UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

REDES NEURONALES CONVOLUCIONALES HERMITIANAS

TESIS

QUE PARA OPTAR POR EL GRADO DE MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:

LEONARDO LEDESMA DOMINGUEZ

TUTOR:
DR. BORIS ESCALANTE RAMÍREZ
POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACION

[CIUDAD DE MÉXICO, AGOSTO, 2019]

וַיְדַבֵּר יהוה אֶל־מֹשֶׁה לֵאמֹר : דַּבֵּר אֶל־אַהֲרֹן וְאֶל־בָּנָיו לֵאמֹר כֹּה תְבָרֲכוּ
פָּנָיו אֵלֶיךָ אֶת־בְּנֵי יִשְׂרָאֵל אָמוֹר לָהֶם :יְבָרֶכְךָ יהוה וְיִשְׁמְרֶךָ : יָאֵר יהוה
וִיחֻנֶּךָּ :יִשָּׂא יהוה פָּנָיו אֵלֶיךָ וְיָשֵׂם לְךָ שָׁלוֹם

Agradecimientos

Como teísta quiero agradecer a Dios, porque siempre ha estado conmigo y a pesar de que muchas veces he caído, el siempre me ha levantado. Y porque me ha permitido cumplir una meta más que es realizar una maestría. Por darme una visión integral, diferente, profundamente espiritual de que la vida es más que lo superficial y lo material.

A mi padre Rogelio Ledesma González, el mejor ejemplo que pude tener para enfrentar los problemas, por todos los consejos y por su apoyo. A mi madre Irene Domínguez Reyes, sin duda, un ejemplo de vida y fe, porque me ha demostrado que no importa que tan difícil sea la situación siempre es posible resistir y superarla, por ensañarme todo lo que sé de Dios.

A mi hermana Abigail Ledesma, simplemente por estar aquí, porque el hecho de estar aquí cada día es un milagro. Por su amor tan único y porque el simple hecho de verla feliz me llena de fuerzas y alegría.

A Diana Barrera por ser un apoyo en este camino, por escucharme y por darme consejos. A Francisco Javier Lozano Espinosa que, gracias a su ejemplo, he aprendido muchas cosas sobre el trabajo en equipo y que nada es imposible cuando de tareas difíciles se trata de resolver, que los obstáculos son mentales y de actitud, y que hay que salir siempre adelante.

A mi tutor de maestría el Dr. Boris Escalante Ramírez, por apoyarme y dirigir este proyecto de investigación. A mis amigos del LAPI: a la Dra. Jimena Olveres, Erik Carbajal, Vivian Triana, Germán González y Erika Mendoza.

Al profesor el Dr. Gibrán Fuentes Pineda, por ser un excelente profesor en las materias del posgrado y ser pieza fundamental en entender los temas centrales de esta tesis.

Finalmente, a la Universidad Nacional Autónoma de México y al posgrado de Ciencias e Ingeniería de la Computación por su alta calidad académica.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT), por el apoyo económico que me proporcionó durante mis estudios de maestría.

Al programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT), específicamente financiando los proyectos con las claves IA103119 y IN116917.

Resumen

Las redes neuronales convolucionales se han convertido en una metodología fundamental en Visión Computacional, específicamente en las tareas clasificación de imágenes y detección de objetos. Muchas de las investigaciones en Inteligencia Artificial han centrado sus esfuerzos en las diferentes áreas de las redes neuronales convolucionales. Recientes investigaciones han demostrado que proporcionar conocimiento a priori a una red convolucional ayuda a mejorar el rendimiento, a reducir el número de parámetros y costo computacional.

Por otro lado, la transformada de Hermite es una herramienta matemática que extrae características relevantes útiles para la tarea de clasificación.

Este trabajo presenta un enfoque novedoso combinando una red convolucional con la transformada de Hermite, a esta nueva arquitectura se le llama red neuronal convolucional Hermitiana (HCN).

La HCN busca mantener las ventajas de las redes convolucionales manteniendo un modelo compacto de Deep Learning sin perder la alta capacidad de representación de características.

Abstract

Convolutional Neuronal Networks (CNNs) have become a fundamental methodology in Computer Vision, specifically in image classification and object detection tasks. Artificial Intelligence has focused much of its efforts in the different research areas of CNN. Recent works has demonstrated that providing CNNs with a priori knowledge helps them improve their performance while reduce the number of parameters and computing time. On the other hand, the Hermite transform is a useful mathematical tool that extracts relevant image features useful for classification task.

This work presents a novel approach to combine CNNs with the Hermite transform, namely, Hermite Convolutional Networks (HCN). Furthermore, the proposed HCNs keep the advantages of CNN while leading to a more compact deep learning model without losing a high feature representation capacity.

Índice general

Índice de figuras

Índice de tablas

Notación

Arreglos y matrices

x	un valor escalar
$\mathbf{x}$	un vector
A	un tensor
$\mathbf{A}$	una matriz

Indexadación

$x^{(i)}$	el *í-esimo* elemento de una entrada de datos
$\mathbf{W}^{(i)}$	el *í-esimo* elemento de una matriz respecto al número de capa asociado
$\mathbf{W}_{i,j}$	el elemento i,j de una matriz
O	una dimensión tensorial, cuando esta contenida en una notación del tipo ()

Operaciones de algebra lineal, cálculo y numéricas

$\frac{\partial y}{\partial x}$	la derivada parcial de y con respecto a x
$\frac{dy}{dx}$	la derivada ordinaria de y con respecto a x
$a!$	el factorial de un número
$\langle \mathbf{a}, \mathbf{b} \rangle$	proyección vectorial del vector $\mathbf{a}$ sobre $\mathbf{b}$
$A \circ B$	un producto Hadamard entre dos tensores
$\mathbf{A} \circ \mathbf{B}$	un producto Hadamard entre dos matrices
μ	la media aritmética
σ	la desviación estándar
ϕ	una base funcional
$\int f(x)\, dx$	una integral definida sobre el dominio de x
$\iint f(x, y)\, dx\, dy$	una doble integral definida sobre el dominio de x y y
$A \otimes B$	una convolución entre dos tensores
$\mathbf{A} \otimes \mathbf{B}$	una convolución entre dos matrices

Palabras clave y abreviaturas

CNN - Convolutional Neural Network
RNN - Recurrent Neural Network
GAN - Generative Adversarial Network
CGAN - Convolutional Generative Adversarial Network
ML - Machine Learning
DL - Deep Learning
MLP - Multi Layer Perceptron
SVM - Support Vector Machine
HMM - Hidden Model Markov
LOOV - Leave one out cross validation
FC - Fully Connected
TL - Transfer Learning
FT - Fine Tunning
FF - Feed Forward
BP - Back Propagation
BN - Batch Normalization
D - Dropout
MP - Max Pooling
ResNet - Residual Network
HT - Hermite Transform
HB - Hermite filter Bank
LF- Learned Filter
HoF - Hermite Orientation Filter
Hconv - Hermite Convolution

Repositorios de códigos fuentes:
Códigos propios:
https://github.com/LeonardoLed/HermiteFilters.git
https://github.com/LeonardoLed/ExtensionHermiteResNet.git

Código de apoyo:
https://github.com/bczhangbczhang/Gabor-Convolutional-Networks.git

Capítulo 1

Introducción

*So many academics forget that our goal, as a profession, is *not* to publish papers. It's to change the world*

Emin Gün Sirer, computer scientist.
co-director of IC3, The Initiative For Cryptocurrencies Contracts

El surgimiento de las redes neuronales convolucionales revolucionó la manera de resolver tareas específicas de visión computacional como son:

- Clasificación de imágenes.
- Reconocimiento y localización de objetos.
- Clasificación y reconocimiento de acciones
- Reconocimiento de rostros.
- Segmentación de imágenes.

En 2012 la red AlexNet[1] redujo a casi nueve puntos porcentuales la tasa de error de clasificación de imágenes en el "The Image Classificacion Challenge"[2], como se muestra en la Figura 1.1.

Paralelo al desarrollo de nuevas arquitecturas de redes convolucionales se han utilizado herramientas matemáticas para el procesamiento de imágenes, como son:

- Transformada de Gabor.
- Transformada de Hermite.
- Transformada Wavelets.
- Basados en polinomios ortogonales discretos.

La transformada de Hermite está basada en el sistema de visión humano. Se ha comprobado que la corteza visual trabaja a través de campos receptivos que son similares a derivadas de gaussianas (que se pueden modelar con diferencias de gaussianas DoG, en sus siglas en inglés) [3][4] y que justamente es parte del fundamento matemático de la transformada de Hermite.

1.1. Motivación

Se han desarrollado varios trabajos utilizando la transformada de Hermite para el tratamiento y procesamiento de imágenes [5][6][7] donde se han tenido buenos resultados de segmentación y clasificación.

La idea de este trabajo se basa en sustituir los filtros convencionales en una red neuronal convolucional por filtros de la transformada de Hermite con la hipótesis de mejorar el rendimiento de la misma, tanto a nivel de número de parámetros hasta la extracción de características más eficientes.

En [35] se reporta uno de los primeros intentos de introducir filtros basados en transformadas

ortogonales (en particular la transformada de Gabor). En ese trabajo se analizan los filtros aprendidos en la red AlexNet al final del entrenamiento (ver Figura 1.2) y se concluye que la mayoría de los filtros son redundantes y de características similares a los filtros de Gabor.

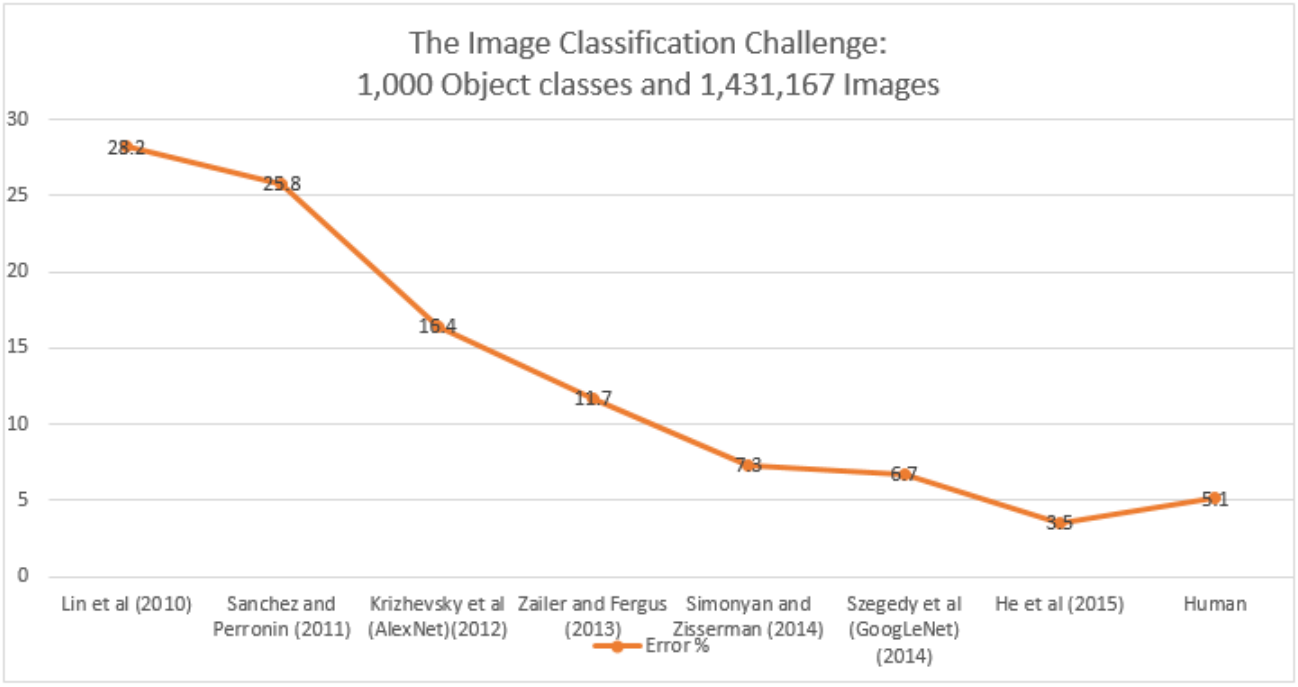

Figura 1.1: Estados del arte en the Image Classification Challenge [ImageNet], se muestra la evolución de la taza de error a lo largo del tiempo [2]. Cabe mencionar que las métricas para evaluar el rendimiento de clasificación en ImageNet han cambiado desde el 2015 y se pueden analizar en [8]. Imagen tomada del curso: "Convolutional Neural Networks for Visual Recognition (Spring 2017)"de la Univesidad de Stanford

¿Por qué no introducir los filtros basados en transformadas desde el inicio del entrenamiento y de alguna manera ir transfiriendo sus propiedades en todas las capas convolucionales?. Se espera que al aplicar este principio se logre una mejora en la extracción de características y por consiguiente una mejor clasificación.

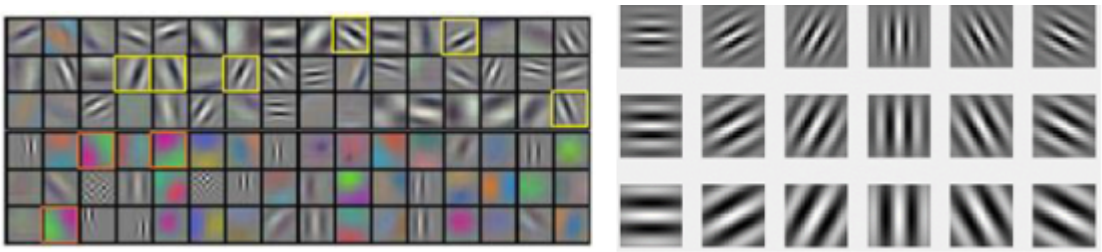

Figura 1.2: Del lado derecho se muestran algunos ejemplos de filtros de Gabor y del lado izquierdo los filtros convolucionales de la red AlexNet después del entrenamiento. Como se puede observar los filtros resaltados en amarrillo son similares a los filtros de Gabor. Imagen modificada de Chen et al [35]

La idea anterior se ha intentado llevar a cabo mediante diferentes maneras:

- Preprocesamiento de las imágenes antes de ser trabajado por la CNN[9].
- Codificación jerárquica de orientaciones utilizando filtros rotacionales activos [10].
- Usando wavelets para obtener la expresión de campos receptivos en una CNN[11][12].
- Inicializando los filtros convolucionales en filtros de Gabor en vez de ser inicializarlos de manera aleatoria[13].

Este proyecto se basa en la definición de un proceso de modulación y un proceso de convolución no convencional, ajustados para trabajar con filtros de Hermite.

1.2. Objetivos

El objetivo principal de esta tesis es:

Introducir filtros de la transformada de Hermite dentro una CNN para optimizar su rendimiento en clasificación de imágenes.

Los objetivos secundarios que se desprenden del principal son:

1. Obtener una arquitectura básica como un clasificador sencillo y fácilmente adaptable a los parámetros de diseño de filtros de Hermite.
2. Ajustar una arquitectura tipo ResNet para que trabaje con filtros de Hermite, para imágenes más complejas de 3 canales.
3. Comparar las arquitecturas propuestas con los reportados en la literatura. En particular:
   a) Otros modelos de CNN.
   b) Basados en filtros de Gabor.

1.3. Clasificación de imágenes

Es importante denotar la diferencia entre clasificación y regresión:

- Clasificación: el objetivo de la clasificación es determinar la clase (etiqueta) a la que pertenece un dato de entrada, dado que el número de clases es fijo y es un valor entero, la clasificación es un modelo de predicción discreto.

  Aunque la salida de un modelo de clasificación regresa un conjunto de valores reales o un valor real (según sea el tipo de clasificación), estos hacen referencia a la probabilidad de pertenencia a cada una de las clases o clase. Es decir, de salida de tipo cualitativa.

- Regresión: el objetivo de la regresión es predecir un valor real numérico dada una entrada, por lo que una regresión es un modelo de predicción continuo. Por ejemplo: el precio de una casa dada una serie de atributos relativos a la casa. Es decir, salida del tipo cuantitativa.

  Dado un tipo de clasificación o regresión también es importante mencionar los tipos de modelos según el conjunto de variables de entrada:

  - Simple: Análisis de clasificación o regresión tomando en cuenta una sola variable.
  - Multivariable: Análisis de clasificación o regresión considerando un conjunto de variables que describen los datos de entrada o la naturaleza del problema.

1.3.1. Tipos de clasificación

Otras de las características fundamentales que es necesario describir, son los tipos de clasificación en relación con su entrada- salida que se puede observar en Tabla 2.1, y que se describen a continuación:

Clasificación Binaria

El objetivo de la clasificación binaria es determinar si un dato de entrada pertenece o no a una y sólo una clase.

Clasificación Multiclase

Se encarga de predecir la pertenencia de un dato de entrada a una clase; la diferencia de la clasificación anterior es que hay más de una clase. (pertenencia exclusiva).

Clasificación Multi-Etiqueta

Un dato de entrada puede pertenecer a varias clases de manera simultánea, es decir la pertenencia no es exclusiva.

1.3.2. Retos de la clasificación de imágenes

Dentro de la clasificación de imágenes hay varios retos aún por ser resueltos y mejorados. Estos retos van asociados por la naturaleza de las imágenes y de su adquisición, tal y como se muestra en la Figura 1.3

a) Iluminación: el cambio de contrastes de luz en imágenes diferentes con objetos iguales puede propiciar a errores de clasificación.

b) Deformación: un objeto puede estar en diferentes formas en una imagen, ya sea por la ubicación en un cierto ángulo o la perspectiva de cómo se adquirió la imagen.

c) Oclusión: un objeto dentro la imagen puede estar parcialmente representado, ya sea porque otros objetos obstaculizan la visión del objeto o bien sólo se muestra una parte.

d) Desorden: ya sea por un patrón regular (textura) o por la cantidad de objetos fuera de interés que hacen más compleja la detección de un objeto dentro de una imagen.

e) Variación intraclase: un objeto puede tener diferentes subclases, y varios objetos de esas subclases pueden estar contenidas en la imagen.

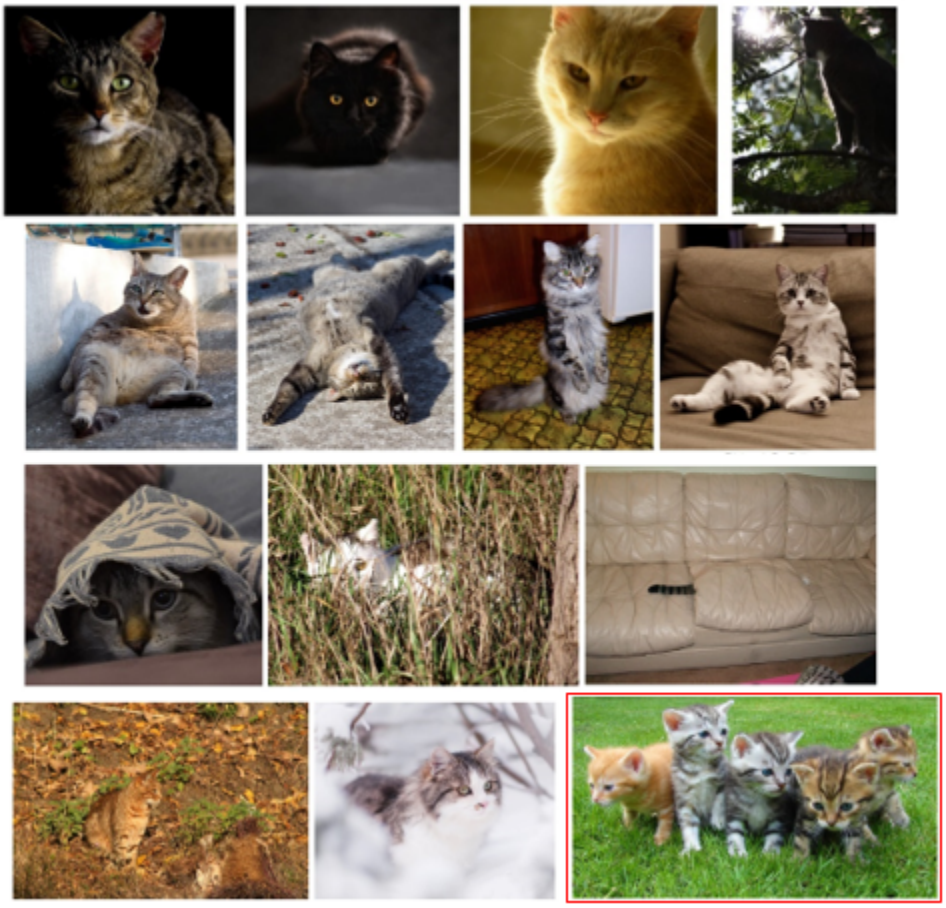

Figura 1.3: Estas imágenes muestran los retos a los que se enfrentan la clasificación de imágenes, la primera fila muestra el problema de la iluminación, la segunda el problema de la deformación; lo que se muestra que a pesar de que trate un mismo objeto (un gato) la modelación de aprendizaje de una red convolucional puede ser muy distinta. En la tercera fila se muestra el problema de oclusión y la última fila el problema del desorden, que son problemas aún más difíciles porque incluso para la visión humana es complicado determinar que es cada objeto. Y la imagen del recuadro rojo muestra el problema de variación intraclases, en este caso las diferentes razas de un gato. Estas imágenes son tomadas del curso: "Convolutional Neural Networks for Visual Recognition (Spring 2017)" de la Univesidad de Stanford.

Capítulo 2

Redes Neuronales Convolucionales (CNN)

By far, the greatest danger of Artificial Intelligence is that people conclude too early that they understand it.

Eliezer Yudkowsky, American AI researcher

2.1. Antecedentes

La siguiente frase podría resumir hacia donde se han dirigido los esfuerzos del desarrollo de la inteligencia artificial en la resolución de tareas que desde el punto de vista del ser humano son tan naturales.

Ironically, abstract and formal tasks that are among the most difficult mental undertakings for a human being are among the easiest for a computer. Computers have long been able to defeat even the best human chess player, but are only recently matching some of the abilities of average human beings to recognize objects or speech. A person's everyday life requires an immense amount of knowledge about the world. Much of this knowledge is subjective and intuitive, and therefore difficult to articulate in a formal way. Computers need to capture this same knowledge in order to behave in an intelligent way. One of the key challenges in artificial intelligence is how to get this informal knowledge into a computer.[14]

Como menciona la frase, el objetivo vital de la Inteligencia Artificial es traducir un conjunto de conocimiento informal, intuitivo y subjetivo a un lenguaje formal que la computadora pueda "entender"; y para cumplir este objetivo, han existido diversas corrientes de pensamiento y metodologías.

Dichas discusiones van relacionadas al aprendizaje, el conocimiento, la experiencia y el propio concepto de inteligencia que incluso van más allá del campo de las ciencias de la computación, que tienden hacer cuestiones más filosóficas y neurocientíficas. Se podría hablar de dos corrientes o escuelas de IA: la Inteligencia Artificial Convencional y la Inteligencia Artificial Computacional.

Es difícil no hablar de conocimiento cuando se habla de aprendizaje, por ello también se habla de dos escuelas de aprendizaje generales, explicadas bajo el concepto del conocimiento:

- *A priori* o conocimiento innato: establece que el conocimiento de una tarea específica es innato y se va desarrollando a lo largo del tiempo, de acuerdo con las necesidades adaptativas del sujeto en su medio. El proceso de aprendizaje sólo va dirigido a la corrección de una acción para la mejora del conocimiento constante.

- Conocimiento adquirido: el sujeto analiza su entorno y en un proceso de aprendizaje constante adquiere el conocimiento que le ayudará a tener una mejor experiencia en situaciones similares.

Estas ideas se han trasladado al campo de la IA a través de diferentes metodologías bioinspiradas [20][21], específicamente dentro del aprendizaje profundo, las redes neuronales. Como se ha mencionado en la introducción esta tesis se busca agregar conocimiento *a priori* a los filtros de una red convolucional a través de la transformada de Hermite.

Las redes neuronales como modelo-metodología [22] de aprendizaje profundo a lo largo de la historia han tenido altas y bajas, tiempos donde fueron populares y tiempos donde fueron prácticamente descreditadas. En el apartado siguiente se dará una breve explicación de

los pasos históricos que dieron origen a las redes neuronales hasta la idea del aproximador universal.

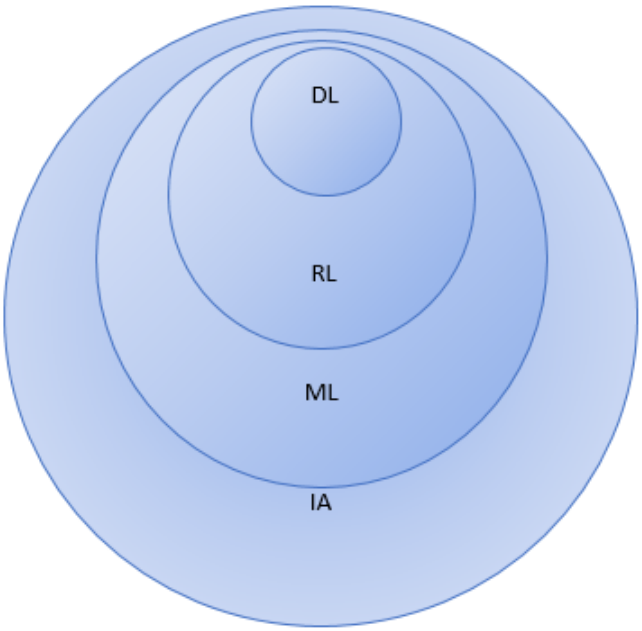

Figura 2.1: Clasificación de los algoritmos de aprendizaje según el nivel de representación de características, IA: Inteligencia Artificial, ML: *Machine Learning*, RL: *Representation Learning*, DL: *Deep Learning*. Imagen tomada del libro *Deep Learning* de Ian Goodfellow.[14]

2.2. Perceptrón multicapa (MLP)

Los algoritmos de aprendizaje automatizado (*Machine Learning* - ML) dependen fuertemente de la representación que de los datos que se establezca, es decir del modelado directo que se tenga sobre las características, variables que conforman el problema de análisis.

El diseño de características (*features*) es un tema de relevancia de la IA para resolver un problema, a esta área se le conoce como "*Representation Learning*". Con este conjunto de características se busca describir el comportamiento de los datos y que sean fácilmente adaptables a nuevas tareas sin intervención humana. El aprendizaje profundo (*Deep Learning* -DL) resuelve el problema central de aprendizaje de características, construyendo características complejas en términos de características más sencillas; en el caso de las redes neuronales convolucionales se trabaja con la idea que primero se extraen esquinas, bordes y formas en las primeras capas de la red, después avanzado, con partes de objetos, objetos completos y escenarios en las últimas capas (Ver Figura 2.3).

Históricamente, después de que Minsk y Papert [15] en su libro Perceptrons en 1969, plantearon el famoso problema del XOR para los perceptrones de Rosenblatt [16] (basados en los

trabajos de McCulloch and Pitts Neuron [17]), inició una etapa difícil de desinterés por las redes neuronales. Básicamente lo que mostraba el problema del XOR, es que los perceptrones no podían resolver problemas linealmente no separables.

Fue el mismo Rosenblatt que después planteo la idea de un perceptrón de tres capas (capa de entrada, salida y una capa oculta) para resolver el problema de la XOR, dando lugar al primer perceptrón multicapa, pero en su tiempo esa idea no era fácilmente aplicable tanto en dos vías: por la capacidad de hardware que en ese momento era insuficiente y las bases de datos para entrenar las redes neuronales.

Es así con el avance de la tecnología computacional y la producción de grandes volúmenes de información que las redes neuronales MLP resurgieron con más fuerza. Los MLPs también son conocidas como redes totalmente conectadas (FC- *fully connected*), y entran dentro del modelo de *feedforward artificial neuronal network* (es el tipo de red neuronal artificial más sencilla la cual no contiene ciclos y la información se mueve en un sólo sentido, hacia adelante).

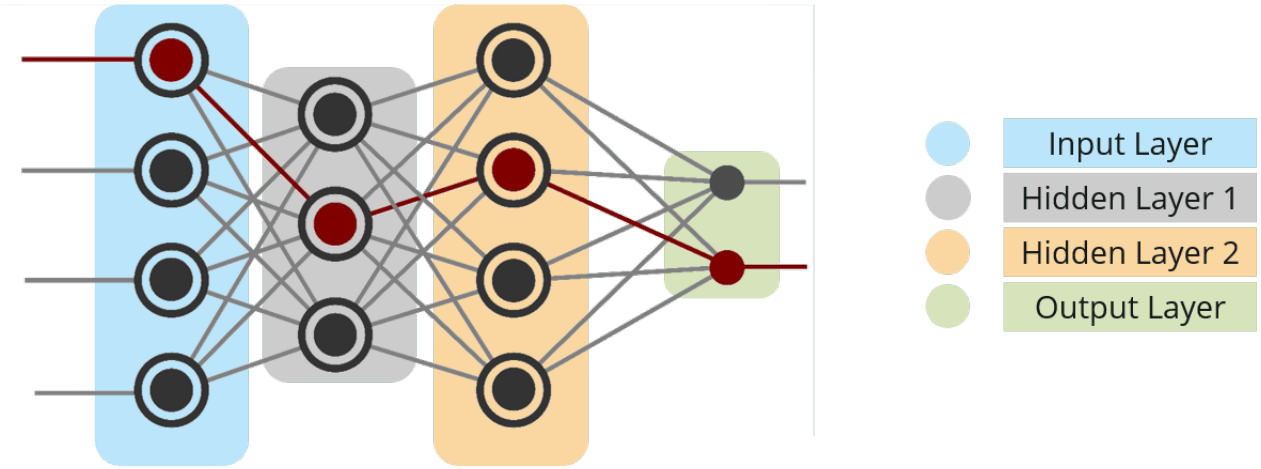

Figura 2.2: Estructura general de un MLP, donde claramente se define a través de una capa de entrada y una de salida y una serie de capas ocultas, imagen tomada de edureka.co, del curso *Multi Layer Perceptron* [18]

De manera general el MLP funcionan en subunidades de procesamiento (neuronas) de entrada y salida conectadas entre sí, aquellas neuronas que están ubicadas en las capas ocultas se encargan de extraer aquellas características fundamentales para la resolución del problema. Al igual que al modelo de perceptrón sencillo cada salida procesada (excepto las de la capa de entrada) pasa por una función de activación.

La función de activación en una MLP es una función no lineal (generalmente); la idea de pasar las salidas de cada neurona a una función de activación va relacionada con los potenciales de acción de las neuronas biológicas.

Los primeros MLPs con el algoritmo de retro propagación fueron aplicados por primera vez

en 1986 con Rumelhart [19]. Tanto las funciones de activación y el algoritmo de retro prograpagación se verán con más detalles.

2.3. Aproximador universal y el problema de la generalización

Como se discutió el apartado anterior el problema fundamental con el que se enfrentan las redes neuronales y cualquier otra metodología de ML y DL, es el problema de la Generalización, que de cierta manera es complicada definirlo, pero se podría entender de la siguiente manera:

Generalización: Es la capacidad de un algoritmo de ser efectivo a través de un rango de entradas. O bien la capacidad de representar características generales que se cumplen en la definición de varios problemas.

Hablando un poco de las funciones biológicas de las neuronas, la generalización sería equivalente a la "plasticidad neuronal" como la capacidad de poder generalizar conocimiento y ser adaptativo a entradas de diferentes estímulos del entorno.

Desde el punto de vista de la IA, la generalización se ha podido resolver parcialmente mediante la transferencia de conocimiento (*Transfer Learning*), el cual consiste en aprovechar los pesos de una red ya entrenada y ajustarlos (*Fine Tunning*) para reconocer o clasificar otro tipo de objetos o resolver otro tipo de tareas.

Otra idea fundamental es la del teorema del aproximador universal que establece que: Una red neuronal con una sola capa oculta con un número finito de neuronas puede aproximar cualquier función continua en conjuntos compactos en R^n [20].

En términos de aprendizaje, el aproximador universal establece que una red neuronal puede aprender cualquier cosa con una sola capa oculta con el número suficiente de neuronas. Dicha idea sigue siendo muy abstracta, y la tendencia en DL, ha sido crecer en profundidad más que en anchura, es decir agregar capas y capas, en vez de trabajar con pocas capas con un gran número de neuronas.

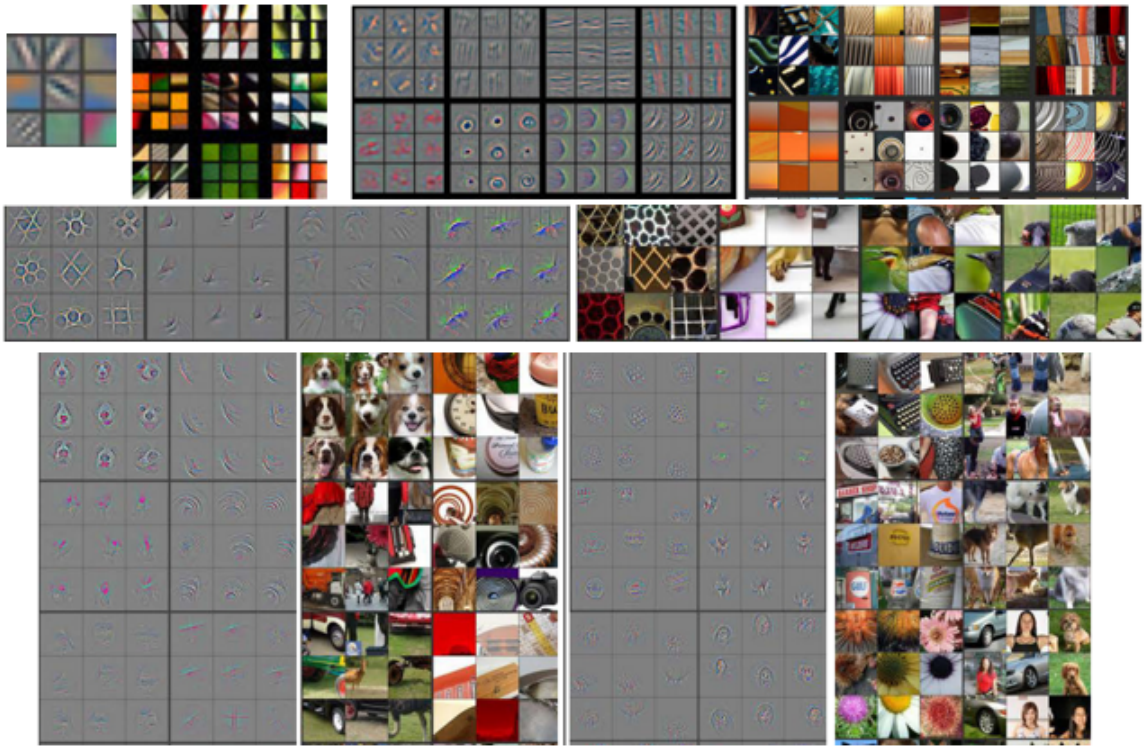

Figura 2.3: Esta imagen muestra la visualización de los filtros en una red entrada de cinco capas convolucionales en datos de validación. Se muestra la relación entre filtros y el fragmento de la imagen analizada. Como se puede observar en la primera capa se extraen características primitivas de una imagen, bordes y esquinas, en la segunda capa contornos, en la tercera texturas y estructuras más complejas. Y finalmente en la cuarta y quinta capa, ya son partes de objetos. Esta imagen esta reportada en [20].

2.3.1. Fundamentos computacionales y matemáticos

Algoritmo de propagación hacia adelante

El primer algoritmo básico para el proceso de aprendizaje de una red neuronal es la propagación hacia adelante (FF- *Feed Forward*), y el algoritmo sólo consiste en evaluar las entradas, ejecutar las operaciones (suma pesada o la establecida) y finalmente trasladar la salida en una función de activación. Para ser sencilla la explicación se utilizará un ejemplo práctico de una pequeña red que se puede observar en la Figura 2.4

Algoritmo de propagación hacia atrás

El segundo algoritmo básico es el algoritmo de retro propagación hacia atrás (BP- *Back Propagation*), y se emplea una vez que la salida total pasó por una función de activación. Utilizando una función de error o también llamada función de pérdida.

Se utiliza la regla de cadena para retro pagar el error hacia atrás, esto con la intención de

utilizar las derivadas parciales más próximas y reciclar el resultado, para no tener que realizar todo el cálculo, como medio de optimización computacional. Del ejemplo de la Figura 2.4, por *Feed Forward* se tiene:

$$a^{(1)} = xz^{(2)} = W^{(1)} \cdot a^{(1)} \rightarrow a^{(2)} = h(z^{(2)})$$

$$z^{(3)} = W^{(2)} \cdot a^{(2)} \rightarrow a^{(3)} = h(z^{(3)})$$

Por *Back Propagation* se tiene:

Usando la función de pérdida con ECM (error cuadrático medio)

$$E = \sum \frac{1}{2}(y - \hat{y})^2$$

Se actualizan pesos hacia atrás empezando con los del conjunto $W^{(2)}$:

$$\frac{\partial E}{\partial W^{(2)}} = \frac{\partial(\sum \frac{1}{2}(y-\hat{y})^2)}{\partial W^{(2)}} = (y-\hat{y}) \cdot (-\frac{\partial \hat{y}}{\partial W^{(2)}}) = (y-\hat{y}) \cdot \left(-\frac{\partial \hat{y}}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial W^{(2)}}\right)$$

$$= -(y-\hat{y}) \cdot \frac{\partial \hat{y}}{\partial z^{(3)}} \cdot \frac{\partial (W^{(2)} \cdot a^{(2)})}{\partial W^{(2)}} = -(y-\hat{y}) \cdot \frac{\partial \hat{y}}{\partial z^{(3)}} \cdot a^{(2)}$$

$$\delta^{(3)} = -(y-\hat{y}) \cdot \frac{\partial \hat{y}}{\partial z^{(3)}}$$

Se actualizan pesos hacia atrás con los del conjunto $W^{(1)}$:

$$\frac{\partial E}{\partial W^{(1)}} = \frac{\partial \left(\sum \frac{1}{2}(y-\hat{y})^2\right)}{\partial W^{(1)}} = (y-\hat{y}) \cdot \left(-\frac{\partial \hat{y}}{\partial W^{(1)}}\right) = (y-\hat{y}) \cdot \left(-\frac{\partial \hat{y}}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial W^{(2)}}\right)$$

$$= \delta^{(3)} \cdot \frac{\partial z^{(3)}}{\partial W^{(1)}} = \delta^{(3)} \cdot \left(\frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial W^{(1)}}\right)$$

$$= \delta^{(3)} \cdot \left(\frac{\partial (W^{(2)} \cdot a^{(2)})}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial W^{(1)}}\right) = \delta^{3} \cdot \left(W^{(2)} \cdot \frac{\partial a^{(2)}}{\partial W^{(1)}}\right)$$

$$= \delta^{3} \cdot W^{(2)} \left(\frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W^{(1)}}\right) = \delta^{(3)} \cdot W^{(2)} \left(\frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial (W^{(1)} \cdot a^{(1)})}{\partial W^{(1)}}\right)$$

$$= \delta^{(3)} \cdot W^{(2)} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot x$$

Si la función de error fuera otra se utilizaría el mismo procedimiento usando la regla de la cadena.

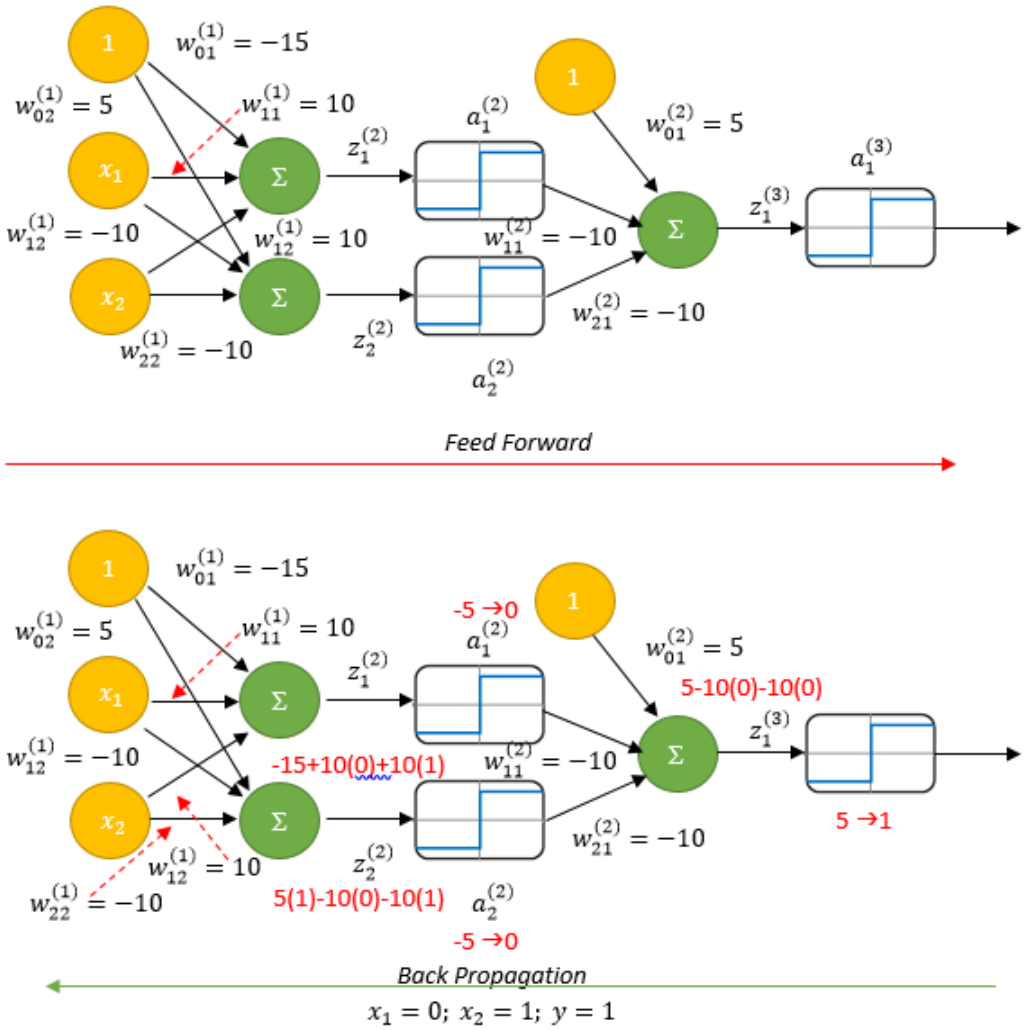

Figura 2.4: Esquema del funcionamiento algorítmico de los procesos de *Feed Forward* y *Back Propagation*, los conjuntos de pesos se denotan con un super índice. La neurona con valor unitario se le llama sesgo o bias, este se coloca en las redes neuronales para darle una mayor exactitud al modelo, sin embargo, a veces es omitido. La función de activación usada es el escalón unitario. Este MLP modela el comportamiento de una compuerta XOR.

2.4. Función de pérdida y función de activación

Según sea el tipo clasificación que se está realizando se denota la función de activación y la función de pérdida a ser utilizada en una arquitectura de una red neuronal, ver Tabla 2.1

Tipo de clasificación o modelo	Distribución Probabilística o Función de Activación	Función de Pérdida
Regresión lineal	Función Unitaria o Escalón	ECM Error cuadrático medio
Clasificación binaria	Sigmoide (Función Logística)	ECB Entropía cruzada Binaria
Clasificación Multiclase	Softmax (Función Logística Generalizada)	ECC Entropía Cruzada Categórica
Clasificación Multi-Etiquita	Conjunto de Sigmoides (Función Logística)	ECB Entropía Cruzada Binaria

Tabla 2.1: Se muestran las funciones de pérdida por modelo de clasificación o regresión, así como las funciones de distribución probabilística más comunes.

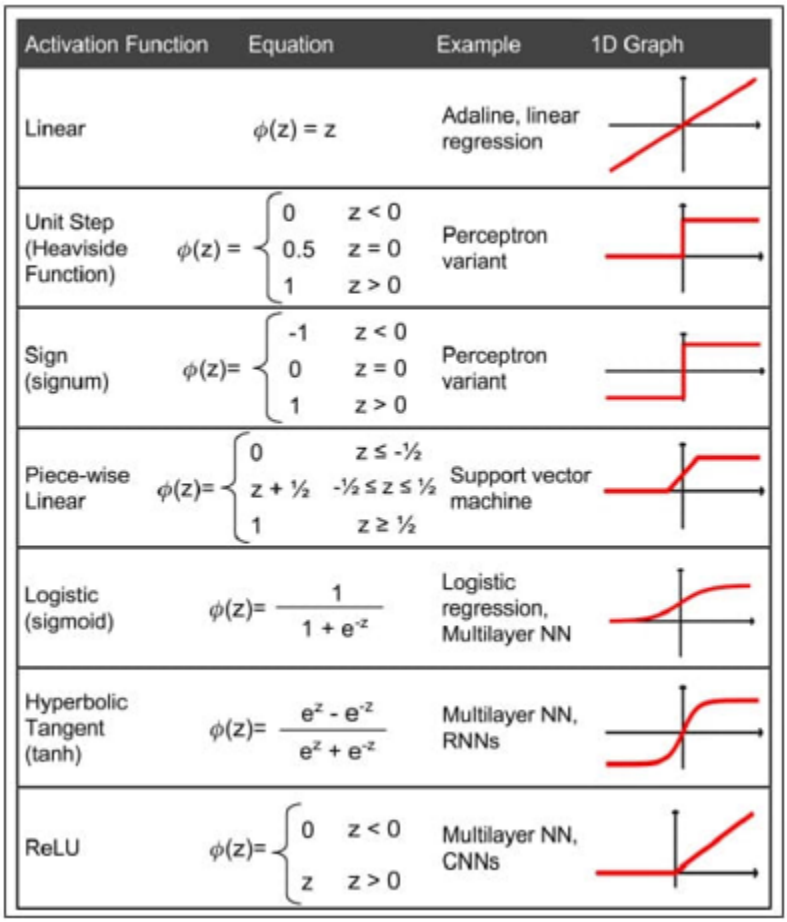

Tabla 2.2: Las funciones de activación más conocidas en redes neuronales, siendo la activación ReLU la más usada en CNNs. Imagen tomada de Sebastian Raschka 2016, portal web.

Un tema que va relacionado con las funciones de activación es la saturación [22], y esta ocurre justo cuando el algoritmo de entrenamiento se estanca en un mínimo local provocando que las funciones de activación trabajen en zonas de saturación, donde la pendiente se aproxima a cero, provocando que los ajustes a los pesos sean nulos o mínimos.

Funciones de activación como la ReLU se propusieron para llevar esos límites de zona de saturación al infinito y tratar el problema de ajustes de pesos de otras maneras.

En cuanto al tema de las funciones de pérdida, estas utilizadas para medir el error, entre las salidas deseadas contra la salida real que el modelo obtiene dada una entrada. Y se tiene la siguientes:

- Error cuadrático medio (ECM): es la función de pérdida más sencilla, ésta se modela con la siguiente ecuación:

$$E = \sum \frac{1}{2} (y - \hat{y})^2$$

  Se introduce un medio porque al ser una función monotónica no afecta en nada su comportamiento y simplifica las derivaciones. Existen otras variantes como son el error cuadrático medio logarítmico, el error absoluto medio, etc.

- Entropía cruzada binaria: Se toma considerando el negativo de la verosimilitud logarítmica

$$E = \frac{-1}{n} \sum_{i=1}^{n} \left[y^{(i)} log\hat{y}^{(i)} + (1 - y^{(i)}) log(1 - \hat{y}^{(i)}) \right]$$

- Entropía cruzada categórica:

$$E = -\sum_{j=1}^{m} \sum_{i=1}^{n} \left[y^{(i,j)} log\hat{y}^{(i,j)} \right]$$

2.5. Funciones de optimización

Al final lo que se busca resolver para ajustar los pesos involucrados en una red neuronal es minimizar la función de pérdida (función objetivo) mediante una función de optimización. Dicha función de optimización puede ser la definición de una función analítica, mediante un método numérico o métodos aproximativos, siendo la de métodos números de las más usadas.

- Gradient descent (GD): el algoritmo del descenso del gradiente es de tipo iterativo y es de uso popular en su versión SGD (*Stochastic Gradient Descent*) en las redes neuronales. Las condiciones fundamentales para aplicar este algoritmo es que la función tiene que ser derivable y definida. Queda expresado de la siguiente manera, sea el problema de optimización:

$$arg\ min_x f(x)$$

El vector en el punto x^t :

$$p^t = -\nabla f(x^t)$$

Sea la nueva notación:

$$\nabla f(x^t) = \nabla f^t$$

El proceso de actualización para el nuevo punto:

$$x^{t+1} = x^t - \mu \nabla f^t$$

Donde μ es el tamaño del paso, llamado también tasa de aprendizaje; se busca que sea lo suficiente pequeña para garantizar cuando $\nabla f^t = 0$:

$$f(x^t - \mu \nabla f^t) < f^t$$

Una visualización grafica de lo que está sucediendo al aplicar GD se puede observar en la Figura 2.5. La tasa de aprendizaje se vuelve un hiperparámetro más a ser ajustado, ya que un número grande o un número demasiado pequeño impacta en el comportamiento directo de la función de optimización.

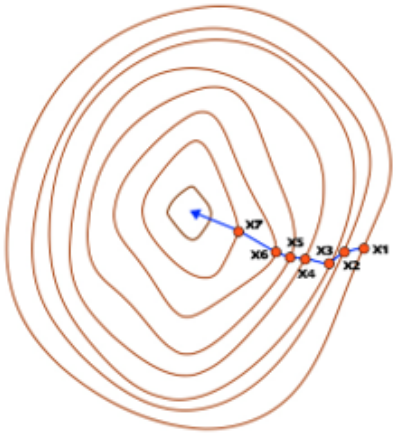

Figura 2.5: Descenso del gradiente hacia un mínimo de la función, los puntos en color naranja indican los pasos de actualización. Imagen tomada en *Towards Data Science*

- SGD con *Momentum*: El momentum ayuda acelerar el algoritmo de SGD en la dirección correcta a través de la adición de una fracción del vector de pesos.
- *Nesterov Accelerated Gradient* (NAG): mejora al momentum usando una media de actualizaciones previas, y para evitar bajadas a regiones de pendiente positiva, NAG se anticipa calculando estimaciones futuras de posiciones de los pesos.
- *Adaptive gradient* (AdaGrad): optimiza la tasa de aprendizaje del SGD, utiliza una tasa de aprendizaje para cada peso en un cierto instante de tiempo.

- *Adaptive Learning Rate Method* (ADEDELTA): una extensión de Adagrad busca evitar que la tasa de aprendizaje se haga infinitesimal, reduce la cantidad de gradientes acumulados a una ventana de cantidad fija.
- Adam: trabaja con dos medias, la media de gradientes cuadrados anteriores s_t y la media de los gradientes anteriores m_t , son ajustados y utilizados para actualizar los pesos mediante la regla de Adam [23].

  Una vez defino los procesos generales básicos de cualquier arquitectura de red neuronal, es necesario describir el algoritmo de aprendizaje, en este caso es el optimizador del gradiente descendente:

Algoritmo 1 Algoritmo general de aprendizaje de una red neuronal

1. Inicializar pesos
2. Para cada entrada
   a) Calcular la salida
   $\hat{y}_i(t) = f(W(t)^T x)$
   b) Actualizar los pesos mediante:
   $w_i(t+1) = w_i(t) + \mu g(t)$
3. Hasta que el error converja menor a un umbral

2.6. Proceso de convolución

Dentro de las CNNs existen otros procesos adicionales a los mencionados anteriores estos son:

- Proceso de convolución
- Proceso de submuestreo

La convolución en el caso continuo, se define de manera usual:

$$s(t) = \int x(a)w(t-a)da \rightarrow s(t) = (x * w)(t)$$

Siendo x es la señal de entrada, w el kernel, y producto de la convolución s , el mapa de características.

Para el caso discreto:

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

De manera que la entrada y el kernel son arreglos multidimensionales llamados tensores, en el caso de dos dimensiones:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) * K(i-m, j-n)$$

Pero las implementaciones en las CNNs son a través de la correlación cruzada, para optimizar la programación.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) * K(m,n)$$

La convolución como operación matemática dentro de una CNN funciona como una extracción de características local mediante los filtros (kernels) aplicada a una señal de entrada (imagen). Los filtros están conformados como una matriz de pesos que serán ajustables durante el BP.

Para definir cómo se hace la convolución en una imagen también se considera:

- *Padding* o relleno: es la adición de valores en los límites de la imagen para que el resultado de la convolución sea del mismo tamaño o bien para darle su debida importancia a los pixeles de los extremos de la imagen. Se hace agregando ceros (*Zero-Padding*) o repitiendo valores de los pixeles a las orillas.
- Stride o ventaneo: el stride es número de pixeles que se dejan como separación para recorrer el filtro tanto horizontal y vertical en el momento de la convolución.
- Tamaño de Filtro: tamaño de kernel, casi siempre se acostumbra a usar filtros de tamaño cuadrado, por ejemplo: 3x3, 5x5, 7x7, etc.
- Número y dimensión de filtros.

2.7. Capas de submuestreo

Las capas de decimación o submuestreo son implementadas en las CNNs con la intención de extraer la máxima respuesta o de máxima energía de un mapa de características. Existen las siguientes formas de hacer un submuestreo:

- *Max pooling* (Muestro máximo): extrae el valor máximo de una ventana de puntos, dado un stride y un valor de padding.
- *Average pooling* (Muestro promedio): obtiene el promedio de una vecindad de valores y la define como la respuesta local.
- *Sum pooling* (Muestro de suma): calcula las sumas del desplazamiento de una ventana en el mapa de características.

Esta son las tres maneras más populares para realizar un submuestreo, sin embargo, hay otras maneras más sofisticadas que ya se han estudiado [24].

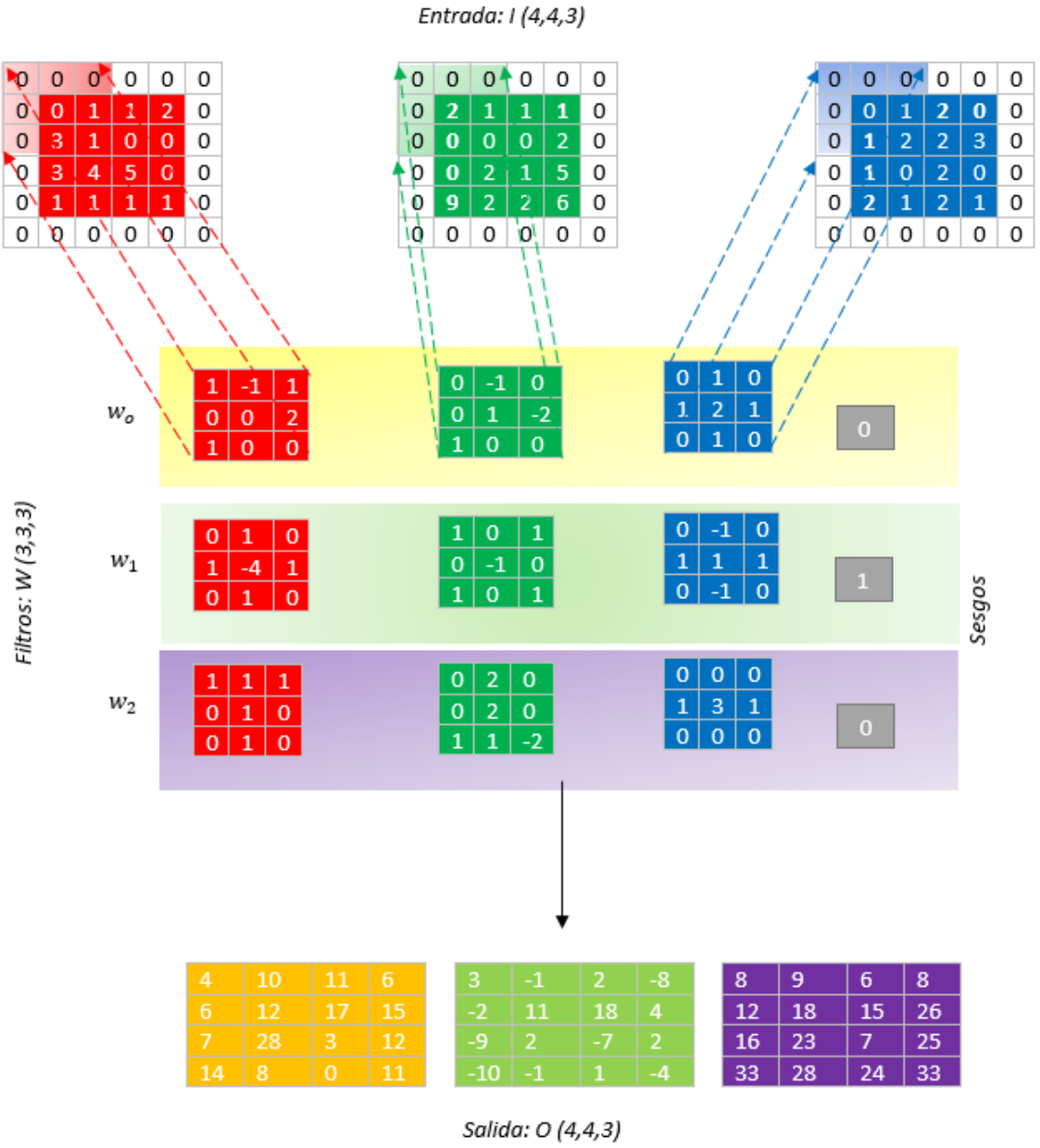

Figura 2.6: Proceso de convolución para una entrada (4,4,3), filtros de tres canales (3,3,3), Padding 1, stride de 1, dando como resultados los mapas de características de debajo de (4,4,3)

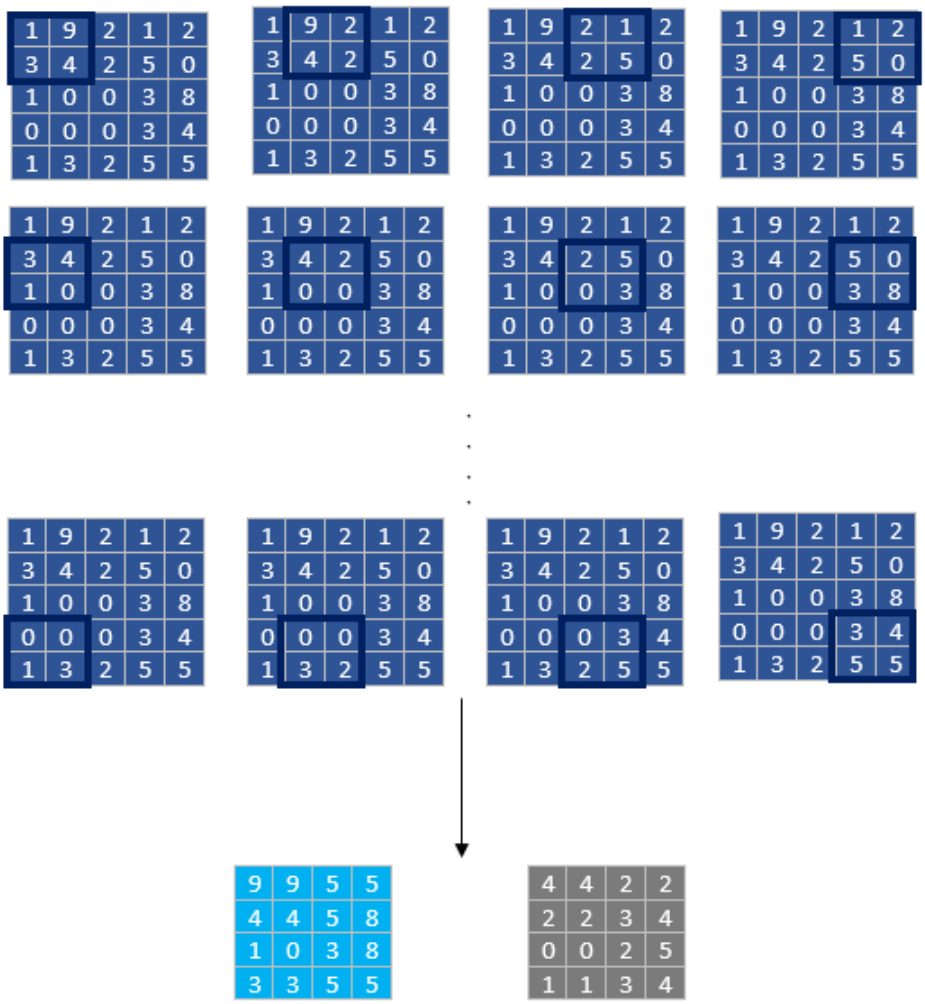

Figura 2.7: Proceso de submuestreo indicando el recorrido de la ventana (2,2) con un stride de 1, y mostrando los resultados en azul claro (*Max pooling*) y gris (*Avarage pooling*) [25]

2.8. Normalización por lote y *dropout*

Cuando se introducen los datos a una CNN es común aplicar un preprocesamiento con el objetivo de que el comportamiento de estos tenga una distribución normal, es decir, media cero y una varianza unitaria. Con eso se evita la saturación temprana de las funciones de activación.

A menudo dicho problema aparece en las capas intermedias debido al comportamiento del proceso de entrenamiento, volviendo lento el aprendizaje a la adaptación de nuevas distribuciones a esto se le conoce como: problema del desplazamiento covariable interno.

La manera de evitar el problema del desplazamiento covariable interno es aplicando la normalización por lote[26]. Para ello se obtiene la media μ_B y la varianza σ_B^2 del lote (batch),

para normalizar las entradas en esa capa mediante:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Y finalmente para obtener la salida:

$$y_i = \gamma \hat{x}_i + \beta$$

A esta última operación se conoce como escalado y traslado, los parámetros γ , β son aprendidos durante el entranamiento. La normalización por lote también podría considerarse como un tipo de regularizador.

Lo que intuitivamente se hace cuando se aplica la operación de normalización por lote, es mantener los pesos de una red neuronal a un rango de valores conocido, es decir evitar la variabilidad extrema de valores de los pesos.

Otro mecanismo para evitar el sobreajuste es la aplicación de una técnica de regularización conocida como *dropout*. Básicamente el *dropout* consiste en evitar la co-adaptación de neuronas a patrones repetitivos o estructuras complejas en los datos de entrada, y de esta manera garantizar una mejor generalización del modelo.

En el *dropout* se activan y se desactivan neuronas en las capas ocultas, a través de una malla de máscaras, aquellas neuronas inactivas no contribuyen en ninguna operación (tanto en FF y en BP) para la salida. Las neuronas que fueron apagadas en la época anterior son restauradas para la siguiente época de entranamiento. En el caso de prueba o validación no se aplica el *dropout*.

De cierta manera, el *dropout* crea una serie de topologías bajo una misma red.
El hiperparámetro para aplicar *dropout* es r, que es la cantidad de nodos al ser apagados de manera aleatoria y se expresa en una probabilidad.

2.9. Métricas de evaluación de modelos

Para evaluar el rendimiento de un algoritmo de aprendizaje hay varias métricas:

- Exactitud (*Accuracy*): es una métrica para evaluar modelos de clasificación, y se define como un cociente del número total de predicciones correctas sobre el total de predicciones. Se usa cuando el número de elementos de cada clase está bien balanceado.

$$exactitud = \frac{Numero\ de\ predicciones\ correctas}{Numero\ total\ de\ predicciones}$$

- Precisión: esta métrica evalúa el rendimiento de un modelo de clasificación con el cociente predicciones correctas sobre el total de las mismas más los falsos positivos. Y se

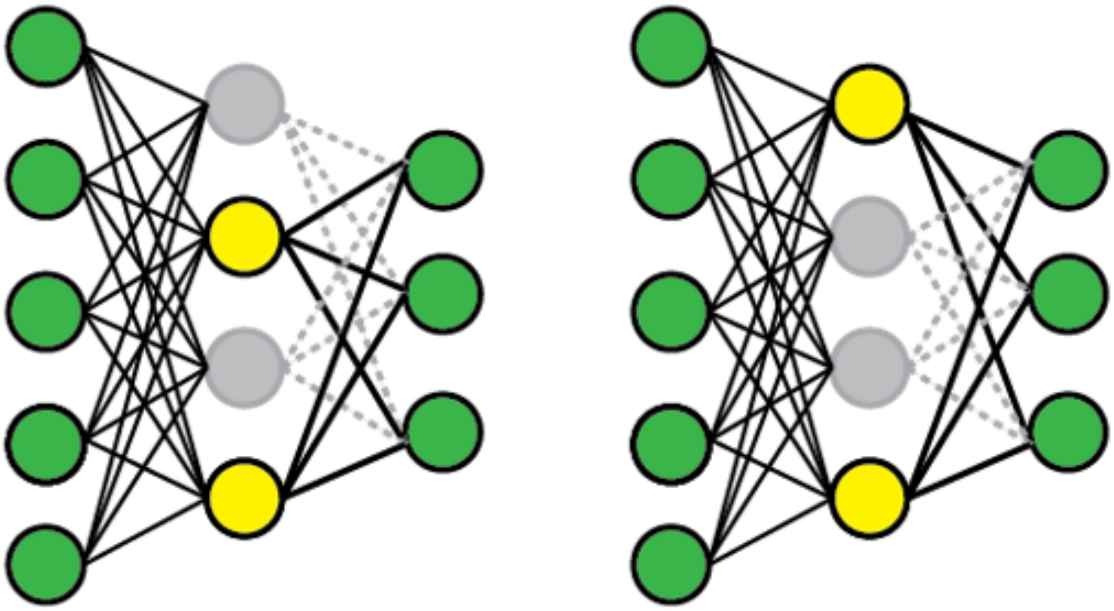

Figura 2.8: Este esquema representa el funcionamiento el *dropout*, los nodos en gris son las neuronas inactivas y los nodos en amarrillo las neuronas activas. Imagen tomada del foro: *Cross Validated* en Stack Exchange [26]

usa cuando el número de elementos de cada clase esta desbalanceado y existe una gran disparidad entre el número de etiquetas positivas y negativas en clasificación binaria. Contesta la pregunta: **¿Dada una predicción de clase en el modelo, que tan probable es que sea correcta?**

Para clasificación binaria:

$$Precisión = \frac{VP}{VP + FP}$$

Para clasificación multiclase se obtiene el promedio de la precisión:

$$Precisión = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{VP}{VP + FP}\right)$$

- Exhaustividad (*Recall*): también llamada tasa de verdaderos positivos, evalúa el rendimiento de un modelo de clasificación a través de un cociente entre las predicciones correctas sobre el total de las mismas, más los falsos negativos (también llamados elementos relevantes). Contesta la pregunta: **¿Dada una clase, que tan probable es que el modelo la detecte correctamente?**

  Para clasificación binaria:

$$exhaustividad = \frac{VP}{VP + FN}$$

Para clasificación multiclase:

$$exhaustividad = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{VP}{VP+FN}\right)$$

- Valor F1 (*F1 score*): es una métrica de uso común en clases desbalanceadas y combina los dos conceptos anteriores para evaluar el rendimiento de un modelo de clasificación. Matemáticamente hablando, el valor F1 es un promedio armónico entre las métricas de precisión y exhaustividad, y se calcula como:

$$valorF1 = 2 \times \frac{presicion \times exhaustividad}{presicion + exhaustividad} = 2 \times \frac{1}{\frac{1}{presicion} + \frac{1}{exhaustividad}}$$

Hay otras métricas de rendimiento según el tipo de clasificación y aplicación del algoritmo de aprendizaje.

Sea la matriz de confusión de una clasificación multiclase.

Entrada/Salida	A	B	C	D
A	9	1	0	0
B	1	15	3	1
C	5	0	24	1
D	0	4	1	15

Tabla 2.3: Matriz de confusión de un ejemplo de un modelo de clasificación para obtener sus métricas.

Los valores de la diagonal principal son **Verdaderos Positivos (VP)** es decir valores que el clasificador predijo de manera correcta. Los valores señalados en rojo son los **Verdaderos Negativos (VN)** para la clase A, en otras palabras, aquellos valores del modelo que, al recibir una entrada diferente de A, el modelo no predice una A. Los valores señalados en azul son **Falsos Positivos (FP)**, valores que el modelo predice una A cuando la entrada no fue una A. Los valores señalados en amarillo son **Falsos Negativos(FN)**, son los errores que más se quieren evitar cuando se trata de una clasificación binara, aquí la entrada es A pero el modelo predice otra cosa menos A.

Haciendo el cálculo de las métricas:

$$exactitud = \frac{9+15+24+15}{80} = \frac{62}{80} = 0{,}775$$

$$presicion = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{VP}{VP+FP}\right)_i = \frac{\frac{9}{9+6}+\frac{15}{15+5}+\frac{24}{24+4}+\frac{15}{7}}{4} = 0{,}7723$$

$$exhaustividad = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{VP}{VP+FN}\right)_i = \frac{\frac{9}{10}+\frac{15}{20}+\frac{24}{30}+\frac{15}{20}}{4} = 0{,}8$$

$$valor\ F1 = 2 \times \frac{1}{\frac{1}{0{,}7723}+\frac{1}{0{,}80}} = 0{,}7859$$

2.10. Problema del desvanecimiento del gradiente

The *Vanishing Gradient Problem* es inherente al uso de arquitecturas muy profundas con ciertas funciones de activación; debido a que los gradientes de las funciones de pérdida se aproximan a cero, dejando de ser óptimo el proceso de entranamiento para el aprendizaje.

Las funciones de activación influyen de manera crucial para que este fenómeno se presente, cuando estas al ser derivadas producen un número muy pequeño y cuando estas derivadas se van reciclando en el proceso de retro propagando hacia atrás (multiplicándose por la regla de la cadena), el cambio será demasiado pequeño para las capas iniciales, casi nulo.

Cuando los pesos de las primeras capas no se actualizan de manera correcta por el desvanecimiento del gradiente el aprendizaje será demasiado lento.

Una de las maneras efectivas de evitar este problema es el uso de otras funciones de activación

como la ReLU, que no produce derivadas pequeñas o la normalización por lote anteriormente vista.

2.10.1. *Residual Network* ResNet

Una de las arquitecturas que revolucionó las CNNs fue la arquitectura de bloques residuales porque los creadores de dicha arquitectura resolvieron de una manera práctica, otra manera de atacar el problema del desvanecimiento del gradiente.

Y la idea consiste en separar la red en pequeños bloques de dos o tres capas, para que al finalizar se realice una operación suma del mapa de características anterior del bloque. Existen un sin número de hipótesis de porque los bloques residuales mejoran el rendimiento de las CNNs, pero no existe un consenso generalizado.

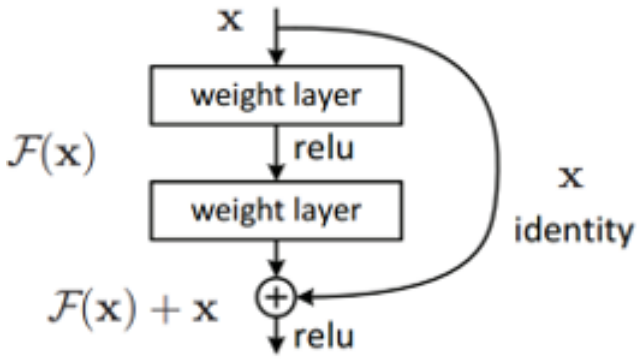

Figura 2.9: Esquema del bloque residual básico, como se puede observar a una entrada x se le aplica dos capas convolucionales $F(x)$ y luego se le suma al final de nuevo la entrada $F(x) + x$ para luego pasarlo por la activación ReLU, imagen tomada de He et al [27].

$$R(x) = H(x) - x$$

La operación residual busca la diferencia entre la distribución $H(x)$ y la entrada x , y como el objetivo de una CNNs es conocer la distribución, despejando:

$$H(x) = R(x) - x$$

Por eso se dice el bloque residual establece una conexión de identidad al sumar la entrada original a las convoluciones anteriores. De manera que la red residual de forma contraria a las arquitecturas tradicionales de CNN aprende la función residual $R(x)$, que es más sencilla que $H(x)$.

Las redes residuales también resuelven el problema del desvanecimiento del gradiente, dado que se puede propagar gradientes más grandes a las capas iniciales y están pueden aprender a la misma velocidad que las últimas capas. En la práctica, se agregan bloques de normalización por lote seguido de la activación ReLU antes de incorporar la entrada a una capa convolucional.

Bloques residuales

Generalmente hay dos topologías de bloques residuales, una básica y otra de embotellamiento (*bottleneck*)

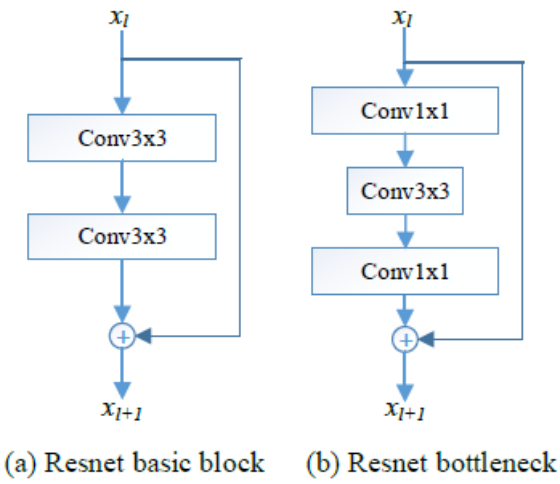

Figura 2.10: Topologías de bloques residuales, imagen tomada de He et al [27].

Para este trabajo se modifica el bloque básico para trabajar con las convoluciones Hermitianas y el bloque *bottleneck* no se ocupa porque los filtros de 1x1 no trasladan las propiedades de la transformada de Hermite.

Arquitectura

Los modelos de ResNet más comunes se muestran en la siguiente tabla:
Para dar un ejemplo sencillo, en la ResNet-34 (ver Figura 2.13), que inicia con la entrada de la imagen, a una primera convolución (conv1) y una capa de *Max pooling*, para luego dar un inicio a una serie de bloques definidas en la tabla anterior. En este caso tres bloques residuales de dos convoluciones (de 64 filtros de 3x3) seguido de otros dos bloques residuales de dos convoluciones (de 128 filtros de 3x3), así sucesivamente hasta terminar con una FC y realizar la clasificación pertinente.

Derivaciones

Las arquitecturas residuales trajeron consigo una serie de adaptaciones en los años posteriores a nuevas arquitecturas:

- *Densely Connected CNN*[28]: Traslada las conexiones directas a no sólo el siguiente bloque contiguo, si no a los demás bloques consecuentes, esto con la idea de trasladar las funciones residuales aprendidas a modo de que influyan en las predicciones hacia adelante.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Tabla 2.4: Repeticiones de cierto número de filtros y de ciertos tamaños a lo largo de la arquitectura residual, imagen tomada de He et al [27].

- Redes profundas con profundidad estocástica: Al incrementar el número de capas de las redes residuales, el tiempo de entrenamiento aumenta significativamente en [29] los autores proponen apagar y prender capas de manera aleatoria durante el entranamiento y el uso de todas las capas en prueba.
- Ensamblaje de redes más pequeñas con ResNet: de manera contraintuitiva en [30] se descubrió que es posible eliminar capas de una ResNet y tener los mismos resultados. Demostrando que una arquitectura ResNet con i bloques residuales tiene 2^i rutas diferentes. De manera que es posible eliminar capas que no comprometan su rendimiento.

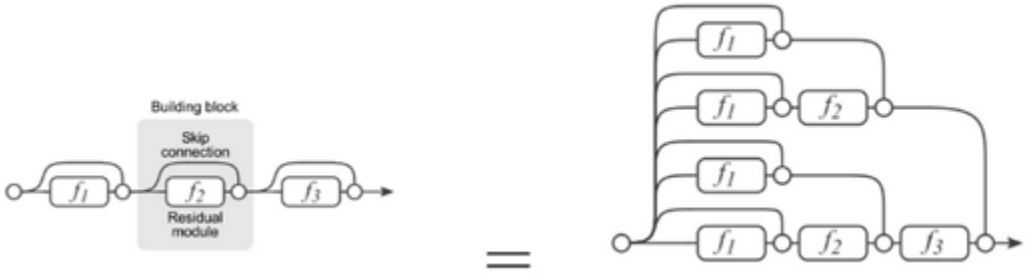

Figura 2.11: Esta imagen muestra el desglose de caminos de una serie de bloques residuales, imagen tomada de Veit et al [30].

- Bases de datos

  i MNIST: Es una base de datos de dígitos escritos con 60,000 imágenes de entrenamiento y 10,000 imágenes, y básicamente es la base de datos modelo a primera

instancia a ser utilizada para evaluar el desempeño de una arquitectura CNNs básica.

El tamaño de las imágenes son de 28 x 28 de un solo canal (escala de grises) y donde el dígito está centrado en una ventana de 20 x20, escrito por personas adultas y adolescentes, aproximadamente 250 escritores.

ii CIFAR 10: Este set de datos tiene 60,000 imágenes, de 32 x 32 en tres canales (RGB), con 10 clases (cada clase tiene 6,000 imágenes) (ver Figura 2.12).

iii CIFAR 100: al igual que la base anterior tiene 60,000 imágenes, pero con 100 clases contenidas en 20 superclases, algunas de ellas; mamíferos acuáticos, pescado, flores, frutas, insectos, arboles, vehículos, etc.

iv ImageNet: Contiene alrededor de 14 millones de imágenes clasificadas en 22 mil clases. El proyecto de recolección de imágenes fue iniciado y dirigido por Li Fei Fei, profesora de la Universidad de Stanford.

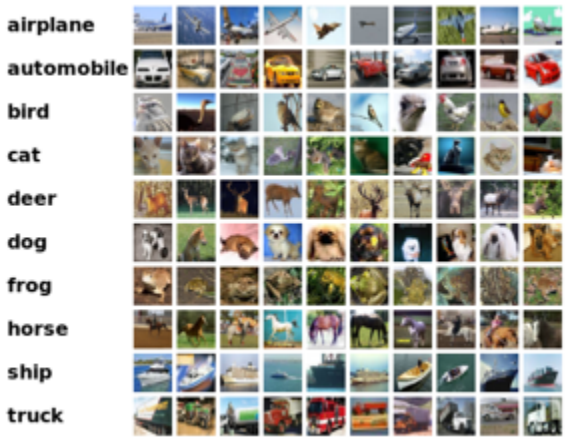

Figura 2.12: Clases de CIFAR-10, imagen tomada del sitio oficial de CIFAR (www.cs.toronto.edu)

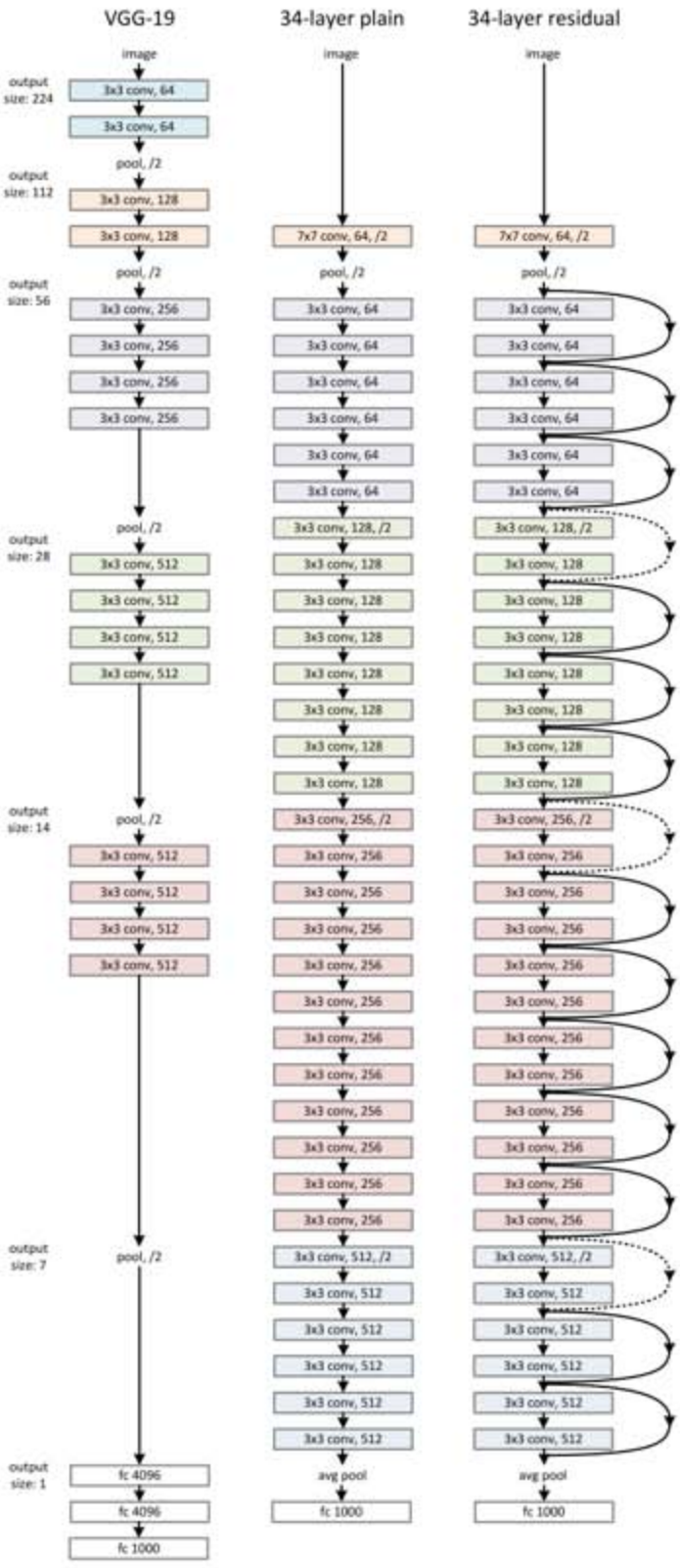

Figura 2.13: Arquitectura ResNet-34, comparada con una arquitectura de 34 capas sin bloques residuales y una arquitectura VG-19. Imagen tomada de He et al [27].

Capítulo 3

Transformada de Hermite

Computer science inverts the normal. In normal science, you're given a world, and your job is to find out the rules. In computer science, you give the computer the rules, and it creates the world

Alan Kay, American computer scientist

En [31] se demuestra que los campos receptivos del sistema de visión humano (HVS, Human Vision System) poseen una gran similitud a las derivadas de gaussianas y dada la relación de recurrencia que existe entre éstas y los polinomios de Hermite, se dice que la Transformada de Hermite se aproxima al HVS.

El trabajo teórico de los polinomios de Hermite fue desarrollado por Martens en 1990 [32] y extendió en [33][34]. En dichos trabajos se describe como la Transformada de Hermite en una herramienta útil para detectar estructuras fundamentales: bordes, esquinas, orientaciones, estimaciones de flujo, etc., y llevar dichos análisis a los dominios del espacio, frecuencia y escala.

3.1. Transformada polinomial

La transformada de Hermite se basa en la descomposición local de señales, la cual establece que una función $f(x)$ se puede aproximar mediante una combinación lineal de polinomios de orden n; las proyecciones de la función sobre la base funcional ϕ para cada n queda expresado como:

$$f_n = \frac{a_n}{c_n} = \frac{\langle f, \phi \rangle}{\langle \phi, \phi \rangle}$$

De manera que la base funcional ϕ sirve como base generadora para crear un espacio funcional normalizado dada una ventana de análisis $v(x - k\tau)$ con corrimiento . La condición de ortogonalidad es necesaria para utilizar un conjunto de funciones y poder definir la base funcional. En el caso de la Transformada de Hermite, la base funcional es generada a través de los polinomios de Hermite, que estos al ser ortogonales y su longitud igual a 1, entonces se obtiene una base funcional ortonormal y los coeficientes de f solo quedan definidos por a_n.

Sea entonces:

$$f_n(k\tau) = \int_R f(x)\phi_n(x - k\tau)v^2(x - k\tau)dx$$

Si los desplazamientos τ son equidistantes de las posiciones de ventana en los periodos de T, el producto de la ventana de análisis y la base funcional se puede expresar como:

$$D_n(x) = \phi_n(-x)v^2(-x)$$

A estas se les conoce como funciones de análisis $D_n(x)$

$$f_n(kT) = \int_R f(x)D_n(x - kT)dx$$

Una vez aplicada la transformada es posible obtener la transformada inversa a través de las funciones de síntesis $P_n(x)$

$$P_n(x) = \frac{\phi_n(x)v^2(x)}{\sum_k v(x - kT)} = \frac{\phi_n(x)v^2(x)}{W(x)}$$

También es necesario obtener la suma sobre todos los órdenes de n :

$$f(x) = \sum_{n=0}^{\infty} \sum_{k} f_n(kT) P_n(x - kT)$$

Cabe mencionar que existe una versión discreta de la HT a través del uso de filtros binomiales.

3.2. Transformada cartesiana

Se genera una base funcional a través de una expansión de polinomios de Hermite H_n producidos aplicando la fórmula de Rodrigues:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n e^{-x^2}}{dx^n}$$

Con una ventana gaussiana:

$$g(x) = \frac{1}{(\sqrt{\pi}\sigma)^{\frac{1}{2}}} e^{\frac{-x^2}{2\sigma^2}}$$

Los coeficientes de Hermite $L_n(x)$ se obtienen usando las funciones de análisis de Hermite (filtros):

$$D_n(x) = H_n(x) g^2(x) = \frac{(-1)^n}{\sqrt{2^n n!}} \frac{1}{\sigma\sqrt{\pi}} H_n\left(\frac{x}{\sigma}\right) e^{\frac{-x^2}{\sigma^2}}$$

$$L_n(x) = \int_x f(x) H_n(x - kT) g^2(x - kT) dx = \int_x f(x) D_n(x) dx$$

Debido a la propiedad de separabilidad espacial y la simetría rotacional de las funciones de análisis y síntesis, es directo obtener el caso bidimensional:

$$D_{n-m,m}(x, y) = D_{n-m}(x) D_m(y)$$

Donde $n - m$ denotan el orden de análisis en la dirección x y m en la dirección y.

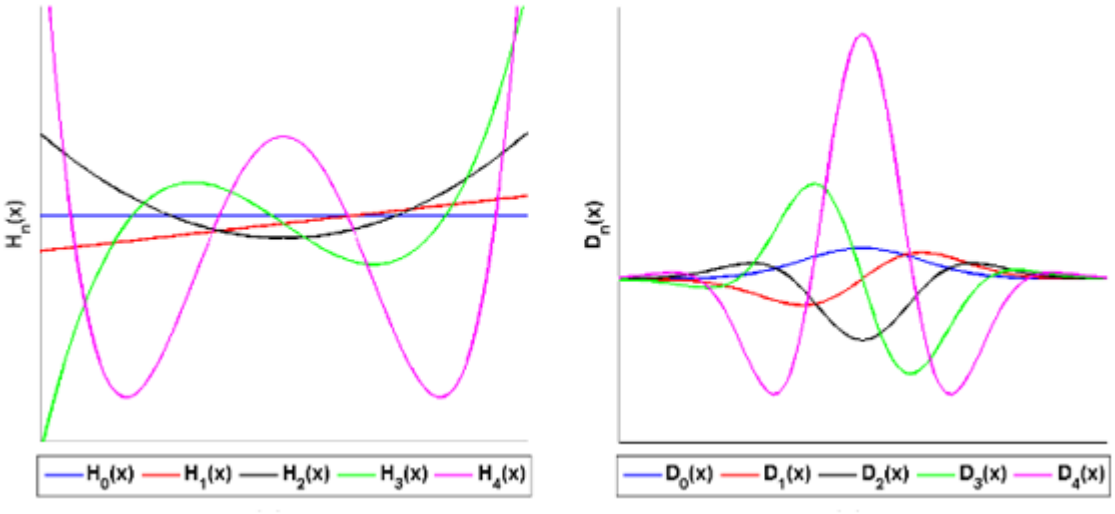

Figura 3.1: Del lado izquierdo se presentan los polinomios de Hermite H_n y del lado derecho las funciones de análisis D_n, ambas hasta cuarto orden.

Los coeficientes de Hermite para dos dimensiones se definen como:

$$L_{n-m,m}(x_0, y_0) = \int_x \int_y f(x, y) D_{n-m,m}(x - x_0, y - y_0) dx dy$$

Para $n = 0, 1, \cdots, \infty$ y $m = 0, 1, \cdots, n$

Figura 3.2: Funciones de análisis bidimensionales $D_{n-m,m}$, del lado derecho en el dominio de la frecuencia y del lado izquierdo en el dominio espacial. Imagen tomada de Olveres et al [5].

La ventana gaussiana se calcula mediante la propiedad de separabilidad:$g(x,y) = g(x)g(y)$ y por la propiedad isotrópica $\sigma = \sigma_x = \sigma_y$

$$g(x,y) = \frac{1}{2\sqrt{\pi}\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}}$$

Y los polinomios de Hermite:

$$H_{n-m,m}(x,y) = \frac{1}{\sqrt{2^n(n-m)!m!}} H_{n-m}\left(\frac{x}{\sigma}\right) H_{n-m,n}(x,y)\left(\frac{y}{\sigma}\right)$$

Hay otras variantes de la Transformada de Hermite que tienen otras utilidades como son:

- STH (*Steered Hermite Transform*): en la Transformada rotada de Hermite [34], se construyen filtros orientables a través de una combinación lineal de las funciones de análisis con respecto a una orientación θ y se describen como combinación lineal. Esta transformada es útil para encontrar patrones de hasta dos dimensiones o estructuras que tiene una orientación prominente localmente.

- Transformada multiresolución y multiescala: sirve para encontrar detalles finos dentro de una imagen [33] que puede estar en una escala de manera notoria, pero en otra escala casi imperceptible, lo que comúnmente se conoce como análisis multiescala y multiresolución.

  La transformada inversa y directa pueden esquematizarse como un proceso recurrente o esquema piramidal (ver Figura 3.3).

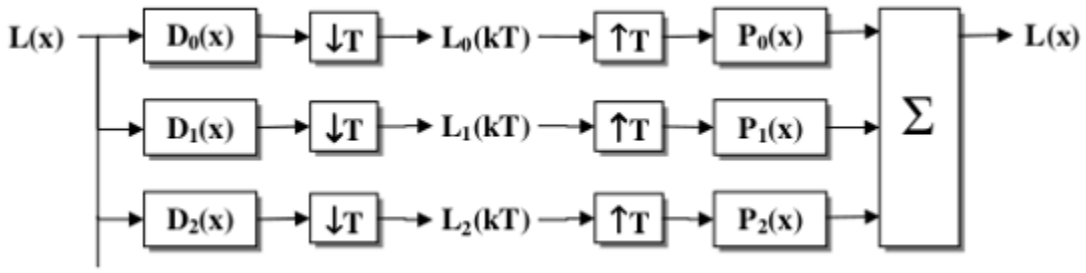

Figura 3.3: Transformada polinomial directa (izquierda) e inversa (derecha).

Capítulo 4

Redes Neuronales Hermitianas

A year spent in artificial intelligence is enough to make one believe in God.

Alan Perlis, American computer scientist,
first recipient of the Turing Award

Este capítulo está dedicado a explicar los procesos involucrados para utilizar filtros de Hermite en una CNN. Basado en [35] y replanteado para soportar los parámetros de los filtros de Hermite. Tomando como premisa de que la HT es una herramienta útil para extraer características de las imágenes y las ventajas ya mencionada en la introducción de esta tesis.

Hay dos procesos fundamentales en los que se centra esta nueva arquitectura:

- Proceso de convolución (modificado para soportar las características de los filtros de Hermite).
- Proceso de modulación.

Ambos procesos requieren de optimización a bajo nivel para la realización de las operaciones con tensores.

Antes de explicar los procesos es necesario definir los siguientes conceptos:

- Filtro aprendido (LF, *Learned Filter*): en una convolución tradicional, los filtros que se usan para la convolución son los mismos que se actualizan sus pesos a través del BP. Sin embargo, un filtro aprendido en este proyecto se limitará en aquellos filtros que sólo se actualizan sus pesos y son modulados.
- Filtro modulado: un filtro modulado es aquel que participa en el proceso de convolución sobre los mapas características, este concepto no existe en las arquitecturas CNN tradicionales. Los filtros modulados son recalculados duranta cada época de entrenamiento.
- Filtro de Hermite Cartesiano: un filtro tradicional obtenido a través de aplicar la transformada de Hermite (en el caso de esta tesis, la Transformada de Hermite Continua, no en su versión discretizada) a una función de entrada impulso, más adelante se explicarán con mayor detalle. Un conjunto de filtros de Hermite formará un banco de Filtros de Hermite (HB- *Hermite filter Bank*).
- Producto de Hadamard: también llamado producto de Schur, es una operación binaria elemento a elemento entre dos matrices o dos tensores de la misma dimensión y es diferente a una operación de multiplicación de matrices usual. El producto de Hadamard no sólo es el distributivo y asociativo como la multiplicación de matrices, si no también conmutativo.

Sean las matrices $\mathbf{A}, \mathbf{B}$, de tamaño $[m, n]$, el producto de Hadamard se define como:

$$(\mathbf{A} \circ \mathbf{B}) = (\mathbf{A})_{ij}(\mathbf{B})_{ij}$$

Ejemplo:

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \circ \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} \end{bmatrix}$$

4.1. Banco de filtros de Hermite

Para definir los filtros de Hermite a través de HT continua requiere de dos parámetros:

- Número de Orientaciones U: son las orientaciones que propagarán las propiedades de la HT a lo largo del proceso de entranamiento. El número de orientaciones y cuáles orientaciones; son hiperparámetros de diseño y asignación manual.
- Tamaño del filtro $h \times w$: por lo general $h = w$, para construir filtros de forma cuadrada, es común usar números primos; 1x1, 3x3, 5x5, 7x7, 11x11, etc.

Específicamente el tamaño de filtro va fuertemente relacionado a las funciones de análisis $D_{n-m,m}(x, y)$, mediante el cálculo de los polinomios de Hermite con orden n y desviación estándar σ.

El orden del polinomio n cumple la regla del tamaño del filtro h; $n \leq h - 1$, de forma que para obtener un filtro de 5x5 este se puede obtener con un polinomio de orden 4. Si el polinomio fuera de menor orden no se podría representar de manera de adecuada dicho filtro.

La desviación estándar sigue la regla $h = 2\sigma^2$, está desplaza, alarga y traslada la ventana gaussiana y cuando se define el orden de los filtros la escala se ajusta a través de este parámetro.

En el desarrollo de esta tesis, se mencionan la selección el banco de filtros de Hermite y la elección de parámetros.

4.2. Proceso de modulación

Una vez definido un banco de filtros de Hermite, se aplica un producto de Hadamard con éstos y los filtros aprendidos de cada capa de la CNN. Los filtros aprendidos tienen la dimensión O, I, U, h, w y los filtros de Hermite U, h, w , por lo que de manera intuitiva el producto de Hadamard se aplicará $O \times I$ veces sobre las dimensiones U, h, w de cada filtro, ver Figura 4.1

Las dimensiones $O \times I$ se refiere al número de canales de salida y de entrada respectivamente, para fines descriptivos se utilizará el concepto de grupo para los canales de salida y subgrupo para los canales de entrada.

A continuación, se describe el proceso de modulación sólo denotando las dimensiones tensoriales:

Sea el banco de filtros de Hermite con las dimensiones tensoriales $HB(U, h, w)$ y el conjunto de filtros aprendidos $LF(O, I, U, h, w)$, el producto de Hadamard queda definido como $LF(O, I, U, h, w) \circ HB(U, h, w) = HoFs(O \times U, I \times U, h, w)$:

Se define $U_n = U_1, \cdots, U_n$, entonces las dimensiones tensoriales quedan desglosadas como:

$$LF(O_1, I, U_n, h, w) \circ HB(U_1, h, w) = HoFs_{U_1}^{O_1}(O_1, I \times U_n, h, w)$$

$$LF(O_1, I, U_n, h, w) \circ HB(U_2, h, w) = HoFs_{U_2}^{O_2}(O_1, I \times U_n, h, w)$$

$$LF(O_1, I, U_n, h, w) \circ HB(U_3, h, w) = HoFs_{U_3}^{O_3}(O_1, I \times U_n, h, w)$$

$$\vdots$$

$$LF(O_1, I, U_n, h, w) \circ HB(U_n, h, w) = HoFs_{U_n}^{O_1}(O_1, I \times U_n, h, w)$$

$$\vdots$$

$$LF(O_n, I, U_n, h, w) \circ HB(U_1, h, w) = HoFs_{U_n}^{O_n}(O_n, I \times U_n, h, w)$$

Se realiza un apilamiento por U_n sobre cada canal de salida de la forma:

$$\begin{bmatrix} HOFs_{U_1}^{O_1}(O_1, I \times U_n, h, w) \\ HOFs_{U_2}^{O_2}(O_1, I \times U_n, h, w) \\ HOFs_{U_3}^{O_3}(O_1, I \times U_n, h, w) \\ \vdots \\ HOFs_{U_n}^{O_1}(O_1, I \times U_n, h, w) \end{bmatrix}$$

$$\vdots$$

$$\begin{bmatrix} HOFs_{U_1}^{O_n}(O_n, I \times U_n, h, w) \\ HOFs_{U_2}^{O_n}(O_n, I \times U_n, h, w) \\ HOFs_{U_3}^{O_n}(O_n, I \times U_n, h, w) \\ \vdots \\ HOFs_{U_n}^{O_n}(O_n, I \times U_n, h, w) \end{bmatrix}$$

Al resultado de aplicar el proceso de modulación se le llaman filtros de orientación de Hermite (HoFs) o bien filtros modulados de Hermite.

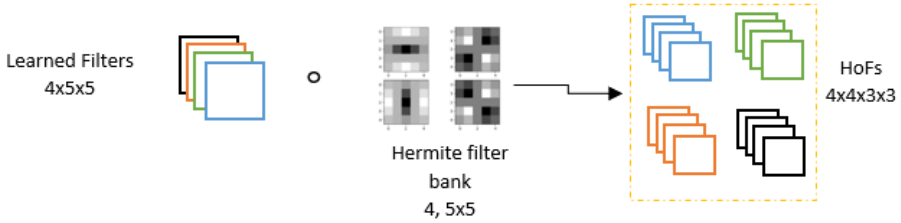

Figura 4.1: En el proceso de modulación se realiza un producto de Hadamard entre los filtros de Hermite y los filtros aprendidos para producir filtros orientados. Imagen inspirada y modificada de Chen et al [35].

Para ilustrar el proceso de modulación con un ejemplo, sea el $HB(5,5,4)$ es decir de cuatro direcciones y de tamaño 5x5 como el que se muestra en la Figura 4.1, y $LF(10,1,4,5,5)$ que se lee de 10 canales de salida (grupos g) , uno de entrada (subgrupos s), de cuatro orientaciones, de tamaño 5x5.

Renombrando, dado que $s = 1$ para el caso del ejemplo recorremos la aplicación del producto de Hadamard al nivel de $s = 4$

El proceso de modulación:

$$LF(g_1, s_1) \circ HF(g_1) = HoF(g_1, s_1)$$

$$LF(g_1, s_2) \circ HF(g_1) = HoF(g_1, s_2)$$
$$LF(g_1, s_3) \circ HF(g_1) = HoF(g_1, s_3)$$
$$LF(g_1, s_4) \circ HF(g_1) = HoF(g_1, s_4)$$

$$LF(g_1, s_1) \circ HF(g_2) = HoF(g_2, s_1)$$
$$LF(g_1, s_2) \circ HF(g_2) = HoF(g_2, s_2)$$
$$LF(g_1, s_3) \circ HF(g_2) = HoF(g_2, s_3)$$
$$LF(g_1, s_4) \circ HF(g_2) = HoF(g_2, s_4)$$

$$LF(g_1, s_1) \circ HF(g_3) = HoF(g_3, s_1)$$
$$LF(g_1, s_2) \circ HF(g_3) = HoF(g_3, s_2)$$
$$LF(g_1, s_3) \circ HF(g_3) = HoF(g_3, s_3)$$
$$LF(g_1, s_4) \circ HF(g_3) = HoF(g_3, s_4)$$

$$LF(g_1, s_1) \circ HF(g_4) = HoF(g_4, s_1)$$
$$LF(g_1, s_2) \circ HF(g_4) = HoF(g_4, s_2)$$
$$LF(g_1, s_3) \circ HF(g_4) = HoF(g_4, s_3)$$
$$LF(g_1, s_4) \circ HF(g_4) = HoF(g_4, s_4)$$

$$LF(g_2, s_1) \circ HF(g_1) = HoF(g_5, s_1)$$
$$LF(g_2, s_2) \circ HF(g_1) = HoF(g_5, s_2)$$
$$LF(g_2, s_3) \circ HF(g_1) = HoF(g_5, s_3)$$
$$LF(g_2, s_4) \circ HF(g_1) = HoF(g_5, s_4)$$

$$LF(g_1, s_1) \circ HF(g_2) = HoF(g_6, s_1)$$
$$LF(g_1, s_2) \circ HF(g_2) = HoF(g_6, s_2)$$
$$LF(g_1, s_3) \circ HF(g_2) = HoF(g_6, s_3)$$
$$LF(g_1, s_4) \circ HF(g_2) = HoF(g_6, s_4)$$

$$\vdots$$

$$LF(g_{10}, s_1) \circ HF(g_4) = HoF(g_{40}, s_1)$$
$$LF(g_{10}, s_2) \circ HF(g_4) = HoF(g_{40}, s_2)$$
$$LF(g_{10}, s_3) \circ HF(g_4) = HoF(g_{40}, s_3)$$
$$LF(g_{10}, s_4) \circ HF(g_4) = HoF(g_{40}, s_4)$$

La dimensión final de los filtros modulados: $HOFs(40, 4, 5, 5)$

La expresión matemática para el proceso de modulación es:

$$c^n_{i,n-m,n} = c_{i,0} \circ D_{n-m,m}$$

4.3. Proceso de convolución

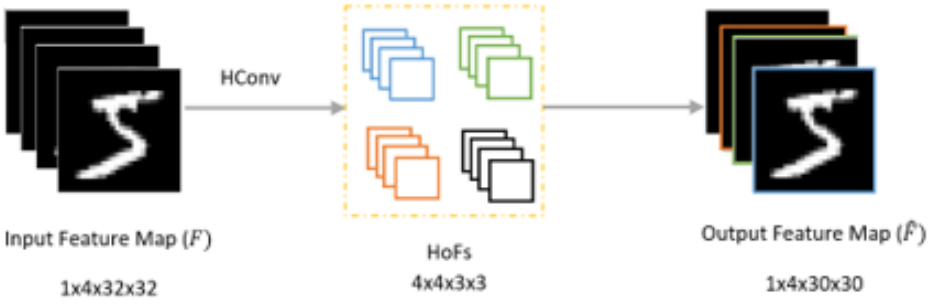

Figura 4.2: Proceso de convolución con filtros modulados HoFs y un mapa de características. Imagen inspirada y modificada de Chen et al [35].

A diferencia de la convolución tradicional, el proceso de convolución se modifica para obtener la respuesta a las U orientaciones. Esta realiza con los filtros de orientación de Hermite (HoFs) y los mapas de características de cada capa.

Dado los filtros modulados $HoFs(O \times U, I \times I, h, w)$ y un mapa de características

$F(SB, IC, H, W)$, donde SB es el tamaño del lote de entranamiento, IC canales de entrada, $H \times W$ el tamaño del mapa de características. En cuestión de dimensiones de tensores, la convolución se define como:

$$HConv = (HoFs(O \times U, I \times I, h, w), F(SB, IC, H, W)) = \hat{F}(SB, O \times U, H, W)$$

La convolución se aplica por orientación y se suma por canales (grupos):

$$HConv \rightarrow \hat{F}(SB_z, O_k, H, W) = \sum_{i=1}^{U_n} HoFs(O_k, I \times U_i, h, w) * F(SB_z, IC_i, H, W)$$

Por ejemplo, sea $F(128, 40, 28, 28)$ y $HoFs(80, 40, 5, 5)$,

$$\hat{F}(1, 1, 28, 28) = \sum_{i=1}^{U_n} HoFs(1, I \times U_i, h, w) * F(1, IC_i, H, W) =$$

$$[HoFs(1, 1, 5, 5) * F(1, 1, 28, 28)] + [HoFs(1, 2, 5, 5) * F(1, 2, 28, 28)] +$$
$$[HoFs(1, 3, 5, 5) * F(1, 3, 28, 28)] + [HoFs(1, 4, 5, 5) * F(1, 4, 28, 28)] +$$
$$[HoFs(1, 40, 5, 5) * F(1, 40, 28, 28)]$$

$$\hat{F}(1, 1, 28, 28) = \sum_{i=1}^{U_n} HoFs(2, I \times U_i, h, w) * F(2, IC_i, H, W) =$$

$$[HoFs(2, 1, 5, 5) * F(1, 1, 28, 28)] + [HoFs(2, 2, 5, 5) * F(1, 2, 28, 28)] +$$
$$[HoFs(2, 3, 5, 5) * F(1, 4, 28, 28)] + [HoFs(2, 4, 5, 5) * F(1, 4, 28, 28)] +$$
$$[HoFs(2, 40, 5, 5) * F(1, 40, 28, 28)]$$

$$\vdots$$

$$\hat{F}(1, 80, 28, 28) = \sum_{i=1}^{U_n} HoFs(80, I \times U_i, h, w) * F(80, IC_i, H, W) =$$

$$[HoFs(80, 1, 5, 5) * F(1, 1, 28, 28)] + [HoFs(80, 2, 5, 5) * F(1, 2, 28, 28)] +$$
$$[HoFs(80, 2, 5, 5) * F(1, 2, 28, 28)] + [HoFs(80, 4, 5, 5) * F(1, 4, 28, 28)] +$$
$$[HoFs(80, 40, 5, 5) * F(1, 40, 28, 28)]$$

$$\vdots$$

$$\hat{F}(128, 80, 28, 28) = \sum_{i=1}^{U_n} HoFs(80, I \times U_i, h, w) * F(128, IC_i, H, W) =$$

$$[HoFs(80, 1, 5, 5) * F(128, 1, 28, 28)] + [HoFs(80, 2, 5, 5) * F(128, 2, 28, 28)] +$$
$$[HoFs(80, 2, 5, 5) * F(128, 3, 28, 28)] + [HoFs(80, 4, 5, 5) * F(128, 4, 28, 28)] +$$
$$[HoFs(80, 40, 5, 5) * F(128, 40, 28, 28)]$$

La convolución también queda expresada de la siguiente manera, en términos de la HT (sólo se suprime el término del tamaño de batch):

$$\hat{F}_{i,n,n-m} = \sum_{j=1}^{U} F^{(j)} \otimes C^{j}_{i,n,n-m}$$

$$\hat{F} = HConv(F, C_i)$$

Esta ecuación es extraída de [35] y adaptada a los filtros de Hermite, sin embargo, se asume las operaciones entre grupos y subgrupos, y sólo se define la convolución para un solo canal.

En la Figura 4.3 se muestra cómo se conjunta el proceso de modulación y convolución; los filtros aprendidos (enmarcados en rojo) son modulados a través de la modulación de Hermite generando los filtros HoFs. Al final se observan los resultados de la convolución en los mapas de características, donde los bloques de cuatro HoFs corresponden a un mapa de características.

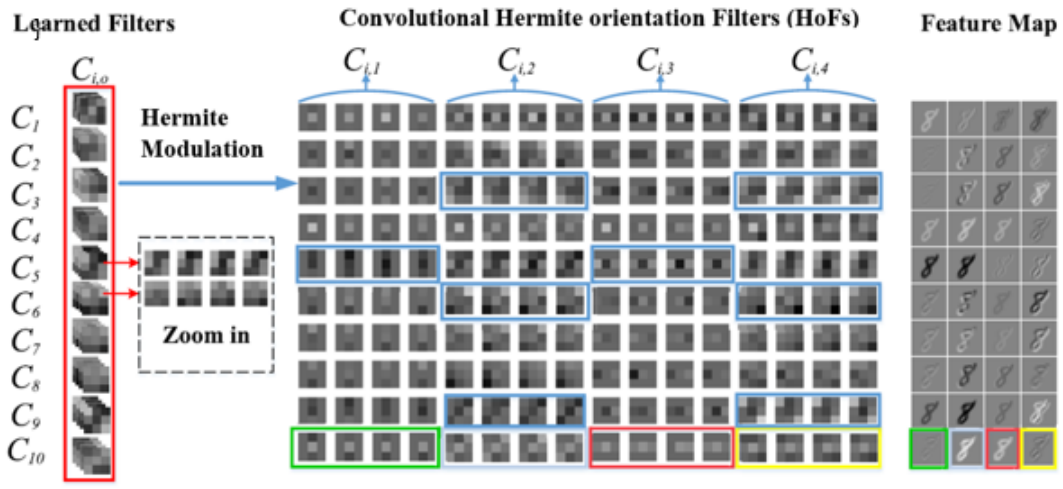

Figura 4.3: Esquema global de los procesos de modulación y convulsión de Hermite. Imagen inspirada y modificada de Chen et al [35].

Como es natural el algoritmo de BP también es ligeramente modificado y se calcula ahora como:

$$\delta = \frac{\partial L}{\partial C_{i,o}} = \sum_{u=1}^{U} \frac{\partial L}{\partial C_{i,u}} \cdot D_{n-m,m}$$

$$C_{i,o}^{t+1} = C_{i,o}^{t} - \eta\delta$$

4.4. Redes convolucionales con transformada de Gabor

La publicación en la cual está inspirado este trabajo [35], la transformada de Gabor es utilizada como extractor de características. La transformada de Gabor también es muy utilizada en el procesamiento de imágenes, y de manera general tiene dos parámetros de ajuste para la construcción de sus filtros: la escala y número de orientaciones.

El transformada de Gabor es un caso especial de Transformada de Fourier en Tiempo Corto cuya ventana es una gaussiana y surgió con la idea de facilitar la inversibilidad de la transformada de Fourier, así como la interpretabilidad en el espacio que se perdía en el dominio de la frecuencia.

Los filtros usados en ese trabajo se pueden ver en la Figura 4.4., la aportación principal fue proponer una nueva manera de involucrar los filtros de Gabor a una arquitectura CNN, bajo el principio de ayudar a la una red dándole conocimiento *a priori*.

Los autores proponen dos arquitecturas una básica para probar con imágenes de un solo canal y una ResNet para imágenes de tres canales. En esta tesis se replica la arquitectura

básica y se aplican varias pruebas para encontrar la de mejor rendimiento con los filtros de Hermite y se implementa una arquitectura ResNet variando el número de capas.

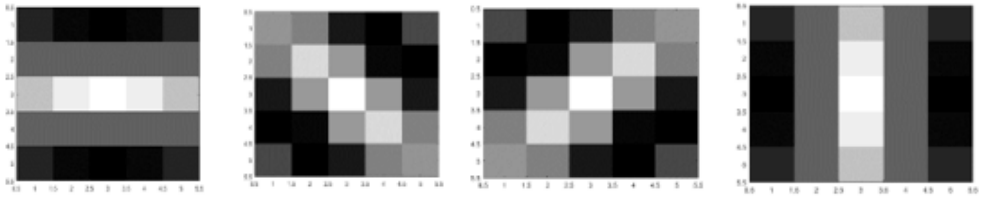

Figura 4.4: Filtros de Gabor utilizados en [35].

4.5. Arquitectura básica

La arquitectura básica tiene cuatro capas convolucionales, en cada una de ellas una normalización por lote y una activación ReLU, así como dos capas de *Max pooling* en las capas ocultas intermedias.

El numero de filtros es de 20,40,80,160 por capas y se hacen varias pruebas variando el tamaño de filtro y la escala.

Se expande la imagen de entrada a cuatro canales (se tetraplica tal cual), esto se hace con la intención de poder encontrar la respuesta a las cuatro orientaciones (son las que se aplicarán). Al final se conecta a una red MLP *Fully Connected* con una capa de *dropout* para clasificación.

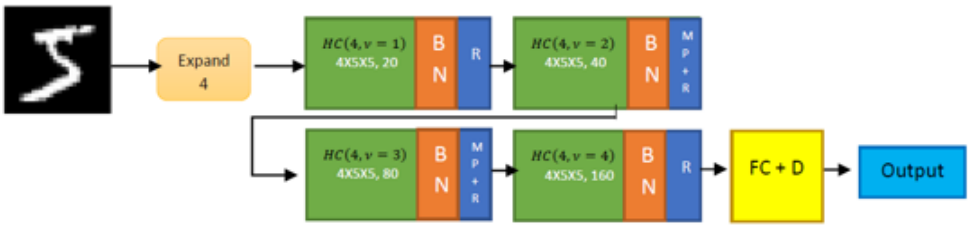

Figura 4.5: Arquitectura básica CNN, donde HC- Hermite Convolution, FC- *Fully Connected*, D- *dropout*, BN- *Bacth Normalization*, R- ReLU activation, MP- *Max Pooling*. Imagen inspirada y modificada de Chen et al [35].

Algoritmo 2 Red Neuronal Convolucional Hermitiana (HCN)

1. Se definen los parámetros y la arquitectura CNN.
2. Definir los filtros de Hermite.
3. Iniciamos el entrenamiento
4. **Hacer**
   Seleccionar un mini lote (*batch*) de entranamiento.
   Se producen los filtros modulados HoFs
   Se aplica el proceso de convolución
   Se calcula la función de pérdida y se aplica el BP.
   Se actualizan los filtros aprendidos.
5. **Hasta** un número máximo de épocas.

4.6. Arquitectura ResNet Hermite

La arquitectura ResNet utilizada se deriva en tres modelos:

- ResNet HCN3-28 con bloque de salida $Conv4\_X$, tres orientaciones.
- ResNet HCN4-40 con bloque de salida $Conv4\_X$, cuatro orientaciones.
- ResNet HCN4-40 con bloque de salida $Conv5\_X$, cuatro orientaciones

En número de filtros propuesto es 32, 64, 64, 128 por bloque residual y el tamaño de filtro también puede variar, pero el propuesto en esta tesis es de 5x5 porque presenta mejores resultados en la arquitectura básica.

CAPA	**RESNET HCN3-28**	**RESNET HCN4-40**
CONV1	$5x5, 32 * U$ $BN(32 * U)$	$5x5, 16 * U$ $BN(16 * U)$
$CONV2\_X$	$\begin{bmatrix} 5 \times 5 & 32 * U \\ 5 \times 5 & 32 * U \end{bmatrix} \times 2$	$\begin{bmatrix} 5 \times 5 & 16 * U \\ 5 \times 5 & 16 * U \end{bmatrix} \times 2$
$CONV3\_X$	$\begin{bmatrix} 5 \times 5 & 64 * U \\ 5 \times 5 & 64 * U \end{bmatrix} \times 4$	$\begin{bmatrix} 5 \times 5 & 32 * U \\ 5 \times 5 & 32 * U \end{bmatrix} \times 4$
$CONV4\_X$	$\begin{bmatrix} 5 \times 5 & 64 * U \\ 5 \times 5 & 64 * U \end{bmatrix} \times 5$	$\begin{bmatrix} 5 \times 5 & 32 * U \\ 5 \times 5 & 32 * U \end{bmatrix} \times 5$
$CONV5\_X$	$\begin{bmatrix} 5 \times 5 & 128 * U \\ 5 \times 5 & 128 * U \end{bmatrix} \times 2$	$\begin{bmatrix} 5 \times 5 & 64 * U \\ 5 \times 5 & 64 * U \end{bmatrix} \times 2$
MLP	128*U,10	64*U,10

Tabla 4.1: Arquitectura ResNet HCN implementadas y propuestas en este trabajo.

Capítulo 5

Desarrollo y resultados

Artificial Intelligence is the new electricity.

Andrew Ng, Chinese American computer scientist,
global leader in AI

5.1. Selección de filtros de Hermite

Los filtros seleccionados para probar las arquitecturas CNNs fueron de varios tamaños: 3x3, 5x5 y 7x7 (ver Figura 5.1), dichos filtros son determinados por el investigador.

Se escogieron cuatro orientaciones:

- Horizontal.
- Diagonal.
- Vertical.
- Diagonal invertida.

También se trabajó con cuatro escalas, tal y como se ve en la Figura 5.2

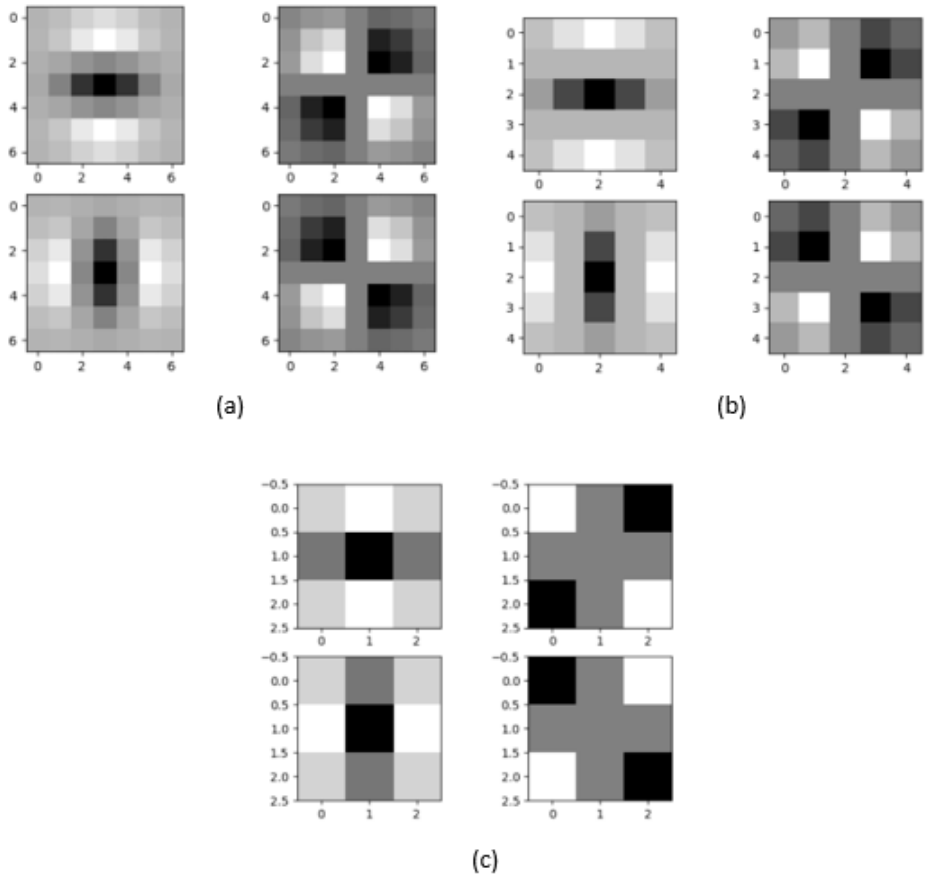

Figura 5.1: Filtros seleccionados (a) de 7x7 orden 6, (b) 5X5 orden 4, (c) 3x3 orden 2

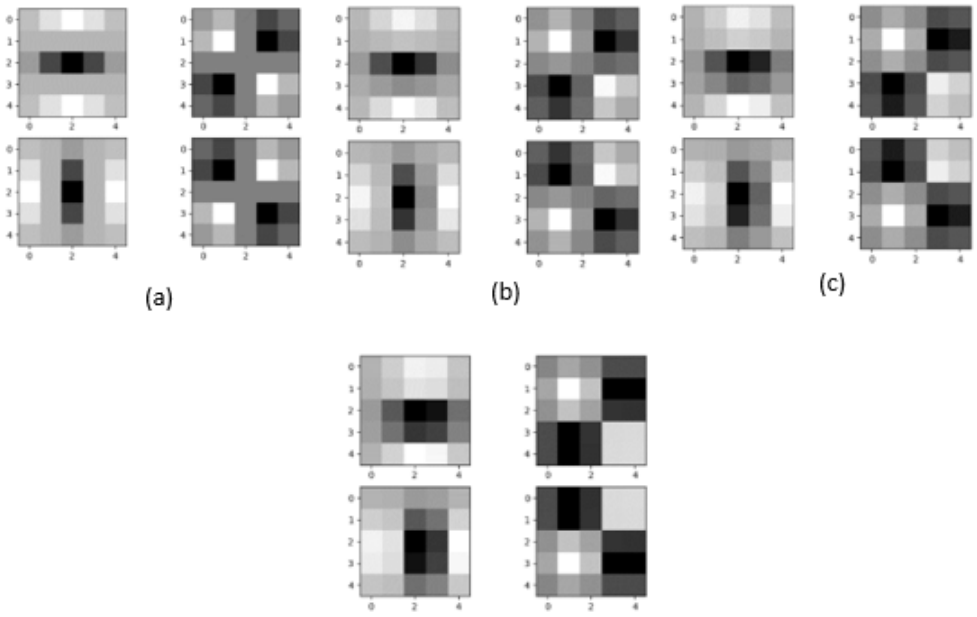

Figura 5.2: Filtros 5x5 a diferentes escalas: (a) escala 1, (b) escala 2, (c) escala 3 (d) escala 4

La manera de obtener los filtros cartesianos es bajo el principio de separabilidad que se explicó en el capitulo 3 de la Transformada de Hermite, se hizo bajo la matriz de orientaciones:

$$\begin{bmatrix} D_{0,2} & D_{1,1} \\ D_{2,0} & -D_{1,1} \end{bmatrix}$$

Donde las funciones de análisis de desglosan de la siguiente manera:

$$D_{0,2} = D_0(x)D_2(y)$$

$$D_{1,1} = D_1(x)D_1(y)$$

$$D_{2,0} = D_2(x)D_0(y)$$

$$-D_{1,1} = -D_1(x)D_1(y)$$

Éstas se determinan ingresando una función impulso a la entrada, para que de cierta manera, se pueda obtener el comportamiento propio de las funciones de análisis (no dependiente a una imagen de entrada en particular). En el caso de la escala se dividió el rango de operación del orden del polinomio; desde una mínima escala donde el orden de los filtros es idóneo para representar el tamaño del filtro deseado hasta la máxima escala donde es necesario un orden superior para no subrepresentar el orden con ese tamaño de filtro.

5.2. Ajustes de la arquitectura básica

Sobre la arquitectura original se hicieron ajustes al número de filtro y varias pruebas:

- Manteniendo una sola escala a lo largo el proceso de aprendizaje.
- Manteniendo un solo tamaño de filtro en todas las capas.
- Intercambiando el tamaño de filtros en cada época en orden ascendente.
- Ocupando cuatro diferentes escalas una diferente por cada capa convolucional.
- Variando las funciones de optimización:
  - SGD
  - Adam
  - ADEDELTA

Los parámetros de mejor rendimiento en esta arquitectura después son empleados en la arquitectura ResNet Hermite y aplicados a los diferentes modelos de la Tabla 4.1.

5.3. Pruebas y resultados

Las pruebas que se describen a continuación son las más relevantes durante el desarrollo de este proyecto.

5.3.1. Arquitectura básica

Se esquematiza el proceso de la arquitectura básica en la Figura 5.3, pero el proceso general en dimensiones tensoriales, con un tamaño de batch de 128, con filtros de Hermite de 5x5 de cuatro orientaciones, sería el siguiente:

$$LF_{C1}(10, 1, 4, 5, 5) \circ HB(4, 5, 5) = HoFs_{C1}(40, 4, 5, 5)$$

$$I(128, 1, 28, 28) * HoFs_{C1}(40, 4, 5, 5) = F_{C1}(128, 40, 28, 28)$$

$$LF_{C2}(20, 10, 4, 5, 5) \circ HB(4, 5, 5) = HoFs_{C2}(80, 40, 5, 5)$$

$$F_{C1}(128, 40, 28, 28) * HoFs_{C2}(80, 40, 5, 5) = F_{C2}(128, 80, 28, 28)$$

$$MaxPooling(F_{C2}(128, 80, 28, 28)) = F_{C2}(128, 80, 14, 14)$$

$$LF_{C3}(40, 20, 4, 5, 5) \circ HB(4, 5, 5) = HoFs_{C3}(160, 80, 5, 5)$$

$$F_{C2}(128, 80, 14, 14) * HoFs_{C3}(160, 80, 5, 5) = F_{C3}(128, 160, 10, 10)$$

$$MaxPooling(F_{C3}(128, 160, 10, 10)) = F_{C3}(128, 160, 5, 5)$$

$$LF_{C3}(80, 40, 4, 5, 5) \circ HB(4, 5, 5) = HoFs_{C4}(320, 160, 5, 5)$$

$$F_{C3}(128, 160, 5, 5) * HoFs_{C2}(320, 160, 5, 5) = F_{C4}(128, 320, 1, 1)$$

$$F_{C4}(128, 320, 1, 1) \rightarrow FC(F_{C4}(128, 320, 1, 1), 1024, 10)$$

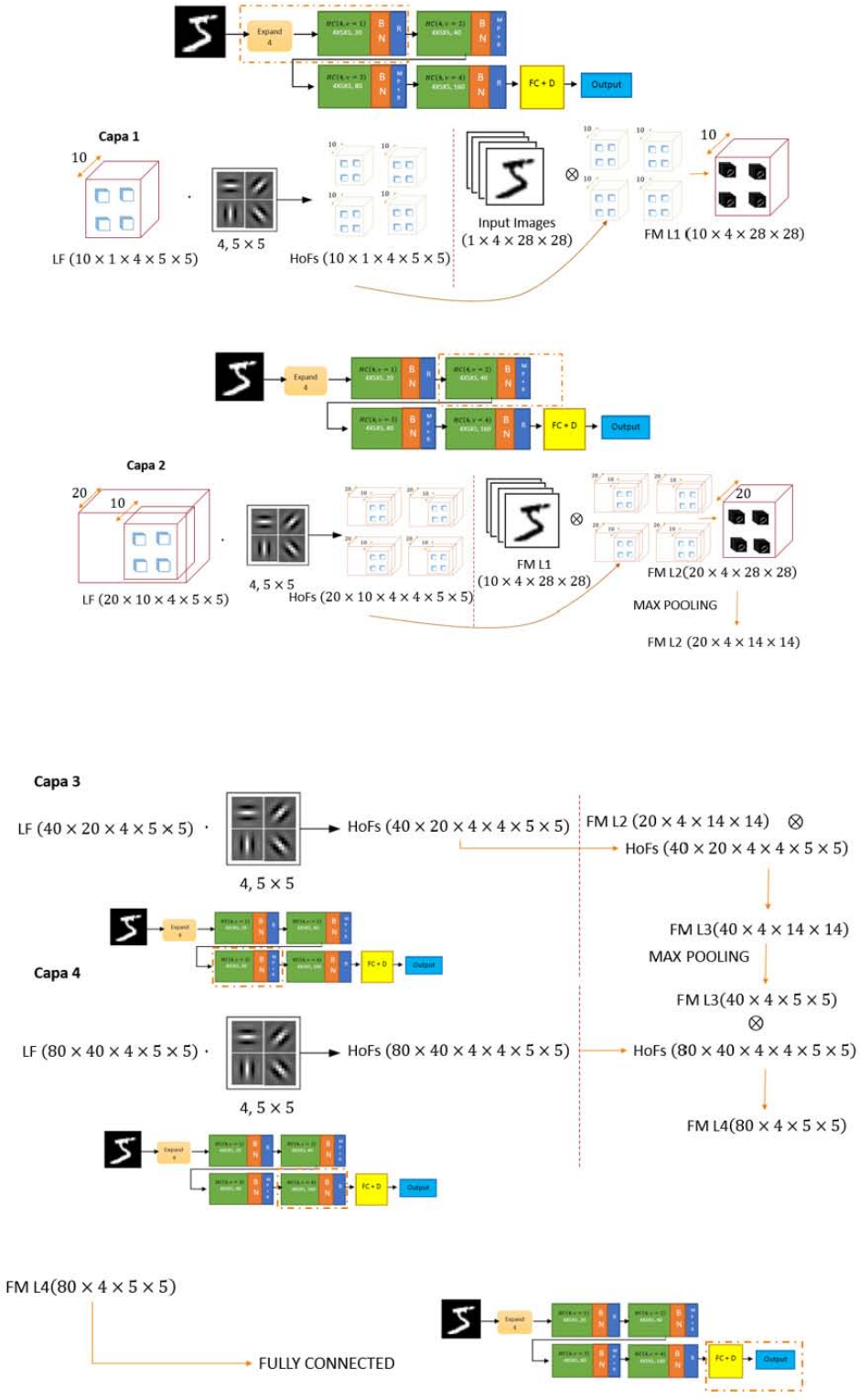

Figura 5.3: Esquema de la arquitectura básica capa por capa convolutiva.

Los resultados obtenidos se muestran según la base de datos que fue probada, las pruebas Fueron hechas en una tarjeta de video NVIDIA 1080.

5.3.2. MNIST

Evaluando con diferentes optimizadores y variando escalas con filtros de 5x5 (Tabla 5.1), variando el tamaño de los filtros (Tabla 5.2). Comparando contra los modelos del estado del arte (Tabla 5.3).

%Accuracy / %error	Optimizador
99.51 / 0.49	SGD con una escala
99.46 / 0.54	SGD con cuatro escalas
99.42 / 0.58	Adam con cuatro escalas
99.34 / 0.66	ADEDELTA con cuatro escalas

Tabla 5.1: Rendimiento de la HCN arquitectura básica en MNIST con diferentes optimizadores y cambiando escalas.

%Accuracy / %error	Tiempo	Tamaño de Filtros
99.51 / 0.49	9 min 30s	SGD (5,5,5,5)
99.39 / 0.61	9 min 10s	SGD (3,3,3,3)
99.31 / 0.58	25 min 9s	SGD (7,7,7,7)
99.27 / 0.66	12 min 10s	SGD (3,3,5,5)

Tabla 5.2: Rendimiento de la HCN arquitectura básica en MNIST variando el tamaño de filtros

%error	Estructura y kerneles usados	Arquitectura	Año
0.18	**Combina n arquitecturas CNN, RNN, etc. (Para MNIST 30 RDLs)**	**RMDL: Random Multimodel Deep Learning for Classification[36]**	**2018**
0.21	5 CNN/ 6-layer: 784-50-100-500-1000-10-10	Regularization of Neural Networks using DropConnect [37]	2013
0.23	35 CNNs, 1-20-P-40-P-150-10	Multi-column Deep Neural Networks for Image Classification [38]	2012
0.23	6-layer 784-50-100-500-1000-10-10	APAC: Augmented PAttern Classification with Neural Network [39]	2015
0.57	10-20-40-80	ORF4 (ORAlign) [10]	2017
0.48	20-40-80-160	GCN4 (7X7) [35]	2018
0.49	10-20-40-80	HCN4 (5X5)	2019

Tabla 5.3: El estado del arte de MNIST

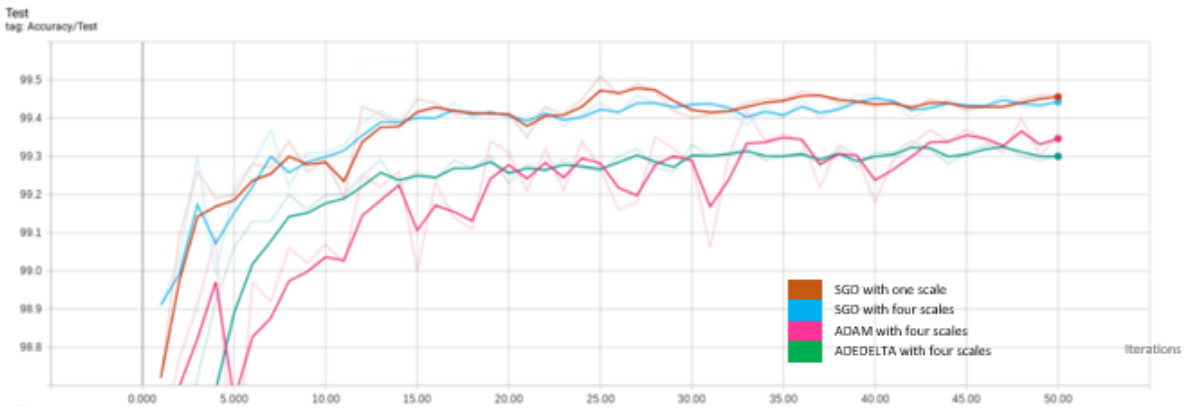

Figura 5.4: Comportamiento de los diferentes optimizadores de la Tabla 5.1

5.3.3. Fashion MNIST: una derivación de MNIST, pero con prendas de vestir

5.3.4. eMNIST: MNIST de letras del alfabeto

Como se puede observar el mejor rendimiento obtenido para MNIST es utilizando el optimizador SGD manteniendo una sola escala, cuando se trata de cosas con mayor detalle como Fashion MNIST es bueno aplicar el incremento del tamaño del filtro en forma ascendente.

Otra discusión que se puede observar a partir de la comparativa con los modelos del estado

%ACC/ %error	Tiempo	Optimizador
97.09/2.91	10 m 12s	SGD (5,5,5,5)
95.01 / 4.99	10 m 2s	SGD (3,3,3,3)
90.13/9.87	30 m 2s	SGD (7,7,7,7)
97.19/2.81	**14 min 14s**	**SGD (3,3,5,5)**

Tabla 5.4: Resultados con Fashion MNIST

%ACC/ %error	Tiempo	Optimizador
99.67 / 0.33	**9 m 50s**	**SGD (5,5,5,5)**
99.60 / 0.40	8 m 55s	SGD (3,3,3,3)
99.41 /0.59	25 min 29s	SGD (7,7,7,7)
99.46 /0.54	10 min 14s	SGD (3,3,5,5)

Tabla 5.5: Resultados con eMNIST

del arte, es que a pesar de que no se supera la exactitud, las arquitecturas empleadas son más profundas y requieren un equipo de cómputo superior, en cambio con una red HCN relativamente sencilla se puede tener resultados próximos y con una capacidad de cómputo menor.

5.3.5. Arquitectura RESNET-HCN

Se implementó las 3 vertientes propuestas de la RESNET-HCN en PyTorch. Para darle un mejor rendimiento se manejaban tazas de aprendizaje diferentes por cada cierto número de épocas.

CIFAR-10 y CIFAR-100

En cuestión de tiempos con el GPU a disposición la arquitectura ResNet tardó en correr alrededor de 9 a 10 horas, pero cercano a las 4.5 horas ya no había un cambio sustancial en el rendimiento y el modelo se podría parar ahí, ver Tabla 5.6.

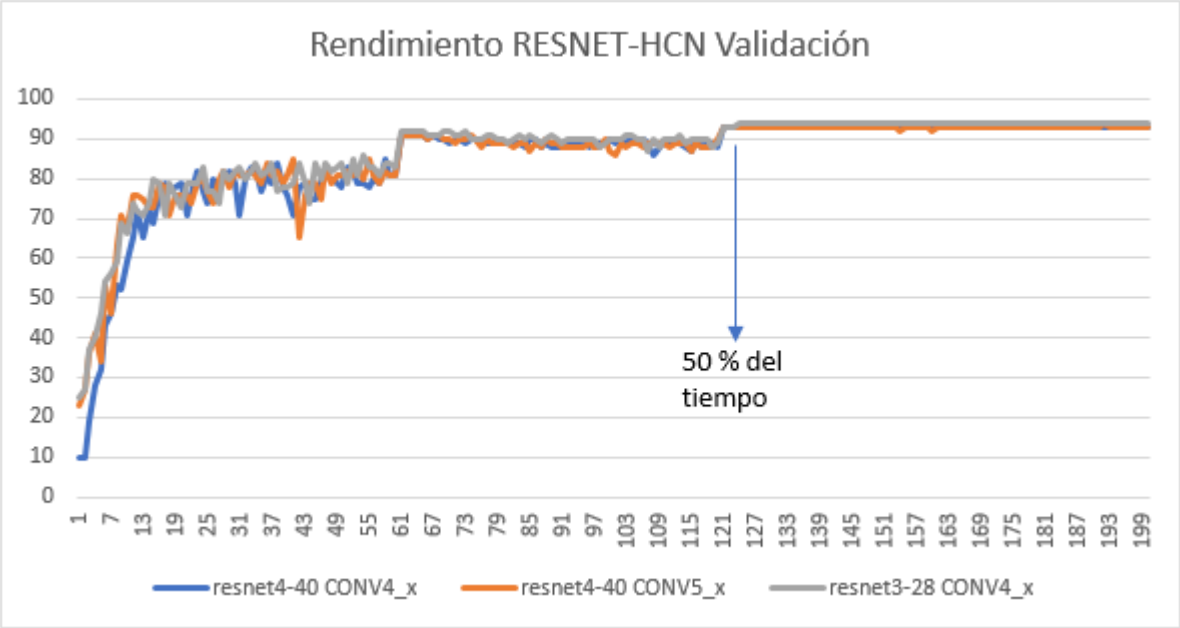

Figura 5.5: Rendimiento de la arquitectura ResNet HCN con CIFAR 10. El eje de las x es el número de épocas de entrenamiento y el eje de las ordenadas la exactitud del modelo para datos de validación.

Se puede observar que la ResNet HCN alcanzó resultados muy similares a los de Gabor con 6 veces menos el número de parámetros entrenables. A modo de comparación se aplicó la misma arquitectura para filtros de Gabor (ya que se desconoce la arquitectura exacta de bloques residuales y tamaños de filtros del mejor modelo de [35]).

Se puede concluir que la ResNet HCN3-28 para bases de datos más complejas (a diferencia de MNIST) aporta más en su conjunto que la ResNet4-40, ya que obtiene mejores resultados.

Método/Arquitectura			**%error CIFAR-10**	**%error CIFAR-100**
VGG			6.32	28.49
	Filtros por etapas	#params		
RESNET-110	16-16-32-64	1.7 M	6.43	25.16
RESNET-1202	16-16-32-64	10.2 M	7.83	27.82
GCN2-110	12-12-24-45	1.7 M	6.34	
GCN4-110	8-8-16-32	6.5 M	4.96	
GCN2-40	16-32-64-128	4.5 M	4.95	24.23
GCN4-40	16-16-32-64	2.2 M	5.34	25.65
WRN-40	64-64-128-256	8.9 M	4.53	21.18
WRN-28	160-160-320-640	36.5 M	4.00	19.25
GCN4-40	16-32-64-128	8.9 M	4.65	20.73
GCN3-28	64-64-128-256	17.6 M	3.88	20.13
HCN4-40_CONV4_X	**16-32-32-64**	**1.7 M**	**5.89**	**22.05**
HCN4-40_CONV5_X	**16-32-32-64**	**2.8 M**	**5.83**	**22.20**
HCN3-28_CONV4_X	**32-64-64-128**	**3.5 M**	**5.03**	**20.36**
GCN3-28_CONV4_X	**32-64-64-128**	**3.5 M**	**6.0**	**21.68**
STATE OF ART [2018] **GPIPE**	**GIANT DEEP LEARNING MODEL PIPELINE** [8 *GPU*]	**557 M**	**1.00**	**8.7**

Tabla 5.6: Resultados con CIFAR-10

Capítulo 6

Conclusiones

Analizando los resultados se puede observar un rendimiento similar obtenido al utilizar los filtros de Gabor. En esta tesis se demuestra que:

- Con una red básica de cuatro capas convolucionales es posible aproximarse a los niveles de exactitud del estado del arte, cuando se está trabajando con imágenes de un canal.
- La reducción de número de parámetros se logra en correspondencia con la tarea y arquitectura que se está resolviendo.
- En medición de tiempos se comporta de manera adecuada con el tipo de hardware que se dispone.

Y se cumplen con los objetivos planteados al inicio de esta tesis:

- Una primera arquitectura en el mundo de las CNN que involucra filtros de Hermite.
- Una integración fácil a arquitecturas ya conocidas como la ResNet con filtros de Hermite.
- Comparación de resultados en diferentes bases de datos contra los estados del arte en clasificación.

6.1. Trabajos a futuro

Aún quedan varios temas por resolver:

- Estudiar y probar el comportamiento de otros filtros de Hermite para la mejora del rendimiento, o incluso agregar otros filtros detectores de mayor orden.
- Aplicar la transformada rotada de Hermite para recuperar otras características de las imágenes.
- Probar con bases de datos más complejas y extensas como el caso de ImageNet.
- Encontrar y resolver una aplicación médica de clasificación para analizar el comportamiento de este tipo de arquitectura.
- Extender el proceso de modulación y convolución modificado a nuevas arquitecturas.
- Aplicar las HCNs a otro tipo de tareas, no sólo clasificación, si no detección, reconocimiento de patrones.

Bibliografía

[1] A. Krizhevsky, I. Sutskever, and G.E. Hinton: "ImageNet classification with deep convolutional neuronal networks". In: *Proc. Adv. Neural Inf. Process. Syst.*, Vol 1, pp. 1097-1105. Nevada (2012).

[2] O. Russakovsky et al: "ImageNet Large Scale Visual Recognition Challenge". *International Journal of Computer Vision, Vol. 115*, Issue 3, pp. 211-252, (2015).

[3] R.E. Soodak: "Two-dimensional modeling of visual receptive fields using Gaussian subunits". *Proc. Natl. Acad. Sci. USA*, Vol. 83, Issue 23, pp. 9256-9263, (1986).

[4] D.L. Ringach: "Mapping receptive fields in primary visual cortex". *J Physiol.*, Vol. 558, Issue 3, pp. 717-728, (2004).

[5] J. Olveres, R. Nava, B. Escalante, E. Vallejo and J. Kybic: "Left ventricle Hermite-based segmentation". *Computers in Biology and Medicine*, Vol. 87, pp. 236-249, (2017).

[6] E. Carbajal, R. Nava, J. Olveres, B. Escalante and J. Kybic: "Texel-based image classification with orthogonal bases". In: *Optics, Photonics and Digital Technologies for Imaging Applications IV*, Vol. 9896. Brussels(2016)

[7] J. Olveres, E. Carbajal, B. Escalante, E. Vallejo and C.M. García: "Deformable Models for Segmentation Based on Local Analysis". *Mathematical Problems in Engineering*. Vol. 2017. (2017).

[8] https://paperswithcode.com/sota/image-classification-on-imagenet

[9] A. Calderón, S. Roa, and J. Victorino: "Handwritten digit recognition using convolutional neuronal networks and Gabor filters". In: *Proceedings of the International Congress on Computational Intelligence CIIC 2003*, Medellin (2003).

[10] Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao: "Oriented response networks." In: *30th IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2017, Volume 2017, pp. 4961-4970. IEEE, Honolulu (2017).

[11] J. Bruna, and S. Mallat: "Invariant scattering convolution networks". *IEEE Trans. Pattern Anal. Mach. Intell*, 45(8), 1872-1886 (2013).

[12] L. Sifra, and S. Mallat: "Rotation, scaling and deformation invariant scattering for texture discrimination." In: 26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2013, pp. 1233-1240. IEEE. Portland (2013)

[13] S. -Y. Chang and N. Morgan: "Robust CNN-based speech recognition Gabor filters". In: Proc. 15th Annu. Conf. Int. Speech Commun. Assoc., pp. 905-909, (2014).

[14] I. Goodfellow, Y. Bengio and A. Courville: "Deep Learning". The MIT Press, pp. 2. Cambridge, USA (2016).

[15] M. Minsky and S. Papert: "Perceptron: an introduction to computational geometry". The MIT Press, Cambridge, USA (1969).

[16] F. Rosenblatt: "Principles of Neurodynamics". Spartan Books, New York, USA (1962).

[17] W.S. McCulloch and W. Pitts: "A logical calculus of the ideas immanent in neuronal nets". Bull. *Math. Biophys*, Vol. 5, pp. 115-137, (1943).

[18] M. Aquid, R. Mehmood, A. Albeshri and A. Alzahrani: "Disaster Management in Smart Cities by Forecasting Traffic Plan Using Deep Learning and GPUs". *Smart Societies, Infrastructure, Technologies and Applications*, pp. 139-154, (2018).

[19] D.E. Rummelhart and J.L. McClelland: "Parallel Distributed Processing: Explorations in Microstructure of Cognition". The MIT Press, Cambridge, USA (1986).

[20] D.L. Ringach: "Mapping receptive fields in primary visual cortex". *J Physiol.*, Vol. 558, Issue 3, pp. 717-728, (2004).

[21] R.E. Soodak: "Two-dimensional modeling of visual receptive fields using Gaussian subunits". *Pro. Natl. Acad. Sci. USA*, Vol.83, Issue 23, pp. 9259-9263, (1986).

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton: "Imagenet classification with deep convolutional neural networks". In *Advances in neural information processing systems*, pp. 1097–1105, (2012).

[23] P. Diederik, P.Kingma and J. Ba: "Adam: A Method for Stochastic Optimization". In 3rd *International Conference for Learning Representations*, San Diego (2015).

[24] D. Yu, H. Wang, P. Chen and Z. Wei: "Mixed Pooling for Convolutional Neural Networks". In *Rough Sets and Knowledge Technology: 9th International Conference*, RSKT 2014, Vol. 24, Issue 26, pp. 364-375, Shanghai (2014).

[25] J. Nagi et al: "Max-Pooling Convolutional Neuronal Networks for Vision-based Hand Gesture Recognition". In *IEEE International Conference on Signal and Image Processing Applications*, pp. 342-247, Kuala Lumpur (2011).

[26] N. Srivastava et al: "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research*, Vol. 15, pp. 1929- 1958, (2014).

[27] He et al: "Deep Residual Learning for Image Recognition". In *29th IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2016, pp. 770-778, Las Vegas (2016).

[28] Huang et al: "Densely Connected Convolutional Networks". In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261-2269, Honolulu (2017).

[29] Huang et al: "Deep Networks with Stochastic Depth". In *ECCV 2016: 14th European Conference*, pp. 646-661, Amsterdam (2016).

[30] A. Veit, M. Wilber and S. Belongie: "Residual Networks Behave Like Ensembles of Relatively Shallow Networks". *In 30th International Conference on Neural Information Processing Systems*, pp. 550-558, Barcelona (2016).

[31] R.A. Young: "Orthogonal basis functions for form vision derived from eigenvector analysis". *ARVO Abstracts Association for Research in Vision and Ophthalmology*, pp. 22, (1978).

[32] J.-B Martens: "The Hermite transform-theory". *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(9), 1595-1606, (1990).

[33] J.L. Silvan and B. Escalante: "The multiscale Hermite transform for local orientation analysis.". *IEEE Transactions on Image Processing*, Vol. 15, Issue 5, 1236-1253, (2006).

[34] A.M. van Dijk and J.-B. Martens: "Image representation and compression with steered hermite transforms". *Signal Processing*, Vol. 56, Issue 1, pp. 115-137, (1943).

[35] S.L. Chen, B. Zhang, J. Han, and J. Liu: "Gabor Convolutional Networks". *IEEE Transactions on Image Processing*, 27 (9), 4357-4366 (2018).

[36] K. Kowsari et al.: "RMDL: Random Multimodel Deep Learning for Classification". *2nd International Conference on Information System and Data Mining*, Lakeland (2018).

[37] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus: "Regularization of Neural Networks using DropConnect". In: *Proceeding ICML'13 Proceedings of the 30th International Conference on International Conference on Machine Learning*, vol. 28, pp. 1058-1066. Atlanta (2013).

[38] D. Ciregan, U. Meier, and J. Schmidhuber: "Multi-column deep neural networks for image classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp.3642- 3649. IEEE Computer Society, Washington (2012).

[39] I. Sato, H. Nishimura, and K. Yokoi, "APAC: Augmented PAttern Classification with Neural Networks," Available: https://arxiv.org/abs/1505.03229, last accessed: 2019/03/24.