others) to identify the plants, they spend too much time studying a family of plant and its varieties, this is because most of these features are hand-crafted [3].

© Springer Nature Switzerland AG 2020
D.-S. Huang et al. (Eds.): ICIC 2020, LNCS 12463, pp. 1–11, 2020.
https://doi.org/10.1007/978-3-030-60799-9_25

4.2. Resultados

Las métricas utilizadas para evaluar el modelo fueron: accuracy (precisión), AUC-ROC, F-measure, recall (FP-rate) y precision (TP-rate). Se dio una explicación de estas métricas en el Capítulo 2 Sección 2.1.5. Dado que el número de elementos de cada clase no es homogénea, cada métrica se obtuvo por medio de un promedio ponderado ("*weighted average*"), esto es, primero cada métrica se obtiene por cada clase, esto significa que fue una clase contra todas, después de esto, el promedio ponderado era obtenido como se muestra en la ecuación 4.1.

$$weighted - avg = \frac{\sum_{n=1}^{C} M_n \times C_n}{\sum_{n=1}^{C} C_n} \tag{4.1}$$

Donde M_n es la métrica obtenida en la clase n, y C_n es el número de elementos en la clase n. Los resultados obtenidos fueron comparados con los obtenidos en [7] y [8], se muestran en el cuadro 4.1.

CUADRO 4.1: Comparación de los resultados experimentales de los modelos propuestos con otros artículos.

		Accuracy	AUC-ROC	F-measure	TP rate	FP rate
Modelo Propuesto	Modelo 224×224	92,0635 %	0,99	0,920	0,920	0,978
	Modelo 416×416	92,8571 %	0,992	0,927	0,928	0,983
[8]	Naive Bayes	76,165 %	0,992	0,761	0,762	0,005
	SVM	83,937 %	0,961	0,839	0,839	0,032
	Logistic R.	73,057 %	0,942	0,728	0,731	0,055
	Decision Tree	72,020 %	0,852	0,718	0,720	0,058
[7]	Bayes	75,64 %	-	-	-	-
	BP	76,02 %	-	-	-	-
	RL	73,057 %	-	-	-	-
	SVM	83,97 %	-	-	-	-

En [8], se crearon dos CNN's básicas para descubrir cómo manejan la tarea de la clasificación de hojas complejas. Los modelos no generalizaban bien los datos y se mencionó que era necesario crear modelos más sofisticados para abordar esta tarea. En el cuadro 4.2, se muestra la comparación de nuestros modelos propuestos con los modelos propuestos en [8].

CUADRO 4.2: Comparación de los CNN's propuestos en [8] con los modelos propuestos en esta investigación.

		Accuracy
Modelo Propuesto	Modelo 224×224	92,0635 %
	Modelo 416×416	92,8571 %
[8]	Modelo Convolucional 1	62 %
	Modelo Convolucional 2	55 %

El tamaño del conjunto de datos en [8] constaba de 468 elementos, con un tamaño de 416×416. Este conjunto de datos fue menor al usado en esta investigación, en la que se usaron 1265 pares. En [7], el conjunto de datos era más pequeño, consistiendo de 193 elementos. Siendo esta una de las razones por las que los resultados fueron pobres, la resolución de las imágenes no fue especificada.

Los modelos híbridos obtuvieron una precisión del 92,0635 % en el Modelo 224×224, y una precisión del 92,8571 % para el Modelo 416×416, sobrepasando las SVM de [7] y [8], donde se obtuvo una precisión del 83,97 % y 83,937 %. Con los modelos convolucionales propuestos en [8], se ve una notable diferencia, ya que su precisión más alta fue de 62 %. El desempeño de los modelos híbridos fue casi similar, siendo el modelo con mayor resolución la que obtuvo mayor precisión, con una diferencia de 0,7936. El uso de imágenes con mayor resolución no brindó al Modelo 416×416 un mejor desempeño comparado con el modelo con menor resolución (Modelo 224×224).

4.3. Conclusiones

Cada especie de planta tiene diferentes tipos de variedades, y estas variedades se comportan de manera diferente en ciertos entornos, esto se debe al continuo desarrollo de mejores variedades. Por eso es importante identificar las diferentes variedades de plantas que tiene una especie; este tipo de hojas se llaman hojas complejas. Como se ve en la figura 3.2, las hojas a simple vista tienen el mismo color, textura y morfología, esto hace que este tipo de hojas sea difícil de clasificar. Para resolver esta tarea, se propusieron dos modelos para clasificar 6 tipos diferentes de hojas de durazno y se creó un nuevo conjunto de datos utilizando hojas del Departamento de Frutas del Colegio de Postgraduados de Texcoco. Estas fotografías fueron tomadas en un ambiente controlado con un fondo blanco. Para entrenar los modelos propuestos, el conjunto de datos se dividió en dos partes, la cara frontal de la hoja y su cara posterior, este par de

muestras representan un elemento del conjunto de datos, para cada modelo se utilizaron dos conjuntos de datos, un modelo entrenado con imágenes de 224×224 y el otro con imágenes de 416×416. Los modelos propuestos superaron significativamente los modelos en [7] y [8]. Los resultados demostraron que es importante considerar ambos lados de una hoja como uno. Los modelos Hybrid CNN obtuvieron más conocimientos sobre cómo clasificar una planta a partir de una hoja completa, en lugar de aprender de un solo lado.

Hoy en día hay conjuntos de datos disponibles con miles y millones de imágenes disponibles para entrenar un modelo, pero desafortunadamente, este no es siempre el caso. A veces, los datos no siempre están disponibles, y es de conocimiento común que cuantos más datos tenga el modelo para entrenar, mejor rendimiento tendrá. Pero no existe una regla universal que indique qué cantidad de datos es suficiente para que el modelo no sufra de overfitting. En este trabajo no se equilibraron todas las clases, en el cuadro 3.1 el número de elementos de la clase 1 con los que el modelo entrenó representa menos del 10 % de todo el conjunto de entrenamiento, y a pesar de este inconveniente, obtuvimos una alta precisión. Por otro lado, las Redes Neuronales Bayesianas han demostrado que pueden manejar pequeños conjuntos de datos y evitar el overfitting [55] y [56]. Siendo este nuestro siguiente paso, aplicando una CNN bayesiana en combinación con nuestro método propuesto, para aumentar la tasa de precisión y ver cómo este nuevo enfoque puede manejar la clasificación de hojas complejas.

Apéndice A

Código Modelos

A.1. Modelo *224X224*

```
def ModelHeavy5HYBR_Early(input_shape=(416,416,3),classes=24):
    S1 = layers.Conv2D(filters=18,kernel_size=(10,10),
        strides=(1,1),padding='valid',name='conv0_0',
        kernel_initializer = initializers.he_uniform(seed=0))
    S2 = layers.Conv2D(filters=30,kernel_size=(5,5),
        strides=(1,1),padding='valid',name='conv1_0',
        kernel_initializer = initializers.he_uniform(seed=0))
    S3 = layers.Conv2D(filters=60,kernel_size=(5,5),
        strides=(1,1),padding='valid',name='conv2_0',
        kernel_initializer = initializers.he_uniform(seed=0))
    X1Face = keras.Input(input_shape,name='Face')
    X1 = S1(X1Face)
    X1 = layers.BatchNormalization(axis=3,name='bn0_0')(X1)
    X1 = layers.Activation('relu')(X1)
    X1 = layers.MaxPooling2D((3,3),strides=(2,2))(X1)
    X1 = S2(X1)
    X1 = layers.BatchNormalization(axis=3,name='bn1_0')(X1)
    X1 = layers.Activation('relu')(X1)
    X1 = layers.MaxPooling2D((3,3),strides=(2,2))(X1)
    X1 = S3(X1)
    X1 = layers.BatchNormalization(axis=3,name='bn2_0')(X1)
    X1 = layers.Activation('relu')(X1)
    #X1 = Dropout(rate=0.05, name='drop1')(X1)
    X1 = layers.MaxPooling2D((3,3),strides=(2,2))(X1)
```

```
##########
X2Back = keras.Input(input_shape,name='Back')
X2 = S1(X2Back)
X2 = layers.BatchNormalization(axis=3,name='bn0_1')(X2)
X2 = layers.Activation('relu')(X2)
X2 = layers.MaxPooling2D((3,3),strides=(2,2))(X2)
X2 =  S2(X2)
X2 = layers.BatchNormalization(axis=3,name='bn1_1')(X2)
X2 = layers.Activation('relu')(X2)
X2 = layers.MaxPooling2D((3,3),strides=(2,2))(X2)
X2 = S3(X2)
X2 = layers.BatchNormalization(axis=3,name='bn2_1')(X2)
X2 = layers.Activation('relu')(X2)
#X2 = Dropout(rate=0.05, name='drop2')(X2)
X2 = layers.MaxPooling2D((3,3),strides=(2,2))(X2)
##################
X = layers.Add()([X1,X2])
#################
X = layers.Conv2D(filters=100,kernel_size=(5,5),
    strides=(1,1),padding='valid',name='conv3',
    kernel_initializer = initializers.he_uniform(seed=0))(X)
X = layers.BatchNormalization(axis=3,name='bn3')(X)
X = layers.Activation('relu')(X)
X = layers.Dropout(rate=0.05, name='drop5')(X)
X = layers.MaxPooling2D((2,2),strides=(2,2))(X)
X = layers.Conv2D(filters=90,kernel_size=(5,5),
    strides=(1,1),padding='valid',name='conv4',
    kernel_initializer = initializers.he_uniform(seed=0))(X)
X = layers.BatchNormalization(axis=3,name='bn4')(X)
X = layers.Activation('relu')(X)
X = layers.Dropout(rate=0.1, name='drop6')(X)
X = layers.MaxPooling2D((2,2),strides=(2,2))(X)
X = layers.Flatten()(X)
X = layers.Dense(classes,activation='softmax',
    name='fc1',
```

```
        kernel_initializer = initializers.glorot_uniform(seed=0))(X)
    model = keras.Model(inputs = [X1Face,X2Back],
            outputs = X, name='Base')
    return model
```

A.2. Modelo *416X416*

```
def ModelHeavy5HYBR_Early(input_shape=(416,416,3),classes=24):
    S1 = layers.Conv2D(filters=18,kernel_size=(5,5),strides=(1,1),
        padding='valid',name='conv0_0',
        kernel_initializer = initializers.he_uniform(seed=0))
    S2 = layers.Conv2D(filters=30,kernel_size=(5,5),strides=(1,1),
         padding='valid',name='conv1_0',
         kernel_initializer = initializers.he_uniform(seed=0))
    S3 = layers.Conv2D(filters=60,kernel_size=(5,5),strides=(1,1),
          padding='valid',name='conv2_0',
          kernel_initializer = initializers.he_uniform(seed=0))
    X1Face = keras.Input(input_shape,name='Face')
    X1 = S1(X1Face)
    X1 = layers.BatchNormalization(axis=3,name='bn0_0')(X1)
    X1 = layers.Activation('relu')(X1)
    #X1 = Dropout(rate=0.5, name='drop0_0')(X1)
    X1 = layers.MaxPooling2D((3,3),strides=(1,1))(X1)
    X1 = S2(X1)
    X1 = layers.BatchNormalization(axis=3,name='bn1_0')(X1)
    X1 = layers.Activation('relu')(X1)
    #X1 = Dropout(rate=0.5, name='drop1_0')(X1)
    X1 = layers.MaxPooling2D((3,3),strides=(2,2))(X1)
    X1 = S3(X1)
    X1 = layers.BatchNormalization(axis=3,name='bn2_0')(X1)
    X1 = layers.Activation('relu')(X1)
    #X1 = Dropout(rate=0.5, name='drop2_0')(X1)
    X1 = layers.MaxPooling2D((3,3),strides=(2,2))(X1)
    ##########
    X2Back = keras.Input(input_shape,name='Back')
```

```
X2 = S1(X2Back)
X2 = layers.BatchNormalization(axis=3,name='bn0_1')(X2)
X2 = layers.Activation('relu')(X2)
#X2 = Dropout(rate=0.5, name='drop0_1')(X2)
X2 = layers.MaxPooling2D((3,3),strides=(1,1))(X2)
X2 = S2(X2)
X2 = layers.BatchNormalization(axis=3,name='bn1_1')(X2)
X2 = layers.Activation('relu')(X2)
#X2 = Dropout(rate=0.5, name='drop1_1')(X2)
X2 = layers.MaxPooling2D((3,3),strides=(2,2))(X2)
X2 = S3(X2)
X2 = layers.BatchNormalization(axis=3,name='bn2_1')(X2)
X2 = layers.Activation('relu')(X2)
#X2 = Dropout(rate=0.5, name='drop2_1')(X2)
X2 = layers.MaxPooling2D((3,3),strides=(2,2))(X2)
##################
X = layers.Concatenate()([X1,X2])
#################
X = layers.Conv2D(filters=120,kernel_size=(5,5),strides=(1,1),
    padding='valid',name='conv3',
    kernel_initializer = initializers.he_uniform(seed=0))(X)
X = layers.BatchNormalization(axis=3,name='bn3')(X)
X = layers.Activation('relu')(X)
#X = Dropout(rate=0.25, name='drop3')(X)
X = layers.MaxPooling2D((2,2),strides=(2,2))(X)
X = layers.Conv2D(filters=140,kernel_size=(5,5),strides=(1,1),
    padding='valid',name='conv4',
    kernel_initializer = initializers.he_uniform(seed=0))(X)
X = layers.BatchNormalization(axis=3,name='bn4')(X)
X = layers.Activation('relu')(X)
#X = Dropout(rate=0.25, name='drop4')(X)
X = layers.MaxPooling2D((2,2),strides=(2,2))(X)
X = layers.Conv2D(filters=150,kernel_size=(5,5),strides=(1,1),
    padding='valid',name='conv5',
    kernel_initializer = initializers.he_uniform(seed=0))(X)
```

```
X = layers.BatchNormalization(axis=3,name='bn5')(X)
X = layers.Activation('relu')(X)
X = layers.Dropout(rate=0.05, name='drop5')(X)
X = layers.MaxPooling2D((2,2),strides=(2,2))(X)
X = layers.Conv2D(filters=180,kernel_size=(5,5),strides=(1,1),
    padding='valid',name='conv6',
    kernel_initializer = initializers.he_uniform(seed=0))(X)
X = layers.BatchNormalization(axis=3,name='bn6')(X)
X = layers.Activation('relu')(X)
X = layers.Dropout(rate=0.1, name='drop6')(X)
X = layers.MaxPooling2D((2,2),strides=(2,2))(X)
X = layers.Flatten()(X)
X = layers.Dense(classes,activation='softmax',name='fc1',
    kernel_initializer = initializers.glorot_uniform(seed=0))(X)
model = keras.Model(inputs = [X1Face,X2Back],
        outputs = X, name='Base')
return model
```

Apéndice B

Arquitecturas

B.1. Modelo *224X224*

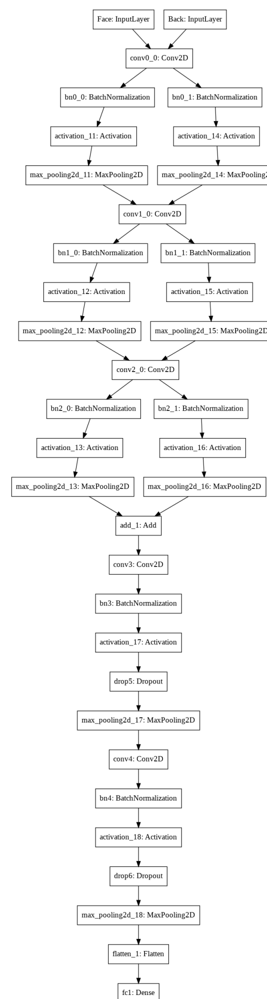

FIGURA B.1: Modelo 224×224

B.2. Modelo *416X416*

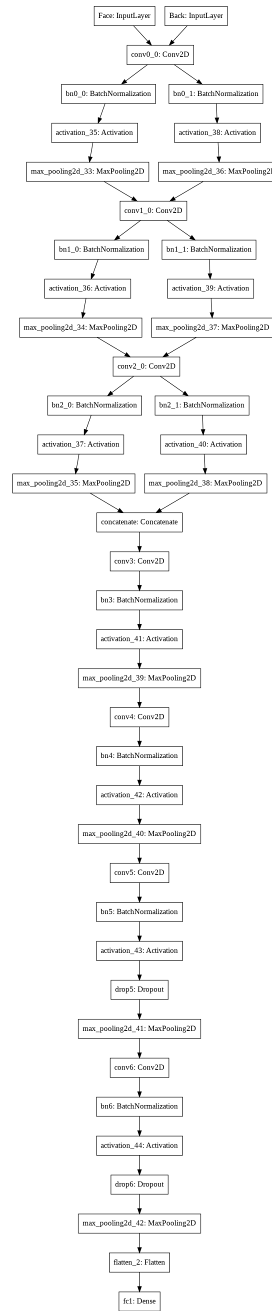

FIGURA B.2: Modelo 416×416

Apéndice C

Gráficas AUC-ROC y Matrices de Confusión

C.1. Modelo *224X224*

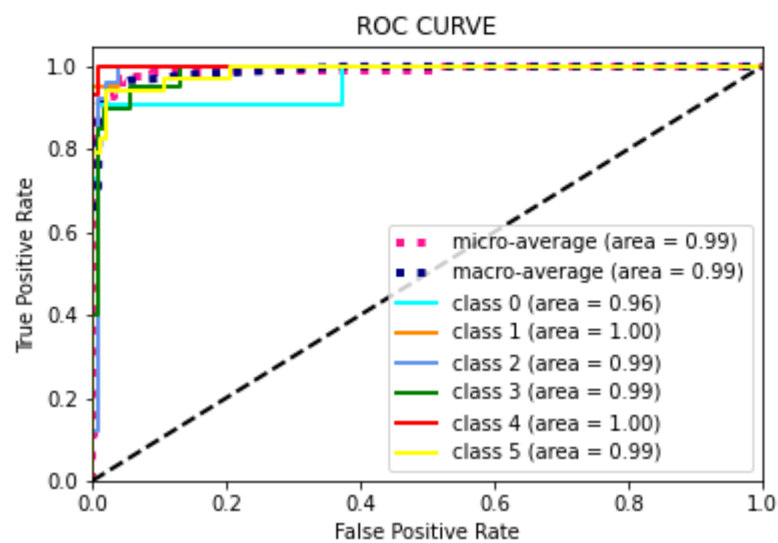

FIGURA C.1: AUC-ROC Modelo 224 × 224

C.2. Modelo *416X416*

Clase	1.	2.	3.	4.	5.	6.
1.	9	0	1	0	0	1
2.	0	20	0	0	0	1
3.	1	0	24	0	0	0
4.	0	0	1	18	0	1
5.	0	0	0	1	13	1
6.	0	1	0	1	0	32

CUADRO C.1: Matriz de Confusión Modelo 224×224.

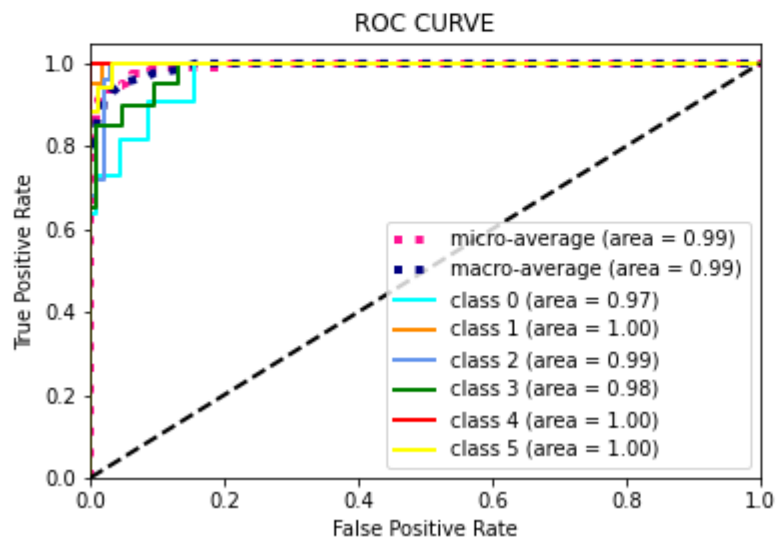

FIGURA C.2: AUC-ROC Modelo 416×416

Clase	1.	2.	3.	4.	5.	6.
1.	8	0	1	2	0	0
2.	0	21	0	0	0	0
3.	1	0	24	0	0	0
4.	0	0	1	17	0	2
5.	0	0	0	0	15	0
6.	0	2	0	0	0	32

CUADRO C.2: Matriz de Confusión Modelo 416×416.

Bibliografía

[1] R. H. L. Ip, L.-M. Ang, K. P. Seng, J. C. Broster y J. E. Pratley, «Big data and machine learning for crop protection», *Computers and Electronics in Agriculture*, vol. 151, págs. 376-383, jul. de 2018. dirección: https://www.sciencedirect.com/science/article/abs/pii/S0168169917314588.

[2] Z. Zhang y E. Sejdić, «Radiological images and machine learning: Trends, perspectives, and prospects», *Computers in Biology and Medicine*, vol. 108, págs. 354-370, mayo de 2019. dirección: https://www.sciencedirect.com/science/article/abs/pii/S0010482519300642.

[3] A. J. E. C. J. C. J., J. L.D. y R. C. J.S., «Automatic Calculation of Body Mass Index Using Digital Image Processing», *Communications in Computer and Information Science*, vol. 916, págs. 309-319, sep. de 2018. dirección: https://link.springer.com/chapter/10.1007/978-3-030-00353-1_28#:~:text=By%20means%20of%20a%20digital,system%20without%20alterations%20in%20samples..

[4] W. Z. S. X. y Z. Y. Y. Z. M. Y., «Leaf recognition based on PCNN», *Neural Comput and Applic*, vol. 27, págs. 899-908, abr. de 2016. dirección: https://link.springer.com/article/10.1007/s00521-015-1904-1.

[5] F. Garcia, J. Cervantes, A. López y M. Alvarado, «Fruit Classification by Extracting Color Chromaticity, Shape and Texture Features: Towards an Application for Supermarkets», *IEEE Latin America Transactions*, vol. 14, n.º 7, págs. 3434-3443, jul. de 2016, ISSN: 1548-0992. DOI: 10.1109/TLA.2016.7587652. dirección: https://ieeexplore.ieee.org/document/7587652.

[6] M. R. Carvalho, F. A. Bockmann, D. S. Amorim, C. R. F. Brandao, J. L. de Vivo M.; de Figueiredo, H. A. Britski, M. C. C. de Pinna, N. A. Menezes, F. P. L. Marques, N. Papavero, E. M. Cancello, J. V. Crisci, J. D. McEachran y R. Schelly, «Taxonomic Impediment or Impediment to Taxonomy? A Commentary on Systematics and the Cybertaxonomic-Automation Paradigm», *Evolutionary Biology*,

vol. 34, págs. 140-143, nov. de 2007. dirección: https://link.springer.com/article/10.1007/s11692-007-9011-6.

[7] J. Cervantes, F. G. Lamont, L. R. Mazahua, A. Z. Hidalgo y J. S. R. Castilla, «Complex Identification of Plants from Leaves», *International Conference on Intelligent Computing*, vol. 10956, págs. 376-387, jul. de 2018. dirección: https://link.springer.com/chapter/10.1007/978-3-319-95957-3_41.

[8] D. A. Niño, J. S. R. Castilla, M. D. A. Zenteno y L. D. Jalili, «Complex Leaves Classification with Features Extractor», *International Conference on Intelligent Computing*, vol. 11644, págs. 758-769, jul. de 2019. dirección: https://link.springer.com/chapter/10.1007/978-3-030-26969-2_72.

[9] A. Caglayan, O. Guclu y A. B. Can, «A Plant Recognition Approach Using Shape and Color Features in Leaf Images», *Image Analysis and Processing – ICIAP 2013. ICIAP 2013. Lecture Notes in Computer Science*, vol. 8157, págs. 161-170, 2013. dirección: https://link.springer.com/chapter/10.1007/978-3-642-41184-7_17.

[10] G. L. Grinblat, L. C. Uzal, M. G. Larese y P. M. Granitto, «Deep learning for plant identification using vein morphological patterns», *Computers and Electronics in Agriculture*, vol. 127, págs. 418 -424, 2016, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2016.07.003. dirección: http://www.sciencedirect.com/science/article/pii/S0168169916304665.

[11] L. Perez y J. Wang, *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*, 2017. arXiv: 1712.04621 [cs.CV].

[12] A. Aakif y M. F. Khan, «Automatic classification of plants based on their leaves», *Biosystems Engineering*, vol. 139, págs. 66 -75, 2015, ISSN: 1537-5110. DOI: https://doi.org/10.1016/j.biosystemseng.2015.08.003. dirección: http://www.sciencedirect.com/science/article/pii/S1537511015001373.

[13] C. Zhao, S. S. Chan, W.-K. Cham y L. Chu, «Plant identification using leaf shapes—A pattern counting approach», *Pattern Recognition*, vol. 48, n.º 10, págs. 3203 -3215, 2015, Discriminative Feature Learning from Big Data for Visual Recognition, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2015.04.004. dirección: http://www.sciencedirect.com/science/article/pii/S0031320315001284.

[14] S. H. Lee, C. S. Chan, S. J. Mayo y P. Remagnino, «How deep learning extracts and learns leaf features for plant classification», *Pattern Recognition*, vol. 71, págs. 1 -13, 2017, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.05.015. dirección: http://www.sciencedirect.com/science/article/pii/S003132031730198X.

[15] B. Wang y D. Wang, «Plant Leaves Classification: A Few-Shot Learning Method Based on Siamese Network», *IEEE Access*, vol. 7, págs. 151 754-151 763, 2019. dirección: https://ieeexplore.ieee.org/document/8869770.

[16] SIAP. (). Estadística de Producción Agrícola. (2019), dirección: http://infosiap.siap.gob.mx/gobmx/datosAbiertos.php.

[17] D. S. Michoacan. (). Michoacán Primer Lugar en Producción de Durazno a Nivel Nacional. (2018, October 15), dirección: https://www.gob.mx/agricultura/michoacan/articulos/michoacan-primer-lugar-en-produccion-de-durazno-a-nivel-nacional?idiom=es.

[18] Anonimo, «Establecimiento de huerto de Durazno», *SAGARPA*, mayo de 2018. dirección: https://nanopdf.com/download/establecimiento-de-huerto-de-durazno_pdf.

[19] J. L.D., M. A., C. J. y R.-C. J.S, «Improving the Performance of Leaves Identification by Features Selection with Genetic Algorithms», *Applied Computer Sciences in Engineering. WEA 2016. Communications in Computer and Information Science*, vol. 657, págs. 103-114, ene. de 2017. dirección: https://link.springer.com/chapter/10.1007/978-3-319-50880-1_10.

[20] J. C. Neto, G. E. Meyer, D. D. Jones y A. K. Samal, «Plant species identification using Elliptic Fourier leaf shape analysis», *Computers and Electronics in Agriculture*, vol. 50, n.º 2, págs. 121 -134, 2006, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2005.09.004. dirección: http://www.sciencedirect.com/science/article/pii/S0168169905001560.

[21] J.-X. Du, X.-F. Wang y G.-J. Zhang, «Leaf shape based plant species recognition», *Applied Mathematics and Computation*, vol. 185, n.º 2, págs. 883 -893, 2007, Special Issue on Intelligent Computing Theory and Methodology, ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2006.07.072. dirección: http://www.sciencedirect.com/science/article/pii/S009630030600806X.

[22] A. R. Backes y O. M. Bruno, «Plant Leaf Identification Using Multi-scale Fractal Dimension», *Foggia P., Sansone C., Vento M. (eds) Image Analysis and Processing – ICIAP 2009. ICIAP 2009. Lecture Notes in Computer Scienc*, vol. 5716, págs. 143-150, 2009. dirección: https://link.springer.com/chapter/10.1007/978-3-642-04146-4_17.

[23] M. Rashad, B. El-Desouky y M. Khawasik, «Plants Images Classification Based on Textural Features using Combined Classifier», *International Journal of Computer Science and Information Technology*, vol. 3, págs. 93-100, ago. de 2011. DOI: 10.5121/ijcsit.2011.3407. dirección: https://www.researchgate.net/publication/274175414_Plants_Images_Classification_Based_on_Textural_Features_using_Combined_Classifier.

[24] Y. Naresh y H. Nagendraswamy, «Classification of medicinal plants: An approach using modified LBP with symbolic representation», *Neurocomputing*, vol. 173, págs. 1789 -1797, 2016, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.08.090. dirección: http://www.sciencedirect.com/science/article/pii/S0925231215013764.

[25] A. Olsen, S. Han, B. Calvert, P. Ridd y O. Kenny, «In Situ Leaf Classification Using Histograms of Oriented Gradients», en *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, sep. de 2015, págs. 1-8. DOI: 10.1109/DICTA.2015.7371274. dirección: https://ieeexplore.ieee.org/document/7371274.

[26] M. Tico, T. Haverinen y P. Kuosmanen, «A method of color histogram creation for image retrieval», ene. de 2000. dirección: https://www.researchgate.net/publication/244449350_A_method_of_color_histogram_creation_for_image_retrieval.

[27] J. Cervantes, J. Taltempa, F. García-Lamont, J. S. R. Castilla, A. Y. Rendon y L. D. Jalili, «Análisis Comparativo de las técnicas utilizadas en un Sistema de Reconocimiento de Hojas de Planta», *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 14, n.º 1, págs. 104 -114, 2017, ISSN: 1697-7912. DOI: https://doi.org/10.1016/j.riai.2016.09.005. dirección: http://www.sciencedirect.com/science/article/pii/S1697791216300516.

[28] H. A. Elnemr, «Feature selection for texture-based plant leaves classification», en *2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems 2017 Intl Conf*

on New Paradigms in Electronics Information Technology (PEIT), 2017, págs. 91-97. dirección: https://ieeexplore.ieee.org/document/8303025.

[29] M. G. Larese, A. E. Bayá, R. M. Craviotto, M. R. Arango, C. Gallo y P. M. Granitto, «Multiscale recognition of legume varieties based on leaf venation images», *Expert Systems with Applications*, vol. 41, n.º 10, págs. 4638 -4647, 2014, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.01.029. dirección: http://www.sciencedirect.com/science/article/pii/S0957417414000529.

[30] S. H. Lee, C. S. Chan, P. Wilkin y P. Remagnino, «Deep-plant: Plant identification with convolutional neural networks», en *2015 IEEE International Conference on Image Processing (ICIP)*, sep. de 2015, págs. 452-456. DOI: 10.1109/ICIP.2015.7350839. dirección: https://ieeexplore.ieee.org/document/7350839.

[31] M. Paco Ramos, V. Paco Ramos, A. Loaiza Fabian y E. Osco Mamani, «A Feature Extraction Method Based on Convolutional Autoencoder for Plant Leaves Classification», en *2019 IEEE Colombian Conference on Applications in Computational Intelligence (ColCACI)*, jun. de 2019, págs. 1-6. DOI: 10.1109/ColCACI.2019.8781985. dirección: https://ieeexplore.ieee.org/document/8781985.

[32] D. Charrez. (). NeurIPS 2019 Stats. (2019, September 6), dirección: https://medium.com/@dcharrezt/neurips-2019-stats-c91346d31c8f.

[33] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak e I. Sutskever, *Deep Double Descent: Where Bigger Models and More Data Hurt*, 2019. arXiv: 1912.02292 [cs.LG]. dirección: https://arxiv.org/abs/1912.02292.

[34] J. Schmidhuber, «Deep learning in neural networks: An overview», *Neural Networks*, vol. 61, págs. 85 -117, 2015, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. dirección: http://www.sciencedirect.com/science/article/pii/S0893608014002135.

[35] S. Theodoridis, *Machine Learning A Bayesian and Optimization Perspective*. Academic Press, 2020, ISBN: 9780128188033. dirección: https://www.elsevier.com/books/machine-learning/theodoridis/978-0-12-818803-3#:~:text=Description,learning%2C%20namely%20regression%20and%20classification..

[36] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[37] S. Ruder, *An overview of gradient descent optimization algorithms*, 2016. arXiv: 1609.04747 [cs.LG]. dirección: https://arxiv.org/abs/1609.04747.

[38] N. Qian, «On the momentum term in gradient descent learning algorithms», *Neural Networks*, vol. 12, n.º 1, págs. 145 -151, 1999, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. dirección: http://www.sciencedirect.com/science/article/pii/S0893608098001166.

[39] J. Duchi, E. Hazan e Y. Singer, «Adaptive Subgradient Methods for Online Learning and Stochastic Optimization», *Journal of Machine Learning Research*, vol. 12, págs. 2121-2159, jul. de 2011. dirección: https://www.researchgate.net/publication/220320677_Adaptive_Subgradient_Methods_for_Online_Learning_and_Stochastic_Optimization.

[40] M. D. Zeiler, *ADADELTA: An Adaptive Learning Rate Method*, 2012. arXiv: 1212.5701 [cs.LG]. dirección: https://arxiv.org/abs/1212.5701.

[41] D. P. Kingma y J. Ba, *Adam: A Method for Stochastic Optimization*, 2014. arXiv: 1412.6980 [cs.LG]. dirección: https://arxiv.org/abs/1412.6980.

[42] S. R. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh y B. B. Chaudhuri, *diffGrad: An Optimization Method for Convolutional Neural Networks*, 2019. arXiv: 1909.11015 [cs.LG]. dirección: https://arxiv.org/abs/1909.11015#:~:text=diffGrad%3A%20An%20Optimization%20Method%20for%20Convolutional%20Neural%20Networks,-Shiv%20Ram%20Dubey&text=Stochastic%20Gradient%20Decent%20(SGD)%20is,the%20steepest%20rate%20of%20change..

[43] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard y L. D. Jackel, «Backpropagation Applied to Handwritten Zip Code Recognition», *Neural Computation*, vol. 1, n.º 4, págs. 541-551, 1989. dirección: https://ieeexplore.ieee.org/document/6795724.

[44] M. D. Zeiler y R. Fergus, «Visualizing and Understanding Convolutional Networks», *CoRR*, vol. abs/1311.2901, 2013. arXiv: 1311.2901. dirección: http://arxiv.org/abs/1311.2901.

[45] V. Nair y G. E. Hinton, «Rectified Linear Units Improve Restricted Boltzmann Machines», en *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ép. ICML'10, Omnipress, 2010, 807–814, ISBN: 9781605589077. dirección: https://dl.acm.org/doi/10.5555/3104322.3104425.

[46] Zhou y Chellappa, «Computation of optical flow using a neural network», en *IEEE 1988 International Conference on Neural Networks*, 1988, 71-78 vol.2. dirección: https://ieeexplore.ieee.org/document/23914.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting», *Journal of Machine Learning Research*, vol. 15, n.º 56, págs. 1929-1958, 2014. dirección: http://jmlr.org/papers/v15/srivastava14a.html.

[48] S. Ioffe y C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift», *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. dirección: http://arxiv.org/abs/1502.03167.

[49] X. Glorot e Y. Bengio, «Understanding the difficulty of training deep feedforward neural networks», Y. W. Teh y M. Titterington, eds., ép. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: JMLR Workshop y Conference Proceedings, mayo de 2010, págs. 249-256. dirección: http://proceedings.mlr.press/v9/glorot10a.html.

[50] K. He, X. Zhang, S. Ren y J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015. arXiv: 1502.01852 [cs.CV]. dirección: https://arxiv.org/abs/1502.01852.

[51] I. J. Goodfellow, «Technical Report: Multidimensional, Downsampled Convolution for Autoencoders», Université de Montréal, inf. téc., 2010. dirección: http://www.iro.umontreal.ca/~lisa/publications2/index.php/publications/show/605.

[52] Y. Taigman, M. Yang, M. Ranzato y L. Wolf, «DeepFace: Closing the Gap to Human-Level Performance in Face Verification», en *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, págs. 1701-1708. dirección: https://ieeexplore.ieee.org/document/6909616.

[53] L. V. Utkin, M. S. Kovalev y E. M. Kasimov, *An explanation method for Siamese neural networks*, 2019. arXiv: 1911.07702 [cs.LG]. dirección: https://arxiv.org/abs/1911.07702.

[54] J. Bromley, J. Bentz, L. Bottou, I. Guyon, Y. Lecun, C. Moore, E. Sackinger y R. Shah, «Signature Verification using a "Siamese"Time Delay Neural Network», *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, pág. 25, ago. de 1993. DOI: 10.1142/S0218001493000339. dirección: https://www.researchgate.net/publication/270283023_Signature_Verification_using_a_Siamese_Time_Delay_Neural_Network.

[55] X. Glorot e Y. Bengio, «Understanding the difficulty of training deep feedforward neural networks», Y. W. Teh y M. Titterington, eds., ép. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: JMLR Workshop y Conference Proceedings, mayo de 2010, págs. 249-256. dirección: http://proceedings.mlr.press/v9/glorot10a.html.

[56] A. Kendall e Y. Gal, «What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?», *CoRR*, vol. abs/1703.04977, 2017. arXiv: 1703.04977. dirección: http://arxiv.org/abs/1703.04977.