Universidad Nacional de Mar del Plata
Facultad de Ingeniería
Departamento de Electrónica y Computación

Aplicación de técnicas basadas en aprendizaje profundo para la clasificación de imágenes satelitales y otras plataformas de observación terrestre

Proyecto Final Ingeniería en Computación

Betti, Ayrton Giraldez, Rocío Natalia

ayrtonbetti@mdp.edu.ar rociogiraldez@mdp.edu.ar

Directora: Dra. Leticia M. Seijas

Codirector: Ing. Jorge Marquez

El presente trabajo de Proyecto Final fue realizado en el Laboratorio de Comunicaciones del Departamento de Electrónica y Computación, ICyTE, Facultad de Ingeniería, Universidad Nacional de Mar del Plata.

Mar del Plata, Diciembre de 2020

Resumen

Los algoritmos de aprendizaje profundo intentan aprender características jerárquicas, correspondientes a diferentes niveles de abstracción. El progreso actual en los modelos de aprendizaje profundo, específicamente en las arquitecturas de redes neuronales convolucionales profundas (CNN), ha mejorado el estado del arte en muchos campos de estudio, incluida la clasificación de escenas de sensado remoto. La elección de una arquitectura de red adecuada para hacer suposiciones sólidas y correctas sobre la naturaleza de los datos de entrada sigue siendo un gran desafío. Este trabajo presenta implementaciones de la CNN AlexNet, la cual se entreno en los conjuntos de datos UC Merced Land Use y WHU-RS mediante el uso de la técnica de Transfer Learning para el problema de clasificación de escenas urbanas de imágenes de alta resolución espacial a clasificar. Se incorpora una capa correspondiente a Spatial Pyramid Pooling (SPP) para hacer uso de diferentes tamaños de imágenes en la entrada a la red. Los resultados son comparables a la literatura, mejorando algunos enfoques publicados.

Palabras Clave: CNN, Transfer Learning, Spatial Pyramidal Pooling, AlexNet, HSR.

Abstract

Deep learning algorithms attempt to learn hierarchical features, corresponding to different levels of abstraction. Current progress in deeplearning models, specifically deep convolutional neural network (CNN) architectures, have improved the state-of-the-art in many fields of study including remote sensing scene classification. The choice of a proper network architecture for making strong and correct assumptions about the nature of the input data is still a big challenge. This work presents an implementation of the Convolutional Neural Network (CNN) AlexNet trained on the well-known datasets UC Merced Land Use and WHU-RS by using Transfer Learning for the High Spatial Resolution Image Scene Classification problem. A layer corresponding to the Spatial Pyramid Pooling (SPP) is incorporated in order to consider different image sizes in the input of the network and multi-scale spatial information of the scenes to be classified. Results are comparable to literature, improving some published approaches.

Keywords: Scene classification, CNN, Transfer Learning, Spatial Pyramidal Pooling, AlexNet, HSR.

Índice general

Índice de Figuras

Índice de Tablas

Índice de Acrónimos

ALU Unidad Aritmético-Lógica.

CISC Conjunto de Instrucciones Complejas.
CNN Red Neuronal Convolucional.
CPU Unidad de Procesamiento Central.
CV validación cruzada.

DL Deep Learning.

FC Fully Connected.

GPGPU computación de propósito general en unidades de procesamiento gráfico.
GPU Unidad de Procesamiento Gráfico.

HPC Computación de Alto Rendimiento.
HSR Alta Resolución Espacial.

IA Inteligencia Artificial.
ILP paralelismo a nivel de instrucción.
ISA Conjunto de Instrucciones.

LSI Integración a Gran Escala.

MCCV Montecarlo Cross Validation.
ML Machine Learning.
MPI interfaz de paso de mensajes.

NUMA Acceso No Uniforme a Memoria.

PC computadora personal.
PSO Optimización por Enjambre de Partículas.

RAM Memoria de Acceso Aleatorio.
RISC Conjunto de Instrucciones Reducido.
RNA Red Neuronal Artificial.

SGD Descenso de Gradiente Estocástico.
SMP multiprocesador simétrico.
SPP Pooling Piramidal Espacial.

SSI Integración a Baja Escala.

TL Transfer Learning.

ULSI Integración a Escala Ultra Grande.
UMA Acceso Uniforme a Memoria.

VLSI Integración a Escala Muy Grande.

Capítulo 1

Introducción

El desarrollo acelerado en el área del sensado remoto de imágenes de Alta Resolución Espacial (HSR) ha generado una gran cantidad de datos con abundante información detallada y estructural pero sin las etiquetas correspondientes, lo que demandaría una gran cantidad de trabajo humano para lograr un etiquetado de las mismas [1]. Sin embargo, durante las últimas décadas, se han realizado notables esfuerzos en el desarrollo de varios métodos para la tarea de clasificación de escenas de imágenes provenientes de la teledetección debido a su importante función para una amplia gama de aplicaciones prácticas, como la detección de peligros naturales, el uso de la tierra y la determinación de la cobertura terrestre, recuperación de imágenes, mapeo de vegetación, monitoreo ambiental y planificación urbana [2].

El progreso actual en los modelos de Deep Learning (DL), específicamente las arquitecturas de Red Neuronal Convolucional (CNN), han mejorado el estado del arte en el reconocimiento y detección de objetos visuales, el reconocimiento de voz y muchos otros campos de estudio [3]. El modelo descrito por Krizhevsky et al. [4], al que se hace referencia con frecuencia como AlexNet, se considera un gran hito que influyó en la rápida adopción de técnicas basadas en DL para el campo de la visión artificial [5].

Los algoritmos de DL aprenden conocimiento jerarquizado características jerárquicas, correspondientes a distintos niveles de abstracción. Las redes convolucionales se han inspirado en la biología y consisten en una arquitectura compuesta de múltiples capas convolucionales, de pooling y totalmente conectadas o Fully Connected (FC), que se pueden entrenar mediante la técnica de aprendizaje supervisado. Sin embargo, en la práctica es difícil entrenar una red de estas características con pequeños conjuntos de datos. En la actualidad, numerosas publicaciones [6] han demostrado que las activaciones intermedias aprendidas por CNN entrenadas previamente en grandes conjuntos de datos como ImageNet [7] pueden transferirse a otras tareas de reconocimiento que presenten datos de entrenamiento limitados [8].

Aunque el uso de redes pre-entrenadas y la técnica de Transfer Learning (TL) pueden ayudar a que implementaciones de redes convolucionales que disponen de cantidades limitadas de muestras etiquetadas logren un rendimiento de clasificación satisfactorio para escenarios de imágenes de sensores remotos HSR, la elección de una arquitectura de red adecuada para hacer suposiciones sólidas y correctas sobre la naturaleza de los datos de entrada es un gran desafío. Por lo tanto, se necesita con urgencia la investigación de una arquitectura de red simple con una potente capacidad de modelado.

En este trabajo de final de carrera se ha realizado un estudio sobre la problemática y

el estado del arte de la clasificación de escenarios urbanos resultantes del sensado remoto y su abordaje con técnicas de aprendizaje profundo. Luego de la presentación de las principales arquitecturas de redes de convolución y de las características principales del sesado remoto, se describen los aspectos de la implementación realizada de clasificadores basados en la arquitectura de CNN AlexNet clásica y AlexNet con una capa de Pooling Piramidal Espacial (SPP). Estos modelos fueron entrenados con los conjuntos de datos estándar UC Merced Land Use y WHU-RS [9][8] mediante el uso de Transfer Learning y el fine-tuning para el problema de clasificación de escenarios urbanos de imágenes de alta resolución espacial.

La estrategia Pooling Piramidal Espacial se introduce para considerar diferentes tamaños de imagen en la entrada de la red y mejorar el conocimeinto jerarquico adquirido. También se aplicaron otras estrategias, como Data Augmentation para lidiar con el problema de las escasas muestras en el conjunto de entrenamiento y, a su vez, con el overfitting. Otra característica de nuestro modelo es la utilización del optimizador Adam para mejorar la convergencia. Los resultados alcanzados son satisfactorios y comparables a la literatura.

Este trabajo se organiza de la siguiente manera: el Capítulo 2 describe los antecedentes de la clasificación de escenarios urbanos con redes de convolución con la técnica de Transfer Learning. El Capítulo 3 desarrolla el concepto de CNN, Transfer Learning y la arquitectura a utilizar AlexNet, junto a otras arquitecturas de importancia dentro del concurso de ImageNet ILSVRC. El Capítulo 4 muestra las distintas cuestiones a tener en cuenta a la hora de entrenar una red de convolución, la cual incluye la exploración de hiperparámetros y la validación cruzada cómo elemento estadístico para evaluar el rendimiento en este proceso de entrenamiento. En el Capítulo 5 se detallan las distintas plataformas de hardware y software que permiten llevar a cabo el aprendizaje profundo. En el Capítulo 6 se menciona el proceso de sensado remoto y las distintas plataformas terrestres disponibles para este proceso. El Capítulo 7 describe las implementaciones logradas y muestra los resultados experimentales. Finalmente el Capítulo 8 presenta la conclusión y posibles trabajos futuros.

Capítulo 2

Técnicas de Aprendizaje Profundo para clasificación de señales de Sensado Remoto

Actualmente estamos atravesando una época en la que cada día que pasa se adquieren enormes cantidades de imágenes de alta resolución por medio del sensado remoto. Este fenómeno nos permite estudiar la superficie de la Tierra con mayor detalle. La clasificación de las fotografías en distintos escenarios y, a su vez, de sus subregiones, con el objetivo de mapear las características visuales en ellas al plano semántico de manera automática es una tarea crucial para numerosas aplicaciones. Entre ellas se pueden nombrar al urbanismo moderno, la detección de objetos geoespaciales, la agricultura, el monitoreo ambiental y la vigilancia militar. Hay muchas técnicas que han sido aplicadas en el pasado a estas situaciones, sin embargo, desde que surgieron las CNN ha habido un enorme progreso en aquellos dominios de aplicación encargados de encontrar patrones en conjuntos de datos que presenten relaciones espaciales entre sus componentes, como lo son las imágenes [10].

Generalmente hay numerosas características compartidas entre distintas clases de objetos o distintos escenarios terrestres. Por ejemplo, encontramos varias similitudes entre una zona residencial y una comercial, las cuales son categorías de escenas típicas en un dataset. Ambas pueden contener carreteras, árboles y edificios, pero difieren en la densidad y la distribución de los elementos espaciales. Dicha complejidad del espacio y los patrones estructurales que presentan los panoramas dentro de los conjuntos de sensado remoto generan que la clasificación de escenas sea un problema desafiante [8].

Los esquemas que se utilizaban hasta hace algunos años en la clasificación de escenarios de Alta Resolución Espacial no lograban resultados del todo convincentes. Si bien se utilizaban modelos clásicos exitosos para ciertas aplicaciones, como *Bag Of Words* y el aprendizaje no supervizado, estos eran incapaces de generar representaciones poderosas aprovechando todos los elementos y las relaciones espaciales de bajo nivel entre los datos de entrada. Estas técnicas pasadas generan representaciones de características de nivel medio pero no tienen comparación a aquellas que son representadas por los algoritmos de aprendizaje profundo. Estos últimos abarcan distintos niveles de abstracción, aprendiendo características jerárquicas y, específicamente, las CNN se conocen como los métodos dominantes en la mayoría de las tareas de detección y reconocimiento en base a sus notables e increíbles resultados en numerosos estudios de referencia [8].

Un problema al tratar de entrenar un modelo es la cantidad de muestras disponibles. Hay muchos datasets modernos que poseen un pequeño número de escenas lo cual condiciona y limita el entrenamiento. Para lidiar con esta situación, una destacable y efectiva

técnica en el campo de clasificación de imágenes satelitales con redes convolucionales es la introducción del mecanismo de pre-entrenamiento también denominado *Transfer Learning (TL)*. Una arquitectura pre-entrenada involucra el proceso de entrenar al modelo con un dataset grande y muy abarcativo, para luego transferir ese aprendizaje y aprovecharlo con el set mas reducido. Marco [11] fue el primero en probar que la transmisión de entrenamiento entre distintas CNN puede alcanzar una performance de clasificación remarcable.

Mas allá de que el mecanismo de pre-entrenamiento puede ayudar a los modelos convolucionales a alcanzar resultados de clasificación muy satisfactorios, sorteando las dificultades de disponer de limitados conjuntos de datos, la elección de una red adecuada implica otro gran desafío que afecta directamente a los resultados finales. Por lo tanto, es de gran importancia que se prueben e investiguen las arquitecturas simples que posean capacidades de modelado poderosas. AlexNet es una red típica, clásica, que ha marcado un antes y un después desde su nacimiento y un modelo sumamente utilizado en estas áreas del conocimiento. Fue presentada exitosamente en la competencia ImageNet del 2012 [4] por primera vez. Esta arquitectura es capaz de alcanzar resultados de clasificación muy buenos si se la entrena correctamente. Sin embargo, considerando las características multi-escala presentes en algunos escenarios, las propiedades de la red AlexNet pueden ser limitadas para aplicaciones particulares. Para ello se necesita otra mejora a la red, con el objetivo de lograr una mejor adecuación a la problemática a enfrentar.

En líneas de alcanzar resultados que estén a la altura de los trabajos publicados hasta el momento y para que la red procese la información de entrada aprovechando cada detalle de los datos disponibles se hace uso de una estrategia multi escala que se conoce como Pooling Piramidal Espacial (SPP). Este es un módulo, propuesto por He [12], que se incorpora a la red pre entrenada y opera en las capas finales de AlexNet, concatenando mapas de características de diferentes dimensiones lo cual aprovecha toda la información espacial de una misma escena y es crucial para obtener resultados comparables con el estado del arte.

Capítulo 3

Redes Neuronales Convolucionales CNN

3.1. Introducción a las Redes Neuronales Artificiales

El cerebro es considerado como un sistema de procesamiento de información altamente complejo, no-lineal y que trabaja realizando su ejecución en simultáneo. A su vez, una red neuronal de cualquier ser vivo se compone de un conjunto de neuronas conectadas entre sí, las cuales a medida que reciben estímulos, reaccionan a estos, dando como resultado una salida y reforzando algunas conexiones, proceso conocido como aprendizaje. Inspiradas en estas redes biológicas, surge la **Red Neuronal Artificial (RNA)**.

Lo que se busca es que el propio sistema pueda aprender a partir de ejemplos, asemejando el proceso de aprendizaje propio de un ser humano. Entonces, una RNA es un modelo matemático diseñado e inspirado en la manera en que el cerebro realiza una tarea particular o función de interés. Esta red es implementada haciendo uso de componentes electrónicos o simulados en software de manera digital en un dispositivo de cómputo [13].

Existen varios tipos de RNA en función de su arquitectura y forma de aprendizaje. La implementación más sencilla es el **perceptrón simple**, modelo matemático que se puede observar en la Figura 3.1, el cual describe el comportamiento de una única neurona artificial.

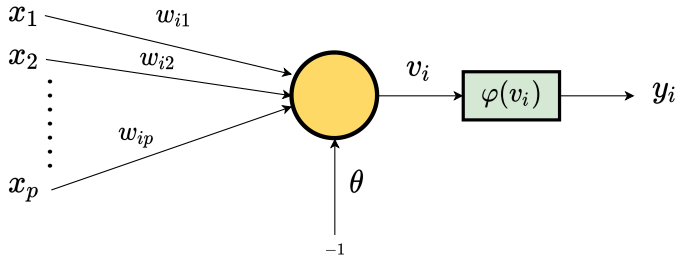


Fig. 3.1: Entrenamiento del perceptrón simple con p entradas [13].

Siendo $w_{i1}, w_{i2}...w_{ip}$ los pesos sinápticos, $x_1, x_2...x_p$ las entradas aplicadas al perceptrón, θ el bias o entrada de umbral externa y φ la función de activación. Luego la operación de una red con p neuronas de entrada y m salidas es:

$$v_i = \sum_{j=1}^{p} w_{ij} x_j$$

$$y_i = \varphi(\sum_{j=1}^{p} w_{ij}x_j - \theta), \vee i, 1 \leq i \leq m$$

donde se da la suma ponderada de las señales de entrada y es comparada con el bias para determinar la salida de la neurona. Cuando la suma es mayor o igual al bias, la salida es igual a 1, en cambio cuando la suma es menor al bias, la salida es cero.

El perceptrón simple tiene una limitación fundamental: sólo puede resolver problemas de clasificación lineales. Esto conlleva a que en problemas más complejos se utilice una arquitectura más sofisticada, conocida como el **perceptrón multicapa**. Esta última se conforma a partir de la interconexión ordenada en capas de una multitud de perceptrones simples. En comparación con implementaciones más sencillas, un perceptrón multicapa es un modelo capaz de aprender datos de una mayor complejidad. Puede reconocer y clasificar patrones de todo tipo de formas, lográndose una mayor potencialidad de uso en áreas de mayor necesidad.

A fin de poder hacer uso de una Red Neuronal Artificial, se la debe de entrenar con un conjunto de datos coherente y suficiente. La etapa de entrenamiento de una RNA es la más importante ya que en esta a partir de los datos que ingresan al modelo se adaptan los parámetros libres: pesos y biases. La red adquiere el conocimiento de su entorno luego de pasar por un algoritmo de aprendizaje. Las fuerzas de las interconexiones de neuronas, es decir los pesos sinápticos w_{ip}, se usan para almacenar el conocimiento adquirido.

Es importante recalcar que se debe de controlar la etapa de entrenamiento para evitar el overfitting, que ocurre cuando se sobreentrena demasiado al sistema y la red memoriza las muestras, respondiendo de manera incorrecta a nuevas consultas. Así pues, se dice que una red generaliza bien cuando obtenemos salidas correctas para entradas no utilizadas en la etapa de aprendizaje.

3.2. El aprendizaje profundo y las redes de convolución

El aprendizaje profundo o Deep Learning (DL) es un conjunto de algoritmos de aprendizaje automático basados en asimilar representaciones de datos. Deep Learning pertenece al campo del aprendizaje de máquina, y se trata de una nueva forma de aprender a partir de la representación de datos, que pone énfasis el aprender a partir del uso de sucesivas capas de procesamiento [14].

Estos algoritmos intentan modelar abstracciones de alto nivel basándose en datos ya etiquetados, usando arquitecturas computacionales que admiten transformaciones no lineales múltiples e iterativas de la información, expresada en forma de tensores. Los algoritmos de aprendizaje profundo contrastan con los algoritmos de aprendizaje poco profundo por el número de transformaciones aplicadas a la señal mientras se propaga desde la capa de entrada a la capa de salida.

Si el numero de capas entre la neurona de entrada y de salida es grande, se habla de una red profunda. No hay un consenso establecido, sin embargo, cuando hablamos de más de ocho capas se la considera profunda o *deep* [4]. Las redes neuronales profundas pueden tener cientos de capas [15].

Una Red Neuronal Convolucional es una red neuronal profunda, algoritmo de Deep Learning, considerada como una variante especial de un perceptrón multicapa. El nombre *red neuronal convolucional* indica que la red emplea una operación matemática llamada convolución, un tipo especializado de operación lineal, en lugar de la multiplicación matricial general, en al menos una de sus capas. Una red convolucional es un perceptrón

multicapa diseñado específicamente para reconocer formas de dos dimensiones con un alto grado de invarianza a la traslación, escalado, inclinación, entre otras formas de distorsión.

El uso clásico y más popular de estas redes es el de procesamiento de imágenes y reconocimiento de patrones visuales. Esto se debe a que una CNN no deforma las dimensiones de las entradas. Se hará foco en el problema de clasificación de datos provenientes del proceso del sensado remoto, tratando de determinar a qué clase pertenece la entrada basándose en los patrones que se identifiquen en ella.

Las redes convolucionales tienen una reminiscencia de las células simples y complejas en la corteza visual primaria [16]. La idea detrás del desarrollo de estas redes se remonta del trabajo pionero *"Receptive fields, binocular interaction and functional architecture in the cat's visual cortex"* publicado en 1962 por D. Hubel y T. Wiese [17]. En éste se relata que las neuronas de la corteza visual disparan potenciales de acción cuando el estímulo visual aparece dentro de su campo receptivo. En las áreas visuales tempranas, las neuronas tienen una sintonía simple y en las áreas visuales altas, la sintonía neuronal se vuelve más compleja. Esto se traduce en que la primer convolución es capaz de detectar características primitivas como líneas o curvas. A medida que nos adentremos en la red, las capas convolucionales reconocerán formas más complejas.

Las CNN utilizan el método de aprendizaje supervisado y su estructura incluye las siguientes etapas [13]:

- Extracción de características: cada neurona toma su entrada sináptica de un campo receptivo local de la capa anterior, logrando la extracción de características locales. Una vez que cierta característica o propiedad ha sido extraída, su ubicación exacta se vuelve menos importante, mientras que su posición relativa a otras características es preservada. Una manera simple de reducir la precisión de la posición de estas características halladas es reduciendo la resolución espacial del mapa de características, esto se puede lograr con las capas de *pooling* [18].

- Mapeo de características: cada capa convolucional de la red está compuesta de múltiples mapas de características. Estos realizan la convolución con un filtro o kernel de tamaño pequeño seguido por una función de activación. El filtro de convolución es el conjunto de pesos de conexión usados por las neuronas en el mapa de características.

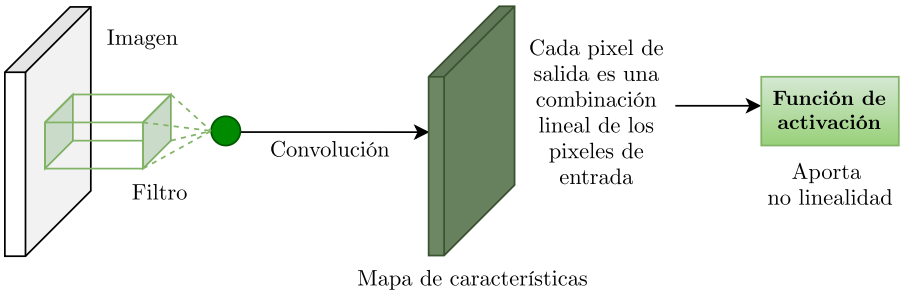


Fig. 3.2: El mapa de características (o de activación) es la salida de un filtro aplicado a la capa anterior. Un filtro dado es convolucionado por toda la capa anterior, moviéndose por los pixeles [13].

Los pesos y biases son los mismos para todas las neuronas que forman los mismos mapas de caracteristicas de una capa oculta o profunda. De esta manera todas las neuronas

de un mapa de característica de la capa oculta están detectando la misma característica en diferentes partes de la imagen. El uso de pesos compartidos hace posible que la implementación de estas redes sea de forma paralela, acelerando el entrenamiento. Dependiendo de los valores de los coeficientes del filtro, la convolución puede implementar una detección de un borde local, un filtro pasabajos o algo completamente diferente.

Todos los pesos en las capas de convolución son aprendidos en la etapa de entrenamiento. La forma en que la red aprende a ajustar sus propios valores de los filtros es a través de un proceso de entrenamiento llamado retropropagación del error (*backpropagation*). Además, la red aprende a extraer sus propias características automáticamente. Debido al intercalamiento de capas de convolución y de submuestreo (pooling) se obtiene un efecto bipiramidal. En cada capa, el número de mapas de características se incrementa mientras que la resolución espacial comparada con la capa anterior se reduce.

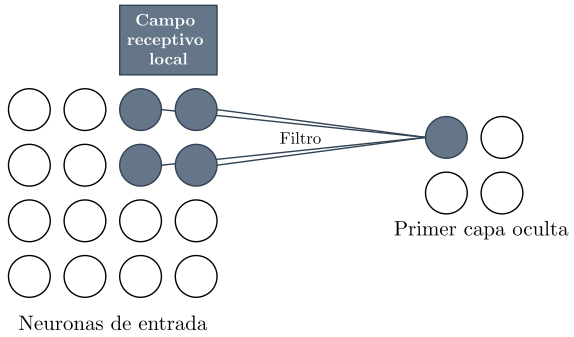


Fig. 3.3: Suponiendo que tengamos una imagen de entrada de 4×4 píxeles, tendremos 4×4 neuronas de entrada que reciben los 4×4 valores de intensidad de los píxeles. Conectamos las neuronas de entrada a una capa oculta, pero no vamos a conectar cada neurona de entrada a cada neurona oculta. En cambio, sólo haremos conexiones en regiones pequeñas y localizadas de la imagen de entrada. La región en la imagen de entrada se la llama campo receptivo local. La red toma el concepto de capas con especialización, cada neurona de una capa no recibe conexiones entrantes de todas las neuronas de la capa anterior, sino de algunas [13].

En una Red Neuronal Convolucional se pueden diferenciar cuatro capas principales, independientemente de la arquitectura trabajada, las cuales se esquematizan en la Figura 3.4:

1. Capa de convolución: usa un filtro (matriz) sobre los píxeles de la imagen y realiza la operación de convolución para obtener un mapa de características (feature map). Cada uno de los filtros pueden ser considerados como identificadores de características (como bordes).

2. Capa de no-linealidad: se suele usar la función de activación ReLU (pone todos los píxeles negativos en cero) para lograr la no-linealidad en la red. Luego la imagen original pasa por varias capas de convolución y ReLU de la red feed-forward para localizar características y patrones de la misma.

3. Capa de pooling: se encarga de la reducción de dimensionalidad del mapa de características. Esta operación tiene el efecto de reducir la sensibilidad a la traslación y otras formas de distorsión de la entrada. Simplifica la información de la salida de la capa convolucional. La capa mencionada opera con cada mapa de características de

manera separada para crear un nuevo conjunto con la misma cantidad de mapas. El tamaño de la operación de pooling es más pequeño que el tamaño del mapa, generalmente de 2×2 pixeles. Las operaciones de pooling son de promedio o de máximo.

4. Capa totalmente conectada: la matriz de mapa de características es convertida en un vector con el cual se alimenta un modelo de RNA. Finalmente tenemos una capa de softmax o sigmoid para clasificar las salidas en las clases que correspondan. Esta capa final no aporta a la clasificación. Por ejemplo, si se utiliza una capa final de softmax, esta aplica una transformación a los valores de salidas de manera que la salida final pueda interpretarse como un vector de probabilidades o una distribución multinomial.

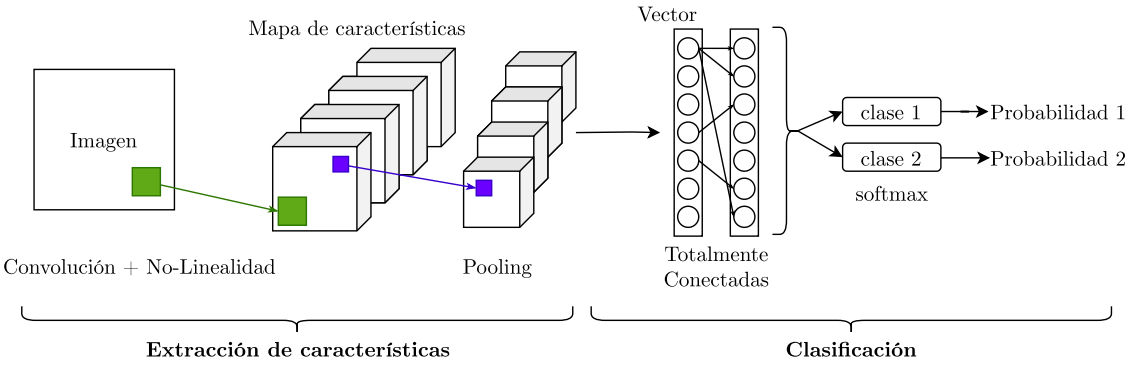


Fig. 3.4: La etapa de extracción de características esta compuesta de una pila de capas de neuronas convolucionales, función de activación y de reducción de muestreo o pooling. Al final de la red tenemos la etapa de clasificación donde se encuentran las capas de neuronas totalmente conectadas que utilizan las características extraídas en las capas anteriores.

3.3. ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

La mayoría de las arquitecturas que se desarrollan en el presente proyecto final de carrera surgieron en el contexto del concurso ILSVRC [19], llevado a cabo entre el 2010 y el 2017. Este consistía en una competencia anual en donde participaban numerosas instituciones y cada una de ellas presentaba la red neuronal que había diseñado. Las diferentes arquitecturas eran entrenadas con sets de público acceso y existían distintas tareas de clasificación en donde se ponían a prueba las redes.

Ahondar en el concurso *per sé* escapa nuestro enfoque, entonces se procederá a mencionar de manera breve tres escenarios de competencia diferentes y se mencionará en base a qué criterios se determinaba el ganador.

El objetivo principal era conseguir el menor porcentaje de error en la clasificación de imágenes, esto es, simplemente, acertar al determinar a qué clase pertenece la imagen de entrada, entre otras categorías de la competencia. Para todas las tareas de clasificación, se tomaba un subconjunto de la base de datos llamada ImageNet, la cual cuenta con más de 14 millones de imágenes etiquetadas por seres humanos, y en total conforman 1000 clases diferentes.

Al poner a prueba una arquitectura, se obtiene como resultado la predicción en formato de distribución multinomial. En otras palabras, se obtienen probabilidades de pertenencia para las 1000 categorías diferentes. Se evalúan dos perspectivas: si la probabilidad de pertenencia más alta obtenida se corresponde con la salida esperada, llamado *error top-1* y también se registran las cinco clases que obtuvieron las probabilidades más altas si una de ellas es la correcta, conocido como *error top-5*. Luego se procede a computar los resultados de las pruebas. En ambos casos, la puntuación se calcula como las veces que una etiqueta predicha coincide con la etiqueta objetivo, dividida por el número de puntos de datos evaluados y finalmente se obtienen a los ganadores.

Cuando las innovaciones se comenzaron a estancar, aproximadamente en el 2017, debido a que ya se alcanzaban niveles de exactitud extraordinarios, se dejó de lado este certamen. Sin embargo, las arquitecturas surgidas en la competencia y entrenadas con ImageNet son ampliamente utilizadas en la actualidad en distintas áreas de aplicación con excelentes resultados, y como base en distintas líneas de investigación. A continuación, describiremos las más representativas para nuestro trabajo.

3.4. AlexNet

AlexNet fue la primer Red Neuronal Convolucional a gran escala que logró buenos resultados en la tarea de clasificación de la base de datos ImageNet [4]. Esto se debe a su arquitectura de ocho capas, que a continuación se analizará en profundidad, y a la metodología de entrenamiento. Si bien hoy en día se han superado los resultados que se obtuvieron con esta red para algunas aplicaciones, es muy conocida por la influencia y el reconocimiento alcanzado desde su publicación. A continuación describiremos la arquitectura y características en base a lo presentado en el artículo original [4].

3.4.1. Arquitectura

La red se compone de ocho capas con parámetros entrenables [4]. Estos se encuentran en aquellas que contienen convoluciones y las que se encuentran al final, llamadas Fully Connected (FC) o totalmente conectadas. Las características particulares de las capas que componen esta red se pueden apreciar en la Tabla 3.1.

En el paper original [4] se muestra un esquema idéntico al de la Figura 3.6, en la cual está graficada una entrada a la red de $224 \times 224 \times 3$. Sin embargo, se demostrará en la sección 7.2.1 que la red funciona con entradas de $227 \times 227 \times 3$.

Capa	Tamaño de entrada a la capa	Filtros
CONV1	$227 \times 227 \times 3$	96 filtros 11×11 stride 4, pad 0
MAX POOL1	$55 \times 55 \times 96$	filtros 3×3, stride 2 Mantiene la profundidad de la entrada
NORM1	$27 \times 27 \times 96$	Capa de normalización
CONV2	$27 \times 27 \times 96$	256 filtros 5×5, stride 1, pad 2
MAX POOL2	$27 \times 27 \times 256$	3×3 filtros, stride 2
NORM2	$13 \times 13 \times 256$	Capa de normalización
CONV3	$13 \times 13 \times 256$	384 filtros 3×3, stride 1, pad 1
CONV4	$13 \times 13 \times 384$	384 filtros 3×3, stride 1, pad 1
CONV5	$13 \times 13 \times 384$	256 filtros 3×3, stride 1, pad 1
MAX POOL3	$13 \times 13 \times 256$	3×3 filtros, stride 2
FC6	9216	4096 neuronas
FC7	4096	4096 neuronas
FC8	4096	1000 (puntaje de clase)

Tab. 3.1: Arquitectura de AlexNet

3.4.2. ReLU

Una de las características más importantes de AlexNet es que fue la primera en utilizar la función de activación no lineal Rectified Linear Unit abreviada como ReLU. Esta función se comporta como se define como:

$$f(x) = max(0, x)$$

Hasta ese momento se utilizaban las siguientes funciones de salida [20]:

$$f(x) = tanh(x) \text{ ó } f(x) = (1 + e^{-x})^{-1}$$

La función tangente hiperbólica y logística sigmoidea tienen zonas de saturación que provocan que las salidas de las neuronas se restringan a ciertos rangos de valores, lo que a su vez genera el efecto de saturación del gradiente y que, por lo tanto, a la hora de realizar la retropropagación para el ajuste de pesos, los parámetros de la red no sufran cambios apreciables, lo que genera un estancamiento en el proceso de entrenamiento.

La ventaja de usar la función ReLU radica en que no posee estas regiones de saturación ya que, como se puede apreciar en la Figura 3.5, esta tiene un comportamiento lineal para entradas positivas, evitando así el problema de desvanecimiento y la saturación de las neuronas. Con el término saturar se hace referencia a que los valores de salida no se encuentran acotados en un rango particular, con lo cual los incrementos entre iteración e iteración son notables. Matemáticamente hablando:

- f es no saturable si:

$$(|\lim_{z \to -\infty} f(z) = +\infty|) \vee (|\lim_{z \to +\infty} f(z) = +\infty|)$$

- f es saturable si f **no es** no saturable.

Las Red Neuronal Convolucional que utilizan la función de activación ReLU son considerablemente más rápidas en términos de velocidad de entrenamiento al compararlas con aquellas que utilizan *tanh* u otras funciones de activación similares.

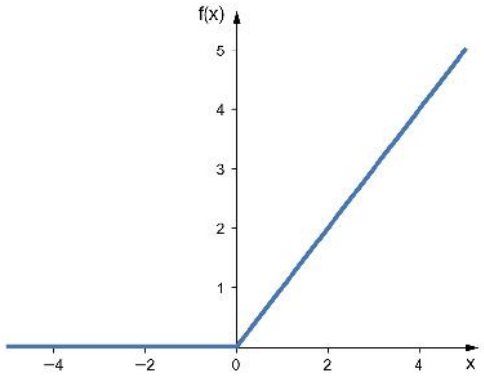


Fig. 3.5: Función de activación ReLU.

3.4.3. Pooling superpuesto

Las capas de pooling realizan una operación de submuestreo en una región $z \times z$, obtienen el resultado y luego pasan a los próximos píxeles, descartando los ya procesados.

Puede pensarse que una capa de pooling consiste en una cuadrícula de unidades espaciadas de s píxeles, cada una resume un vecindario de tamaño $z \times z$ centrado en la ubicación de la unidad de pooling. Si se establece $s = z$, se obtiene la agrupación local tradicional como se emplea comúnmente en las CNN. Al establecer $s < z$, se obtiene una agrupación superpuesta. Se ha demostrado que este tipo de pooling colabora a disminuir el overfitting.

Esta última se corresponde a la estructura utilizada en la arquitectura original de AlexNet, con $s = 2$ y $z = 3$.

3.4.4. Entrenamiento

Al observar detenidamente el diagrama de AlexNet en la Figura 3.6 se ve que la arquitectura se esquematiza dividida en dos. Gráficamente se observa que luego de la primera capa de convolución, que arroja una salida de profundidad 96 se divide en dos partes de profundidad de 48 cada una. En efecto, esto se debe a que la red fue entrenada en el paper original en dos unidades de procesamiento gráfico o GPU debido a que el modelo completo no entraba en una sola. En el Capítulo 5 se hablará sobre los recursos de hardware para el entrenamiento de las redes profundas detalladamente.

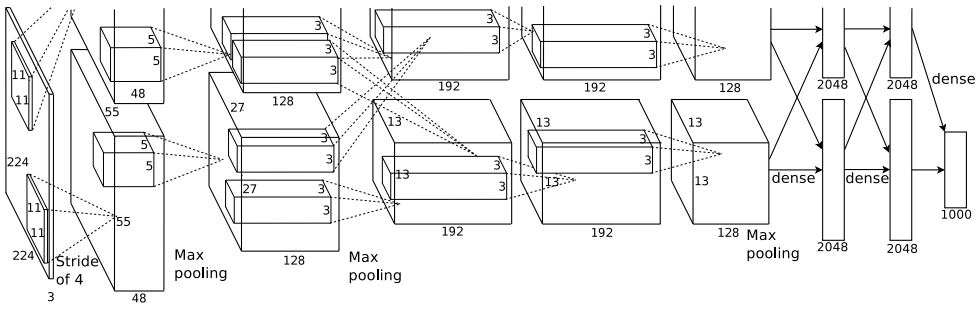


Fig. 3.6: Una ilustración de la arquitectura de la red convolucional del paper original de AlexNet. [4, Figura 2]

Los diseñadores de esta red sostenían que 1.2 millones de ejemplos de entrenamiento eran suficientes para entrenar a una red muy grande en una sola GPU. Dichos elementos de hardware son fácilmente paralelizables. Sin embargo, disponiendo de dos, uno puede acceder directamente a la memoria del otro y la dificultad radica en determinar las capas de la red que deben comunicarse, pensando, sobre todo, en la validación cruzada.

Finalmente, luego de probar distintos enfoques, encontraron que la arquitectura resultante mostrada en la Figura 3.6, la cual no solo disminuye el tiempo de entrenamiento, sino que también, daba mejores resultados si se comparaba al utilizar únicamente la mitad de la red en un único GPU.

En cuanto a los pesos en cada capa, fueron inicializados a partir de una distribución gaussiana de media cero con desviación estándar 0.01. Los biases en las capas convolucionales segunda, cuarta y quinta, así como en las capas ocultas completamente conectadas fueron inicializados con la constante 1 (uno). Esta inicialización acelera las primeras etapas del aprendizaje al proporcionar a las ReLU entradas positivas. En las capas restantes se setearon los biases a 0 (cero).

En la implementación original de AlexNet se utilizó una tasa de aprendizaje igual para todas las capas, que fue ajustada manualmente durante el entrenamiento cuando esta se estancaba. Lo que se hizo fue dividir la tasa de aprendizaje por 10 cuando el error de validación no mejoraba con la tasa de aprendizaje empleada. La tasa de aprendizaje se inicializó en 0.01 y se redujo tres veces antes de finalizar el entrenamiento.

Describiendo las conexiones de la arquitectura, los filtros de la segunda, cuarta y quinta capa convolucional están conectados únicamente a los filtros de la capa previa que residen en la misma GPU (ver Figura 3.6). Los filtros en la tercer capa de convolución están conectados completamente con todos los de la segunda capa en la otra GPU. También hay comunicación entre GPUs en las capas FC al final de la red. La función ReLU se aplica a la salida de cada capa convolucional y a las totalmente conectadas. Se evitó hablar de detalles matemáticos de las capas de normalización utilizadas porque son bastante complejas y se descubrió que no generan mejoras importantes [20].

3.4.5. Data Augmentation

La técnica principal para reducir el overfitting es tener transformaciones sutiles de las imágenes presentadas. El costo de procesar imágenes puede ser computacionalmente alto. Para que no interfiera con el entrenamiento, una opción es generar un lote con las imágenes transformadas, previo a presentarle las mismas a la red, mientras esta está ocupada analizando el lote previo. De esta manera tampoco se ocupa espacio en disco durante mucho tiempo. Los autores Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton emplearon dos métodos para el aumento de datos.

El primero consiste en hacer recortes aleatorios de la imagen a presentar. Originalmente hay que preprocesar el set de datos ya que las imágenes suelen poseer distintas dimensiones a las que acepta la CNN. En el caso del set de ImageNet son de $256 \times 256 \times 3$ píxeles. Lo que se hace es tomar porciones de 227 píxeles tomando el centro en distintos puntos. De una única imagen se pueden extraer diez, limitando los parches de $227 \times 227 \times 3$ en las cuatro esquinas, luego en el centro y espejando todos los resultados.

La segunda forma adoptada en el paper original de AlexNet para aumentar el volumen de datos de entrenamiento fue cambiar la intensidad de los canales RGB, aumentando los componentes principales en un valor aleatorio. Esto les ayudó a disminuir el error *top-1*

en un 1 % y se sostiene en la premisa en que la identidad de un objeto es independiente de la intensidad de los colores y la iluminación en la imagen.

3.4.6. Dropout

La técnica de dropout consiste en "apagar" una de las neuronas ocultas con una probabilidad de 0.5. Con apagar se refiere a que el valor de salida valga cero. En este caso, dicha neurona, tampoco participará en el cálculo de retropropagación del error. Este método de abandono reduce las dependencias complejas entre las neuronas, ya que una no puede depender de la existencia de las otras.

En la arquitectura original de AlexNet se utiliza dropout en las dos primeras capas completamente conectadas. En cuanto al costo computacional, la técnica de dropout genera que el número de iteraciones necesarias para converger se duplique.

Este modelo de AlexNet fue entrenado durante aproximadamente 90 épocas a través del conjunto de entrenamiento de 1.2 millones de imágenes, que tomó de cinco a seis días en finalizar, utilizando dos GPU NVIDIA GTX 580 de 3 GB [4].

3.5. Otras arquitecturas convolucionales disponibles

Se presentarán otras arquitecturas importantes para la clasificación de imágenes. Estos modelos son redes neuronales profundas que se caracterizan por sus distintas innovaciones que aportaron a la hora de mejorar los resultados en el concurso ILSVRC [19].

3.5.1. VggNet

Luego de ver los impresionantes resultados obtenidos por AlexNet, los investigadores de aquel momento se dedicaron a buscar alternativas para aplicar e innovar sus arquitecturas. Todavía quedaba un campo amplio para introducir mejoras innovadoras e intuían que el error se podía disminuir en gran medida.

Había distintas posibilidades para investigar. Por un lado se estaba comentando que disminuir el tamaño de los filtros era beneficioso. También se podrían probar otros métodos de entrenamiento. Simonyan y Zisserman [21] decidieron ahondar en la profundidad, añadiendo más capas convolucionales y dió como resultado la red convolucional bautizada como VggNet.

Diseñada por Karen Simonyan y Andrew Zisserman de la Universidad de Oxford, fue ganadora en la tarea de localización en el concurso ILSVRC en 2014 y obtuvo el segundo puesto en la tarea de clasificación en el mismo concurso. Sus tasas de error fueron *top-1* de 37,5 % y *top-5* de 17,0 %. Se utilizan dos versiones: VGG19 (con 19 capas) y VGG16 (con 16 capas).

3.5.1.1. Arquitectura

Las entradas a esta arquitectura son datos que en el caso del paper original son imágenes RGB normalizadas a 224×224 píxeles. El preprocesamiento consta simplemente de extraer el valor RGB medio a cada píxel. Luego, en las capas de convolución, se le aplican filtros muy pequeños, de 3×3. Es el tamaño más pequeño que se puede utilizar sin perder las relaciones vecinales entre píxeles (izquierda/derecha, arriba/abajo y centro).

La idea principal era utilizar varios filtros pequeños, apilando capas de convolución. Entonces la red terminaba resultando más profunda que las existentes anteriormente, pero más eficiente. Los diseñadores probaron numerosas variantes en la arquitectura. En este sentido, el paper publicado [21] profundiza en las versiones de 11, 13, 16 y 19 capas con pesos, las cuales se encuentran en la Figura 3.7. Las que dieron una mejor performance y las más utilizadas son las versiones de 16 y 19 capas. En la siguiente tabla extraída del paper se ven todas las configuraciones probadas, y las dos últimas columnas son las que, en efecto, fueron utilizadas para aplicaciones reales.

Para entender las razones de utilizar varias capas convolucionales apiladas en lugar de una sola, se debe pensar en que una pila de dos capas convolucionales con filtros de 3×3 (sin un pooling espacial en el medio) tiene un campo receptivo efectivo de 5×5. Siguiendo con este análisis, tres de estas capas apiladas conforman uno de 7×7 [21]. La ganancia está en que, por un lado se incorporan tres rectificaciones no lineales (ReLU) en lugar de una sola. Por otro, se disminuye el número de parámetros utilizado con los filtros más pequeños. Tres filtros de 3×3, con C número de canales, tendrán $3 \times (3^2C^2) = 27C^2$ parámetros, en oposición con los que requeriría un único filtro de 7×7, $(7^2C^2) = 49C^2$, un 81 % más. Esta reducción se traduce directamente en un mayor ahorro de memoria con la ventaja de obtener el mismo campo receptivo.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig. 3.7: Configuraciones VggNet (columnas). En negrita se destacan los cambios entre cada configuración. Notación conv<tamaño del campo receptivo - numero de canales> [21, Tabla 1]

3.5.1.2. Entrenamiento

Los diseñadores de VggNet se basaron en los procedimientos utilizados por Krizhevsky [19] cuando entrenó AlexNet. El tamaño del lote se estableció en 256 y el momento a 0.9.

Se aplicó la técnica de dropout con una relación de deserción establecida en 0.5. La tasa de aprendizaje se estableció inicialmente en 10^{-2}, y luego disminuyó en un factor de 10 cuando la validación se estancaba. En total, la tasa de aprendizaje disminuyó 3 veces, y el aprendizaje se detuvo después de 370K iteraciones (74 épocas). A pesar del mayor número de parámetros y la mayor profundidad de esta redes en comparación con Krizhevsky [19], se requirieron menos épocas para converger debido a la regularización implícita impuesta por una mayor profundidad sumado a menores tamaños de filtros de convolución y la pre-inicialización de ciertas capas.

Los primeros valores de los pesos de la red son importantes, ya que una mala inicialización puede detener el aprendizaje debido a la inestabilidad del gradiente en capas profundas [22]. Para sortear este problema, se comenzó con el entrenamiento de la configuración A (Figura 3.7) eligiendo números aleatorios. Luego, al tomar las arquitecturas más profundas, se inicializan las primeras cuatro capas convolucionales y las últimas tres (las FC) con las capas de la red A (las capas intermedias se inicializaron aleatoriamente). Esto no disminuye la tasa de aprendizaje de las capas pre-inicializadas, lo que les permite cambiar durante el aprendizaje.

Para obtener las imágenes de entrada de 224×224, se recortaron aleatoriamente imágenes dispuestas en el set de entrenamiento. Para aumentar aún más el conjunto, los recortes se sometieron a un cambio horizontal aleatorio y un cambio de color aleatorio RGB.

Las redes en la Figura 3.7 fueron entrenadas en un sistema equipado con cuatro GPU NVIDIA Titan Black, entrenar una sola red tomó 2-3 semanas dependiendo de la arquitectura. Luego, fueron puestas a prueba individualmente y se obtuvieron las menores tasas de error con las variantes D y E de la Figura 3.7.

Para el concurso [19] utilizaron una combinación de todas las posibilidades, promediando la respuesta de cada una. Así lograron ganar el segundo puesto en la tarea de clasificación, con un 6,8 % de error *top-5*, en comparación con el 6,67 % que obtuvo la arquitectura ganadora, y un 7,1 % en localización.

Para finalizar, no se quiere dejar de mencionar que luego de haber ganado en la categoría de localización, se descubrió que al promediar únicamente las configuraciones D y E el error disminuye a 7,0 %.

3.5.2. GoogLeNet

Mientras que en Oxford se desarrollaba VggNet, en la Universidad de Michigan se estaba gestando otra arquitectura diferente pero basada en las mismas premisas: ir más profundo. El modelo GoogLeNet supera en número de capas a todos los enfoques previos, sumando un total de 22 (veintidós).

Inception es un nombre utilizado para referirse a GoogLeNet [23]. Esta segunda denominación es basada en el módulo especial que se agregó en la nueva red bautizado de esa manera. El objetivo principal de esta a red era crear una herramienta innovadora, que mejore los resultados que existían hasta el momento y que lo haga de forma eficiente, aprovechando los recursos computacionales al máximo.

3.5.2.1. Arquitectura

Al diseñar una red convolucional, la dificultad radica en la selección de las capas de convolución: se debe decidir tanto el tamaño de los filtros de convolución, la cantidad y el orden en que serán colocados a lo largo de la red. Los diseñadores de GoogLeNet pensaron

en una idea innovadora, decidieron colocar varias convoluciones en paralelo para que la red decidiera por sí misma cuál era la conveniente en cada etapa del proceso. Estas operaciones simultáneas se agrupan en bloques los cuales se denominaron **Inception**. Apilando varios de estos elementos lograron la arquitectura conocida como GoogLeNet.

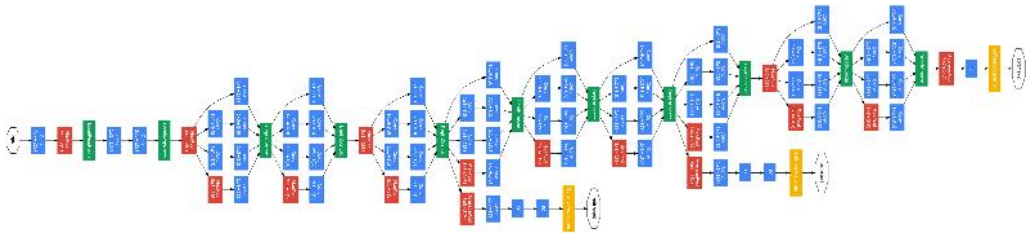


Fig. 3.8: Arquitectura GoogLeNet [23, Figura 3].

3.5.2.2. Módulo Inception

Los filtros de convolución seleccionados tienen las siguientes dimensiones 1×1, 3×3 y 5×5. Los tamaños fueron elegidos por conveniencia. Al finalizar las convoluciones en paralelo se ejecuta la concatenación de todos los resultados, unificando el resultado en un único tensor de salida.

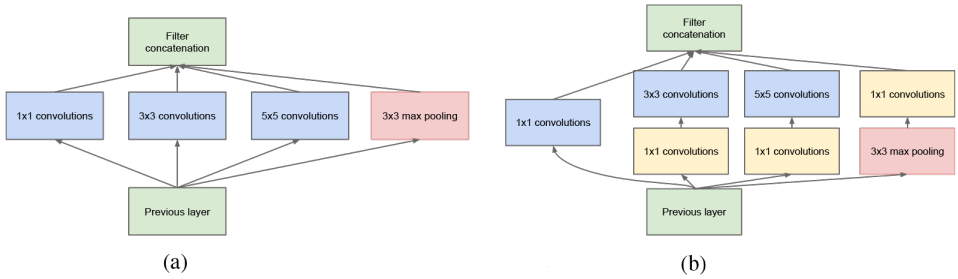


Fig. 3.9: A la izquierda la versión del módulo Inception sin la reducción de las dimensiones espaciales. A la derecha se le agregan estos elementos, las convoluciones 1×1 [23, Figura 2]

El problema con el bloque dispuesto como se ve en la Figura 3.9-a es que una única convolución 5×5 en una capa intermedia de la arquitectura tiene un costo computacionalmente alto, considerando que las entradas en esas fases suelen presentar un gran volumen de datos.

Por ejemplo, pensemos que se tiene una entrada de $28 \times 28 \times 256$. El volumen luego de la convolución y posterior concatenación es de $28 \times 28 \times 672$. Es ineficiente realizar este procedimiento, sobre todo considerando que se trabajará con un gran número de capas.

La solución es el esquema de la Figura 3.9-b, en donde se reducen dimensiones con la ayuda de convoluciones 1×1. Los investigadores justificaron esta decisión sosteniendo que incluso al trabajar disminuyendo las dimensiones, los bloques Inception pueden contener mucha información sobre un fragmento de la imagen relativamente grande.

La red consta de 22 (veintidós) capas de profundidad, haciendo referencia a la cantidad de bloques Inception y contando solo capas con parámetros (o 27 capas si también se tiene en cuenta las capas de pooling). El número total de capas (bloques de construcción independientes) utilizados para la construcción de la red es de aproximadamente 100.

Otro dato de interés es que se quitaron la mayoría de las capas totalmente conectadas al final de la red, dejando una sola. Estas son las que más parámetros utilizan y descubrieron que al intercambiar estas últimas por un pooling promediado mejoraban la exactitud *top-1* en un 0,6 %. Se mantenía el dropout ya que resulta esencial para no caer en el overfitting.

Si se observa detenidamente la arquitectura de GoogLeNet, se observa que de algunos módulos salen ramificaciones con capas FC. La razón para dicha disposición es que al ser la red tan extensa, al calcular los gradientes y estos propagarse a lo largo de la red, terminaban dando resultados muy bajos, tanto que no iban a tener fuerza de decisión. Entonces, basándose en las arquitecturas previas, mucho más escasas en números de capas, consideraron que los módulos intermedios ya podrían discriminar de manera apropiada. Se añadieron capas FC en tempranos lugares de la red, que participaban de los cálculos de retropropagación del error, y contribuyen a combatir el problema del desvanecimimiento del gradiente que ocurre cuando el gradiente toma valores cercanos a cero. Esto se da mientras el gradiente se retropropaga a las primeras capas, donde la multiplicación repetitiva puede reducir demasiado al gradiente o saturarse en valores muy mínimos.

3.5.2.3. Entrenamiento

Las redes GoogLeNet fueron entrenadas usando el sistema de aprendizaje automático distribuido DistBelief, utilizando una implementación basada en CPU. El entrenamiento utilizó la técnica de Descenso de Gradiente Estocástico asíncrono con 0,9 de momentum, una tasa de aprendizaje fija (disminuyendo la tasa de aprendizaje en un 4 % cada 8 épocas).

Para el concurso ILSVRC en el 2014, entrenaron siete modelos iguales de GoogleNet y utilizaron como predicción un ensamble de todas las salidas. Esto les permitió ganar el primer puesto en la tarea de clasificación, obteniendo un error *top-5* de 6,67 %.

3.5.3. ResNet

En el contexto de seguir haciendo crecer la profundidad de las redes, surge ResNet [15] o red residual que rompió todos los esquemas ya que alcanzó la suma de 152 (ciento cincuenta y dos) capas. La idea perseguida era demostrar que se podría crear una red eficiente sin tener la necesidad de hacer convoluciones en paralelo, lo que era muy complejo computacionalmente. El paradigma ResNet permitió por primera vez entrenar redes muy profundas de más de 100 (cien) capas y aún así obtener buenos resultados. Esta arquitectura marcó un punto de inflexión, no sólo porque dio mejores resultados que las anteriores arquitecturas, sino también presentó menores tiempos de entrenamiento reduciendo considerablemente el número de parámetros.

Las redes residuales fueron propuestas por Kaiming en 2015 y se desempeñaron excelentemente en el ImageNet Large Scale Visual Recognition Challenge de ese año, ganando en todas las categorías [4]. Una red neuronal residual es una Red Neuronal Artificial que toma el concepto de las neuronas piramidales ubicadas en la corteza cerebral, las cuales se conectan con neuronas en capas no necesariamente contiguas, saltando capas intermedias. Las redes neuronales residuales típicas son implementadas con dos o tres saltos que contienen la no-linealidad (ReLU).

3.5.3.1. Arquitectura

Al comenzar a pensar en la problemática principal de hacer una red más profunda, los autores probaron agregando capas a una CNN "plana" como VggNet, pero se dieron cuenta que el rendimiento de la misma no mejoraba, sino que empeoraba.

Un motivo para saltarse capas dentro de una RNA es para evitar el problema de desvanecimiento del gradiente. La omisión de capas simplifica la red de manera efectiva, usando menos capas en las primeras etapas de entrenamiento. Luego la red gradualmente recupera las capas omitidas cuando aprende los mapas de características posteriormente.

ResNet propone que en lugar de tratar de aprender a partir de mapeos subyacentes de x y $F(x)$, es posible aprender la diferencia entre los dos, que es el "residuo" o mapeo residual $H(x) = F(x)+x$ y, posteriormente, ajustar el último a la entrada, como se observa en la Figura 3.10.

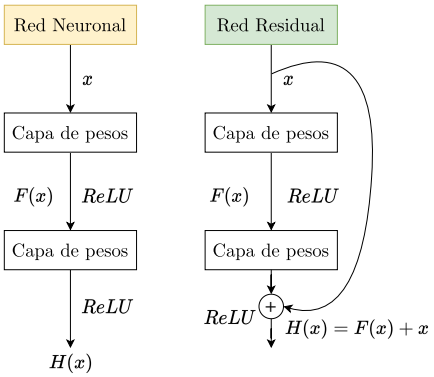


Fig. 3.10: Ahora la red intenta aprender de $F(x) + x$. El problema de entrenar redes neuronales muy profundas fue alivianado con la introducción de una nueva capa: el bloque residual.

Las redes neuronales profundas son más difíciles de entrenar a medida que aumenta, justamente, la profundidad y es por ello que se re-formulan las capas como funciones de aprendizaje residual con referencia a la capa de entrada, en vez de aprender funciones sin referenciar.

El diseño en cuanto a la implementación del modelo es similar a VggNet. Se cumplió el objetivo de construir una red simple y elegante. Las convoluciones son de 3×3, se reduce a la mitad el tamaño espacial (pooling). No hay capas FC, excepto el clasificador al final y no fue necesario utilizar la técnica de dropout. Todo esto se puede observar en la Figura 3.11.

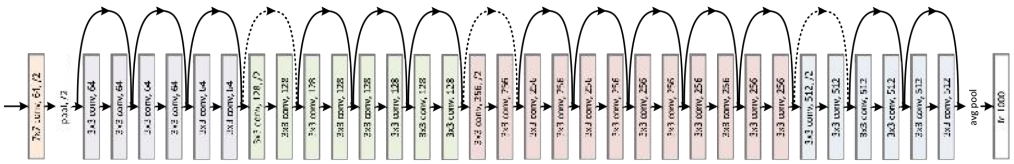


Fig. 3.11: Arquitectura ResNet [15, Figura 3]

ResNet de 152 capas logró el primer puesto en todas las categorías del ILSVRC en el año 2015. Específicamente alcanzó un *top-1* error de 19,38 % y *top-5* error del 3,57 % en

la tarea de clasificación [15]. Este error supera al ojo humano.

3.6. Transfer Learning

La técnica de *transfer learning* hace referencia a la situación donde lo aprendido por una red, por ejemplo una distribución P_1, es utilizado y explotado para mejorar la capacidad de generalización en otra configuración, un ejemplo la distribución P_2. Entonces se puede asumir que muchos de los factores que representan las variaciones del conjunto P_1 son relevantes y están presentes en P_2. Esta nueva herramienta consiste en explotar el conocimiento adquirido por la técnica del Machine Learning tradicional y se esquematiza la Figura 3.12.

Si hay significativamente más datos en el primer escenario P_1, entonces eso puede ayudar a aprender representaciones que son útiles para generalizar rápidamente a partir de muy pocos ejemplos extraídos de P_2. Esto es de mucha ayuda debido a que numerosas categorías visuales comparten nociones de bajo nivel de bordes y formas visuales, los efectos de cambios geométricos, cambios en la iluminación, entre otras [24].

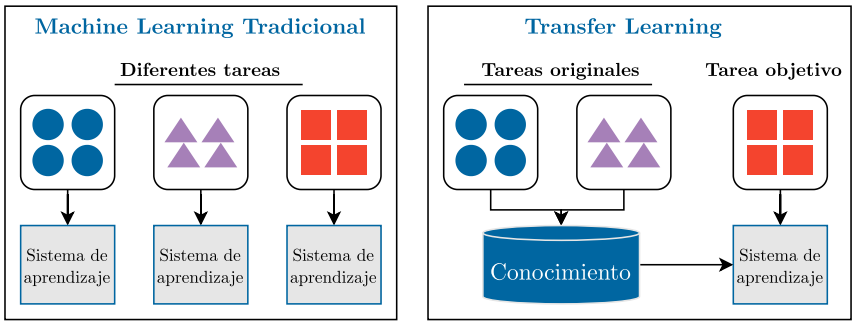


Fig. 3.12: Comportamiento de la técnica de Machine Learning tradicional vs. Transfer Learning.

A grandes rasgos hay dos formas de adaptar una red pre-entrenada de convolución a un problema en particular [11]:

1. Volcar las imágenes a la CNN sin modificarla y tomar las salidas de la penúltima capa, que representan los **mapas de características** para entrenar un nuevo clasificador que se adapte al nuevo problema.

2. Usar las imágenes de entrenamiento congelando, o no, los pesos de algunas de las capas de la red original, haciendo de esta manera **fine-tuning** para la tarea de interés.

La primera solución es sencilla y no requiere de ningún esfuerzo extra, excepto por la necesidad de diseñar el nuevo clasificador. La segunda solución es más prometedora ya que permite lograr una adaptación más profunda haciendo un uso de los datos de interés, y así explotando todo el potencial de las redes convolucionales [11].

Cuando se realiza fine-tuning, se debe decidir qué capas de la red original quedan congeladas y cuales podrán seguir entrenando, es decir, adaptando sus pesos con los datos de entrenamiento de la nueva distribución P_2. Estas elecciones impactan tanto en la precisión, performance y tiempo de entrenamiento, dependiendo mucho de la similitud entre el problema original y el objetivo.

Generalmente las primeras capas suelen congelarse, debido a que en ellas se encuentran las características de bajo nivel que suelen funcionar bien para muchos tipos de problemas [11].

Capítulo 4

Entrenamiento de Redes Convolucionales

Para entrenar una red de aprendizaje profundo, como son las CNN, explicadas en el capítulo anterior, hay cuestiones que son específicas y claves para obtener un buen rendimiento. En este capítulo analizamos las cuestiones a tener en cuenta a la hora de trabajar en el proceso de entrenamiento de este tipo de redes.

4.1. Hiperparámetros

Las investigaciones en el área del aprendizaje de máquina se enfocan en el desarrollo de métodos capaces de capturar ciertos elementos de interés de un conjunto de datos. Con esos elementos se pueden obtener desde estructuras propias de los datos (clustering o agrupamiento) hasta la habilidad de predecir ciertos valores a partir de características dadas, las cuales pueden ser de tipo discretas (clasificación) o continuas (regresión).

En la actualidad existen una gran variedad de métodos de aprendizaje, entre los más populares los modelos inspirados en las redes neuronales biológicas y los basados en filtros. Un rasgo en común que tienen estos métodos es que son parametrizados por un **conjunto de hiperparámetros**, los cuales deben de ser configurados de manera apropiada por el usuario para maximizar la usabilidad del aprendizaje requerido.

Estos hiperparámetros ajustan numerosos aspectos de los algoritmos de aprendizaje en uso, y afectan directamente a los resultados del modelo y su rendimiento [25].

Los hiperparámetros son variables cuyos valores son establecidos antes de que el proceso de entrenamiento comience. En contraste, el valor de otros parámetros libres como los pesos que componen las distintas conexiones de todas las neuronas del modelo, son obtenidos en el mismo entrenamiento. Entonces, para los parámetros, la forma más habitual de que estos sean estimados es mediante algoritmos de optimización, pero no ocurre lo mismo para los hiperparámetros, los cuales deben de ser determinados de manera externa al algoritmo de aprendizaje en sí.

4.1.1. Clasificación de hiperparámetros

Las CNNs tienen más hiperparámetros para establecer que una RNA. Estos hiperparámetros afectan al tamaño de los mapas de características [26]. Particularmente para las distintas redes de convolución, existen varios hiperparámetros a tener en cuenta, los cuales se pueden dividir en dos tipos [27]:

- Los hiperparámetros que determinan la estructura y arquitectura de la red:

- Tamaño del filtro.
- Tipo de filtro: valores propios del filtro.
- Stride: cantidad de pasos que el filtro pasará sobre la imagen de entrada.
- Padding: agrega capa de ceros para asegurar que el filtro pueda pasar por los bordes de la imagen.
- Número de capas ocultas.
- Funciones de activación: permite al modelo aprender predicciones no lineales.

- Los hiperparámetros que determinan el entrenamiento de la red como la tasa de aprendizaje, el momentum, el número de épocas y el tamaño de lote.

Describiremos los hiperparámetros más importantes, relevantes para nuestro trabajo.

4.1.2. Tasa de aprendizaje

En el proceso de entrenamiento de un modelo de red neuronal artificial se busca ajustar los pesos y biases, es decir, los parámetros libres del modelo, para minimizar el error medido por la función de costo propuesta. El ajuste de los pesos se realiza de acuerdo a los errores cometidos por la red durante la presentación de cada patrón. Luego haciendo uso del algoritmo de Descenso de Gradiente Estocástico, por sus siglas en ingles *stochastic gradient descent*, la corrección aplicada a los pesos sinápticos que conectan la neurona i a la j se definen con la regla delta [13]:

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n)$$

donde $\Delta w_{ji}(n)$ es la corrección de pesos a aplicar, η es el parámetro de tasa de aprendizaje, $\delta_j(n)$ es el gradiente local calculado como el producto entre la señal de error y la derivada asociada a la función de activación. $y_i(n)$ es la señal de entrada a la neurona j, es decir, la salida de la neurona i.

Cuanto más pequeño sea el parámetro de tasa de aprendizaje η, más pequeño serán los cambios a realizar en los pesos sinápticos en la red de una iteración a la siguiente. Esta mejora está atada al costo de un entrenamiento más lento en tiempo.

Si el valor de la tasa de aprendizaje es demasiado grande, en el proceso de entrenamiento cada actualización se realizará a pasos enormes, lo cual puede ser bueno si se desea ir rápido en el proceso de aprendizaje. La desventaja de esto es que se puede saltar a un mínimo local de la función de costo y quedar estancado en el fondo del pozo, buscando el siguiente punto rebotando al azar y no llegar al mínimo global. Ademas un valor de η muy grande puede hacer que la red se vuelva inestable.

En la Figura 4.1 se puede ver graficados tres casos con distintos tamaños de tasa de aprendizaje y el comportamiento de la función de costo en función del ajuste de los pesos.

En cuanto a la implementación de este hiperparámetro, en general una buena opción suele ser aquella que disminuye la tasa de aprendizaje a medida que el modelo se acerca a una solución, es decir, a medida que transcurren las épocas de entrenamiento. La idea es lograr al principio del entrenamiento avances de aprendizaje más rápido y luego ir reduciendo la tasa para lograr ajustes más pequeños y facilitar la convergencia del proceso de entrenamiento al mínimo de la función de costo.

Dentro de las heurísticas propuestas por Haykin [13] para lograr un mejor rendimiento en el algoritmo de retropropagación del error se menciona las tasas de aprendizaje. Todas

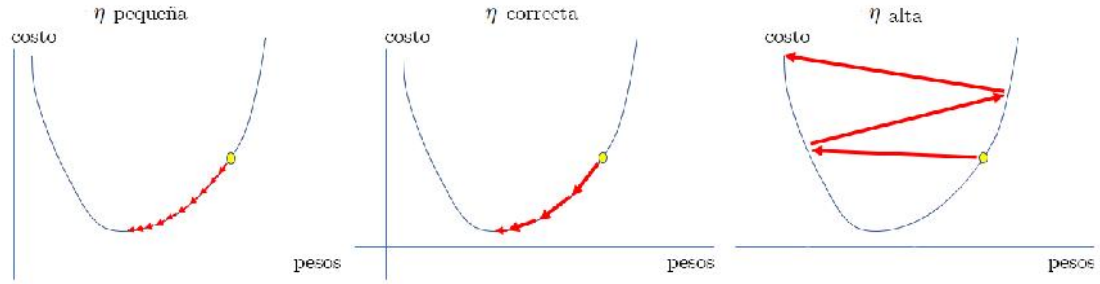


Fig. 4.1: Función de costo vs. pesos del modelo, en tres casos con distintos tamaños de tasa de aprendizaje.

las neuronas en el perceptrón multicapa deberían idealmente aprender a la misma velocidad. Las últimas capas suelen tener gradientes locales que las capas al comienzo de la red. Es por esto que el parámetro η de tasa de aprendizaje debería ser un valor pequeño para las últimas capas. En el caso de neuronas con muchas entradas, estas deberían tener una tasa más pequeña de aprendizaje que neuronas con pocas entradas.

4.1.3. Momentum

El momentum es otro hiperparámetro a considerar a la hora de entrenar un modelo de red convolucional. Se usa para la ponderación de gradientes e indica cuánto del peso anterior será retenido en el nuevo cálculo.

Es una manera simple de incrementar la tasa de aprendizaje evitando el peligro de inestabilidad. Esto se logra modificando la regla delta mencionada anteriormente incluyendo el término de momentum:

$$\Delta w_{ji}(n) = \alpha \cdot \Delta w_{ji}(n) + \eta \cdot \delta_j(n) \cdot y_i(n)$$

donde η es la tasa de aprendizaje y α es la constante momentum. Estos parámetros pueden tomar valores en el intervalo [0,1].

4.1.4. Número de épocas

Durante el proceso de entrenamiento hay que determinar el momento de detención del mismo. Asociado a esto existe el hiperparámetro de número de épocas, el cual indica la cantidad de iteraciones en el conjunto de entrenamiento entero que se realizará en este proceso.

La duración del entrenamiento está asociado a dos problemas que pueden llegar a darse: un entrenamiento insuficiente donde el rendimiento de la red es pobre, o un overfitting donde la red se encuentra sobre-entrenada y existe una pérdida de la capacidad de generalización.

El perceptrón multicapa que es entrenado con el algoritmo de retropropagación del error, aprende por etapas a medida que las sesiones de entrenamiento continúan. Con el objetivo de obtener una buena generalización, es difícil saber cuándo es el mejor momento para detener el entrenamiento si miramos a la curva de aprendizaje de entrenamiento por sí sola. Como mencionamos anteriormente la red puede llegar a sobreentrenarse si no se detiene al proceso en el momento justo.

Una forma de identificar al overfitting es incorporando un conjunto de validación, obtenido a partir del conjunto de entrenamiento original. El primer subconjunto de muestras es usado para entrenar la red pero cada cierta cantidad de épocas, el entrenamiento se detiene, se congelan los pesos y biases, y se calcula el error sobre el conjunto de validación. Este error se utiliza como indicador para detener el entrenamiento si se observa que el mismo comienza a aumentar. Luego de que la fase de validación está completada, el entrenamiento continua por otro periodo y el proceso se repite hasta que el error de validación supere cierto valor limite.

Este procedimiento es conocido como el método de detección temprana de entrenamiento (*early stopping*) el cual es sencillo de entender y muy utilizado en la práctica. Una representación gráfica de este método se puede apreciar en la Figura 4.2.

Típicamente, el modelo no da tan buenos resultados en el subconjunto de validación como en el de entrenamiento. Cómo se puede observar en la Figura 4.2, el error de entrenamiento decrece de manera monótona a medida que transcurren las épocas. En contraste, el error sobre el conjunto de validación decrece hasta un mínimo y luego continuado el entrenamiento, comienza a incrementar. Cuando observamos la curva de entrenamiento, parece que el modelo puede mejorar corriendo aún más épocas, pero el crecimiento del error sobre el conjunto de validación mientras que el error de entrenamiento decrece está indicando una situación de overfitting. Esta heurística sugiere que el punto mínimo de la curva de validación sea utilizado como un criterio para detener la etapa de entrenamiento [13].

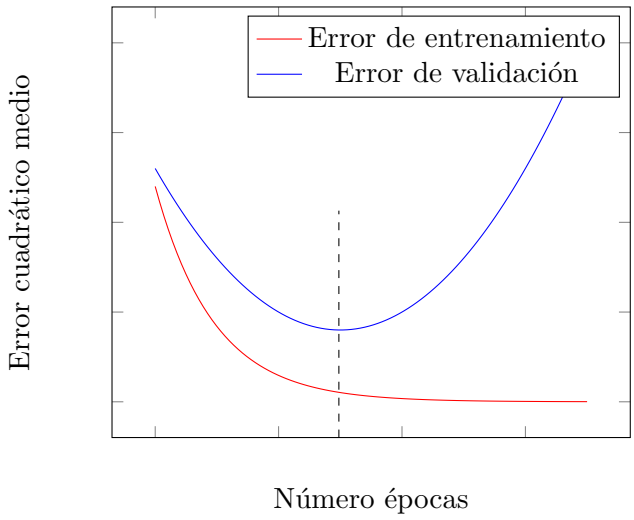


Fig. 4.2: Regla de detención temprana basada en la validación cruzada. La línea punteada indica el momento de parada.

4.1.5. Tamaño del lote

Un hiperparámetro a tener en cuenta es el tamaño de lote. Su valor define qué tipo de entrenamiento se lleva a cabo, determinando el modo en que la red procesa las distintas muestras o señales de nuestro conjunto de datos [28]:

- En el entrenamiento por Descenso de Gradiente Estocástico (SGD), también conocido cómo en línea, se utiliza de una muestra a la vez para entrenar al modelo,

donde se hace una corrida, se calcula el error y se realiza la retropropagación. Esto quiere decir que se actualizan los pesos sinápticos después de cada muestra del conjunto de entrenamiento. El entrenamiento en línea es superior al entrenamiento por lote o Batch, ya que en general permite sortear los mínimos locales y acelerar la convergencia.

- Con el entrenamiento por lote los pesos se actualizan sólo luego de pasar todas las muestras del conjunto de entrenamiento. El entrenamiento por lote minimiza directamente el error total; sin embargo, el entrenamiento por lotes puede obligar a actualizar muchas veces las ponderaciones hasta que se cumpla alguna de las reglas de parada y por tanto pueden ser necesarias muchas lecturas de datos.
- En el entrenamiento por mini lotes las muestras del conjunto de datos se dividen en grupos de tamaño similar y el entrenamiento hace uso de la información de este grupo de muestras. El entrenamiento por mini lotes ofrece una solución intermedia entre el entrenamiento por lotes y en línea.

Hoy en día se suele usar un tamaño de lote mayor a uno con valores recomendados de: 2, 4, 8, 16, 32, 64, 128, 256. Un tamaño de lote grande permite aprovechar el poder computacional disponible al utilizar la multiplicación de matrices en los cálculos de entrenamiento. Sin embargo, esto requiere de mayor memoria. Un tamaño de lote más pequeño induce más ruido en los cálculos de error lo cual puede ayudar a no caer en un mínimo local. Un valor estándar suele ser de tamaño treinta y dos.

4.2. Weight Decay

En el área del Deep Learning existe un problema central que es cómo hacer que un algoritmo funcione bien no sólo con los datos del conjunto de entrenamiento, sino también con muestras inéditas para el modelo, es decir, del conjunto de prueba. Muchas estrategias utilizadas en el aprendizaje automático están diseñadas explícitamente para reducir el error de prueba a expensas de un mayor error en el proceso de entrenamiento. Estas estrategias se conocen como regularización [24].

Weight decay, o regularización L2, es de las técnicas de regularización más utilizadas en modelos de aprendizaje automático, y es aplicable a los pesos de una red convolucional. En particular para la librería PyTorch, se aplica tanto a los pesos como a los biases. Esta técnica de regularización agrega una pequeña penalización, generalmente la norma L2 de todos los pesos del modelo, a la función de costo, logrando de esa manera un acercamiento de los pesos sinápticos y biases al origen.

La técnica de weight decay suele utilizarse para evitar el problema de overfitting, manteniendo los pesos en valores mínimos y así sortear un crecimiento fuera de control de los mismos y, por lo tanto, la explosión del gradiente. De esta manera los pesos que se deben simplemente al ruido de un subconjunto de datos pequeño en particular, no se recuperarán tan fácilmente, y los pesos finales del modelo se ajustan a la mayor cantidad de datos y por menor medida al ruido.

4.3. Técnicas de optimización y exploración de hiperparámetros

El objetivo de la exploración de los hiperparámetros es buscar entre diversas configuraciones de hiperparámetros hasta dar con la que tenga como resultado un rendimiento

óptimo. Normalmente, el proceso de exploración de hiperparámetros es un trabajo manual muy laborioso, dado que el espacio de búsqueda es muy extenso y la evaluación de cada configuración puede ser costosa [29].

Los modelos de redes convolucionales como los estudiados pueden tener más de diez hiperparámetros, y encontrar la mejor combinación puede ser visto como un problema de búsqueda. La elección apropiada de estos parámetros juega un rol fundamental en el éxito del modelo convolucional a entrenar.

El número de hiperparámetros para un modelo de aprendizaje suele ser pequeño, menor a cinco, pero puede alcanzar hasta cientos para algoritmos de aprendizaje más complejos. Ha sido demostrado empíricamente que en muchos casos sólo un par de hiperparámetros impactan de manera significativa al rendimiento del modelo, sin embargo identificar a los relevantes es difícil [25].

El proceso de encontrar el hiperparámetro óptimo en el área de aprendizaje de máquina se conoce como optimización de hiperparámetros. Los algoritmos de aprendizaje de máquina como las redes neuronales y las redes de aprendizaje profundo, para las tareas de clasificación de señales, involucran a un número de hiperparámetros mencionados anteriormente que tiene que establecerse para poder correrlos y hacer uso de estos mismos.

A fin de seleccionar los valores apropiados de hiperparámetros para un conjunto de datos específico de antemano, los usuarios pueden recurrir a valores por defecto especificados en las distintas herramientas y plataformas de software, o configurarlas de forma manual, por ejemplo basándose en recomendación de la literatura, experiencia o prueba y error [30].

La búsqueda de hiperparámetros es generalmente realizada de forma manual ajustándose a valores previamente utilizados en otros trabajos con buenos resultados. Otra manera es haciendo uso de técnicas de optimización como la búsqueda en cuadrícula o la búsqueda aleatoria [31].

Entre los algoritmos más conocidos se encuentra la búsqueda en cuadrícula o barrido de parámetros y es el método tradicional para realizar una optimización de hiperparámetros. Es una búsqueda exhaustiva que funciona por fuerza bruta a través de un subconjunto del espacio de hiperparámetros especificado de forma manual, y busca sobre cada combinación posible de hiperparámetros. Este algoritmo debe de estar guiado por alguna métrica de rendimiento, como suele ser la medición de validación cruzada en el conjunto de entrenamiento.

Otro algoritmo de optimización es la búsqueda aleatoria la cual utiliza combinaciones aleatorias de hiperparámetros. Esto significa que no se prueban todos los valores.La búsqueda aleatoria muestrea valores seleccionados aleatoriamente del dominio de los hiperparámetros. Se generaliza bien en espacios continuos y mixtos. Es necesario definir un mecanismo de terminación, normalmente mediante el establecimiento de un número de iteraciones o estableciendo un límite objetivo a conseguir.

La búsqueda aleatoria al igual que la búsqueda en cuadrícula permite paralelismo teniendo en cuenta la independencia del hiperparámetro. Ambas búsquedas comparten la maldición de la dimensionalidad, es decir, la búsqueda se vuelve impráctica cuando el número de hiperparámetros es lo suficientemente grande.

Algunas tareas describen espacios de búsqueda altamente complejos donde la existencia de un hiperparámetro se condiciona por el valor de otros. Un ejemplo simple seria en la optimización de la arquitectura de una red neuronal, donde el número de capas ocultas es un hiperparámetro y el tamaño de cada capa induce a la creación de un conjunto adicional

de hiperparámetros que depende del número de capas. En nuestro caso en un principio no nos enfrentamos a este tipo de problema de optimización de arquitectura del modelo ya que utilizaremos un modelo pre-entrenado.

4.3.1. Computación natural

En la búsqueda de encontrar los valores óptimos para configurar los hiperparámetros relacionados al entrenamiento de un modelo convolucional aparece el área de la computación natural, y particularmente el área de la inteligencia de enjambre.

Existen una gran cantidad de problemas que pueden ser resueltos mediante el uso de ciertos algoritmos del área del aprendizaje de máquina pero que requieren de un alto costo para su resolución. Este costo está asociado a las variables de tiempo y espacio, ocurriendo que al intentar disminuir cualquiera de los dos, la otra variable incrementa de forma exponencial. Entonces existe una búsqueda de nuevos modelos alternativos que rompan con estas limitaciones que poseen los modelos convencionales.

La computación natural surge como una alternativa a la computación clásica, en la búsqueda de nuevos paradigmas que puedan proporcionar una solución efectiva. Esta disciplina trata de capturar la forma en que la naturaleza lleva a cabo procesos de cálculo desde hace millones de años y engloba un conjunto de modelos que tienen como característica común la simulación del modo en que la naturaleza actúa sobre la materia.

La Computación Natural estudia la forma en que las diversas leyes de la naturaleza producen modificaciones en determinados sistemas (desde hábitats hasta conjuntos de moléculas, pasando por organismos vivos) que pueden ser interpretados como procesos de cálculo sobre sus elementos [32].

Distintos ejemplos a mencionar son: la Red Neuronal Artificial (McCulloch, Pitts, 1943) inspirada en el cerebro, los algoritmos genéticos (Holland, 1975) con inspiración en el ADN y la computación con membranas (Gh Paun, 1998) basada en el concepto de células.

Una rama de la computacion natural que es importante mencionar como opción a los métodos de búsqueda y optimización es la inteligencia de enjambres. La inteligencia de enjambre es un conjunto de tecnicas que estudia el comportamiento colectivo de los sistemas descentralizados, autoorganizados, naturales o artificiales. El concepto se emplea en los trabajos sobre Inteligencia Artificial (IA). La expresión fue introducida por Gerardo Beni y Wang Jing en 1989, en el contexto de los sistemas robóticos móviles [33].

Inspirados por la naturaleza, especialmente por ciertos sistemas biológicos, los sistemas de inteligencia de enjambre están típicamente formados por una población de agentes simples que interactúan localmente entre ellos y con su medio ambiente. Los agentes siguen reglas simples y, aunque no existe una estructura de control centralizado que dictamine el comportamiento de cada uno de ellos, las interacciones locales entre los agentes conduce a la emergencia de un comportamiento global complejo. Como ejemplos naturales se incluyen las colonias de hormigas, el alineamiento de las aves en vuelo, el comportamiento de rebaños durante el pastoreo y el crecimiento bacteriano [34].

En el área de la Inteligencia Computacional se habla de la Optimización por Enjambre de Partículas (PSO o *particle swarm optimization*) haciendo referencia a una heurística inspirada en el comportamiento de algunas partículas en la naturaleza. Los métodos de optimización por enjambre de partículas se atribuyen originalmente a los investigadores Kennedy, Eberhart y Shi [35].

PSO permite optimizar un problema a partir de una población de soluciones candidatas, denotadas como "partículas", moviendo éstas por todo el espacio de búsqueda según

reglas matemáticas que tienen en cuenta la posición y la velocidad de las partículas. El movimiento de cada partícula se ve influido por su mejor posición local hallada hasta el momento, así como por las mejores posiciones globales encontradas por otras partículas a medida que recorren el espacio de búsqueda. El fundamento teórico de esto es hacer que la nube de partículas converge rápidamente hacia las mejores soluciones. PSO es una metaheurística, ya que asume pocas o ninguna hipótesis sobre el problema a optimizar y puede aplicarse en grandes espacios de soluciones candidatas. Sin embargo, como toda metaheurística, PSO no garantiza la obtención de una solución óptima en todos los casos.

4.4. Validación cruzada

La validación cruzada (CV) es un procedimiento estadístico utilizado en el ámbito del ML para cuantificar la solidez y habilidad de predicción de un modelo sobre un conjunto de datos que no se ha utilizado en la etapa de construcción o entrenamiento del mismo.

Se utiliza principalmente en entornos en los que el objetivo es la predicción, y se desea estimar la precisión con la que un modelo funcionará en la práctica. También se usa comúnmente para comparar y seleccionar un modelo para un problema de modelado predictivo dado que es fácil de entender, implementar y da como resultado estimaciones de habilidades que generalmente tienen un sesgo menor que otros métodos.

En un problema de predicción, un modelo generalmente recibe un conjunto de datos en los que se ejecuta el entrenamiento (conjunto de datos de entrenamiento) y un conjunto de datos de datos desconocidos vistos por primera vez con los que se prueba el modelo (conjunto de datos de validación o de prueba). El objetivo de la validación cruzada es probar la capacidad del modelo para predecir nuevos datos que no se utilizaron para estimarlo, con el fin de señalar problemas como el sobreajuste (*overfitting*) y dar una idea de cómo el modelo generaliza a un conjunto de datos independientes.

En una ronda de cualquier técnica de validación cruzada se dividirá cada muestra de datos en subconjuntos complementarios. Inmediatamente se realiza el análisis en un subconjunto y se valida el análisis en el otro subconjunto. Luego para reducir la variabilidad, en la mayoría de los métodos se realizan múltiples rondas utilizando diferentes particiones, combinando los resultados de validación, por ejemplo, haciendo uso del promedio.

Es importante que cualquier preparación de los datos antes de ajustar el modelo ocurra en el conjunto de datos de entrenamiento asignado por la validación cruzada dentro del ciclo en lugar de en el conjunto de datos más amplio, aplicando esto también a cualquier ajuste de hiperparámetros. Si no se realizan estas operaciones dentro del ciclo, se pueden producir fugas de datos y una estimación optimista de la habilidad del modelo invalidando los resultados.

La validación cruzada de k-fold es probablemente la más popular entre las estrategias de CV, sin embargo existen otras opciones que se detallaran a continuación.

4.4.1. Validación cruzada de k iteraciones

La validación cruzada es un método muy utilizado para [36]:

- La estimación del modelo aprendido a partir de los datos disponibles utilizando un algoritmo. En otras palabras, para medir la generalización de un algoritmo.
- Comparar el rendimiento de dos o más algoritmos diferentes y descubrir el mejor

enfoque para los datos disponibles o, alternativamente, para comparar el rendimiento de dos o más variantes de un modelo parametrizado.

En la validación cruzada, dividimos el conjunto de datos D en dos particiones, es decir, conjunto de entrenamiento denotado por T y conjunto de prueba denotado por R donde la unión de estos dos subconjuntos es el conjunto de datos completo y la intersección de ellos es el conjunto vacío [37] :

$$T \cup R = D \ , \ \ T \cap R = \oslash$$

El subconjunto T se utiliza para entrenar al modelo. Después de que el modelo es entrenado, el R se usa para probar el rendimiento del modelo.

En K-fold cross validation, dividimos aleatoriamente el dataset D en K particiones $D_1, \ldots, D_k$ en donde todas tienen idealmente el mismo tamaño y

$$\bigcup_{i=1}^{k} D_i = D \quad D_i \cap D_j = \oslash \ para\ todo\ i, j \in \{1, ..., K\}\ ,\ i \neq j$$

El algoritmo de validación cruzada incluye k iteraciones y dentro de cada iteración, una de las k-particiones es usada como conjunto de prueba y el resto de los datos son utilizados para entrenamiento. La estimación del error es el promedio de los errores de test de cada iteración. Es importante que se cumpla la siguiente condición: $T \cap R = \oslash$, caso contrario se estará introduciendo una parte o la totalidad de los elementos de test al modelo para que las aprenda. En el mundo del machine learning esto es hacer trampa.

Al momento de elección del modelo a utilizar, existen parámetros del mismo, como se menciono anteriormente en la sección 4.1, que deben de ser determinados de antemano y que se conocen como *hiperparámetros*. En la búsqueda de estos parámetros, dividimos el conjunto total D en tres subconjuntos, conjunto de entrenamiento T, conjunto de prueba R y conjunto de validación V. Por lo general, el conjunto de entrenamiento cuenta con más elementos que los subconjuntos R y V. El objetivo es hallar un conjunto de parámetros con buenos resultados, entonces se utiliza a T para entrenar al modelo con los valores seleccionados. Para cada valor de hiperparámetro (o hiperparámetros) se entrena al modelo y luego se obtiene su rendimiento con el conjunto de validación. Una vez que esto se realiza para todos los valores deseados a probar, finalmente el hiperparámetro de valor óptimo seleccionado es aquel cuyo resultado de estimación es el mejor para el conjunto de validación V.

Una vez determinados los hiperparámetros de la red, el modelo se entrena usando el conjunto de entrenamiento (donde se usa el valor del parámetro encontrado). Luego, el modelo se prueba en el conjunto de prueba R y la estimación de su rendimiento es el promedio de errores del conjunto de pruebas en cada etapa de la validación cruzada (los errores de prueba de los k_i subconjuntos utilizado como test set en las k iteraciones).

En la validación cruzada con el conjunto de validación, tenemos:

$$T \cap R = \oslash, \ \ T \cap V = \oslash, \ \ V \cap R = \oslash$$

Los subconjuntos de validación y prueba no deben superponerse, es decir que no deben existir muestras de validación que también se encuentren en el conjunto de prueba. De esta manera los parámetros del modelo no deben decidirse sobre el subconjunto de prueba, y es así que al tener $V \cap R \neq \oslash$ entonces los resultados obtenidos no serán representativos.

Desde ya, entrenar un algoritmo y evaluar su performance sobre el mismo conjunto de datos resulta en resultados muy optimistas. La validación cruzada nació para resolver este problema y su premisa fundamental es que una buena estimación de la performance del modelo surge al evaluar la salida del algoritmo sobre datos que no se utilizan para entrenamiento. De los datos disponibles, se aparta un subconjunto para evaluar la performance. Estas muestras de validación pueden tomar el rol de nuevos datos siempre y cuando sean variables aleatorias independientes e idénticamente distribuidas (i.i.d.) [38].

Para realizar mediciones independientes de la performance de un algoritmo uno debe asegurar que los factores que afectan las mediciones son independientes en cada corrida. Estos factores son [36]:

- Los datos de entrenamiento de los cuales el algoritmo aprende.
- Los datos de prueba con los cuales se mide la performance del algoritmo.

Si existen datos del conjunto de prueba R que aparecen en más de una ronda del proceso iterativo, los resultados obtenidos en estas rondas son dependientes y por lo tanto no es valido una comparación estadística. No sólo los conjuntos de datos deberán de ser independientes, tampoco debería existir ningún tipo de superposición de los datos utilizados para validar y entrenar en una misma ronda. Un algoritmo tendrá mejores resultados sobre los datos que haya visto durante la fase de aprendizaje que aquellos desconocidos. Por esta razón una superposición entre los datos de entrenamiento y de validación puede terminar en una sobre-estimación en la medida de performance.

El principal interés de la validación cruzada radica en la universalidad de la heurística de división de datos. Sólo supone que los datos están distribuidos de manera idéntica, y que las muestras de entrenamiento y validación son independientes, lo que incluso puede ser flexibilizado [36]. Por lo tanto, la validación cruzada se puede aplicar a (casi) cualquier algoritmo en (casi) cualquier marco, como la regresión, la estimación de densidad y la clasificación entre muchos otros.

A menudo se observa en la literatura que el procedimiento adoptado es tomar el dataset, particionarlo en tres subsets, entrenar con el subconjunto de entrenamiento, cortar el entrenamiento según el error de validación y evaluar la performance en el conjunto de prueba. Muchas veces este método es el preferido por ser computacionalmente más eficiente (es un único entrenamiento).

Hay otro punto importante también para justificar optar por esta relativa sencillez, en el deep learning se suelen disponer de conjuntos de datos notablemente grandes, escenarios donde no hay que preocuparse por la alta varianza de los datos. Es decir, la sensibilidad de las estimaciones no está tan fuertemente ligada a cómo se dividen los subconjuntos de entrenamiento, validación y test y por eso se pueden tomar estas libertades [39]. Sin embargo para nuestros casos debido a que los conjuntos de datos son relativamente pequeños no pudimos tomar esa libertad.

Finalmente el procedimiento general puede ser resumido como el siguiente:

1. Se divide el conjunto de datos en k grupos.
2. Para cada grupo único:
   a) Se toma un grupo como un conjunto de datos de prueba o de validación.
   b) Se toman los k - 1 grupos restantes como un conjunto de datos de entrenamiento.

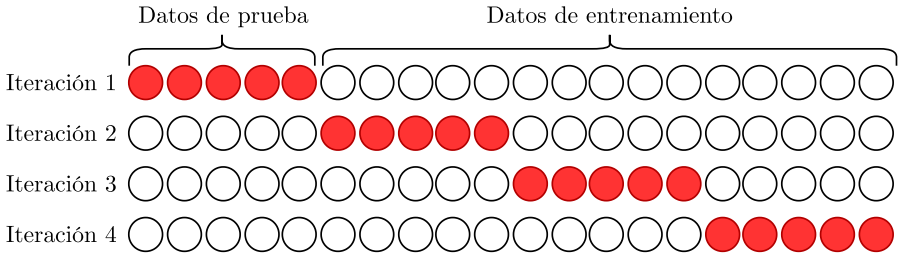


Fig. 4.3: Para una iteración dada, haciendo uso de validación k-fold con valor $k = 4$, el subconjunto marcado en rojo es usado para validar el modelo entrenado con los datos que quedan.

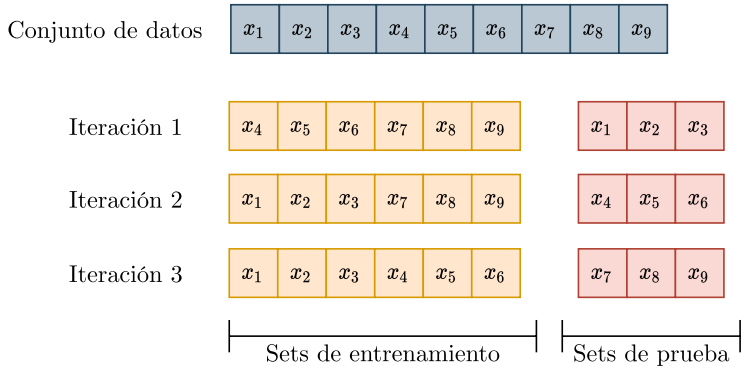


Fig. 4.4: Validación k-fold con valor $k = 3$.

c) Se utiliza el conjunto de entrenamiento para entrenar al modelo y se evalúa utilizando el conjunto de prueba.

3. Se almacena la puntuación de la evaluación y se descarta el modelo.

4. Se promedian los resultados de los modelos.

El tamaño de k debe seleccionarse de manera cuidadosa debido a que un valor mal elegido puede resultar en una mala idea de la verdadera habilidad actual del modelo, como un resultado de precisión con una alta varianza, es decir, que cambia demasiado según los datos seleccionados, o con un alto sesgo, teniendo una sobreestimación de las capacidades del modelo. Existe una compensación de sesgo-varianza asociada con la elección de k en la validación cruzada de k-fold. En tanto que k aumenta, la diferencia de tamaño entre el conjunto de entrenamiento y los subconjuntos de remuestreo se reduce. Luego a medida que esta diferencia disminuye, el sesgo de la técnica se reduce [40].

Una táctica común para seleccionar el valor de k es de modo que cada grupo de muestras de datos sea lo suficientemente grande como para ser estadísticamente representativo del conjunto total de datos. Otra táctica es tomar de $k = 10$, un valor que se ha encontrado a través de la experimentación que generalmente resulta en una estimación de la habilidad del modelo con un sesgo bajo y una varianza modesta. Entonces la elección de k suele ser 5 ó 10, pero no existe una regla formal aunque se ha demostrado empíricamente que estos valores producen estimaciones de la tasa de error de prueba que no sufren ni un sesgo excesivamente alto ni una varianza muy alta [41].

La ventaja de este método sobre otros es que todas las observaciones se utilizan tanto para el entrenamiento como para la validación, y cada observación se utiliza para la

validación exactamente una vez.

4.4.2. Montecarlo o validación repetida de submuestreo aleatorio (MCCV)

Algunos autores han señalado que el método de validación cruzada, a pesar de su gran popularidad, sufre de ciertos defectos. B. Efron [42] señala que la técnica de validación cruzada no es un buen candidato para estimar el error de predicción y P. Zhang [43] sostiene que un modelo bajo el criterio de cross validation es idóneo para sufrir overfitting. Por estas razones se optó por utilizar una alternativa conocida como Montecarlo Cross Validation (MCCV), también llamada como Shuffle Split o Random Permutation CV [44].

En contraposición a la validación cruzada, la estrategia de MCCV crea múltiples divisiones de los datos que conformarán los sets de entrenamiento, validación y testeo, de manera aleatoria con la posibilidad de que exista repeticiones en estos conjunto para distintas iteraciones. Se asume que cada partición producida podrá contener muestras que en otra iteración pertenecieron a una partición diferente, es decir, cada división de MCCV es independiente de cualquier otra.

La proporción de los datos que pertenecen a cada subconjunto y la cantidad de repeticiones son controladas por el equipo desarrollador. El incremento del número de iteraciones tiene un efecto decreciente sobre la incertidumbre en la estimación de la performance. Por ejemplo, para tener una visión global de un modelo serían adecuadas unos 25 ciclos siempre y cuando el usuario esté dispuesto a aceptar cierta inestabilidad en los resultados. Si se desea obtener estimaciones estables de la performance, se sugiere escoger un número grande de repeticiones, entre 50 y 200. Cuanto más grandes sean los sets, más repeticiones se necesitarán para reducir la incerteza en los resultados [40].

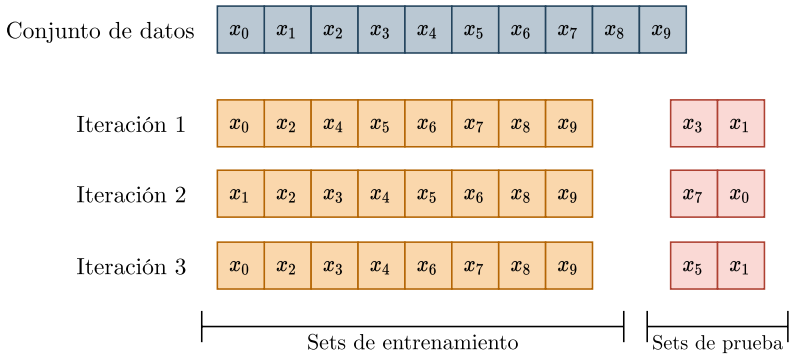


Fig. 4.5: Validación cruzada Montecarlo con un valor de repetición = 3.

La ventaja de este método sobre la validación cruzada de k veces es que la proporción de la división de entrenamiento y validación no depende del número de iteraciones, es decir, el número de particiones. En [45] sostienen que esta técnica es más confiable que la validación cruzada descrita en la sección anterior ya que, a pesar de que en su estudio ambos métodos dieron errores similares, la desviación estándar de la aplicación de MCCV es más pequeña que aquella en CV. Por esta razón, a la hora de evaluar el modelo seleccionado en nuestro estudio se utilizó Montecarlo Cross Validation en las implementaciones finales de ambos conjuntos de datos.

La desventaja de este método es que es posible que algunas observaciones nunca se seleccionen en el subconjunto de validación, mientras que otras muestras pueden seleccio-

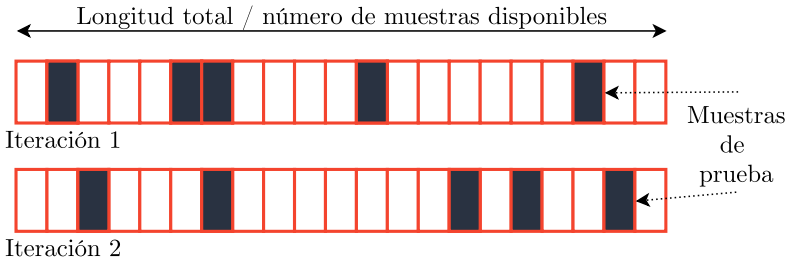


Fig. 4.6: Validación cruzada Montecarlo con un valor de repetición = 2 donde se puede observar la aleatoriedad del método en cuanto a la elección de muestras para formar los conjuntos.

narse más de una vez. En otras palabras, los subconjuntos de validación pueden superponerse.

4.5. Matriz de confusión

La estimación de la precisión de clasificación (*classification accuracy*) puede llevarse a cabo haciendo uso de la matriz de confusión. La matriz de confusión es una buena opción para reportar los resultados de un problema de clasificación de M clases debido a que permite observar la relación entre las salidas del clasificador y las verdaderas. Una matriz de confusión [46] contiene información acerca de las clasificaciones producidas y actuales realizadas por un sistema de clasificación. El rendimiento de estos sistemas es evaluado haciendo uso de los datos de la matriz. En el campo de la Inteligencia Artificial (IA) una matriz de confusión es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo las diferentes clases o resultados de la clasificación.

- Verdadero positivo (VP) es la cantidad de positivos que fueron clasificados correctamente como positivos por el modelo.
- Verdadero negativo (VN) es la cantidad de negativos que fueron clasificados correctamente como negativos por el modelo.
- Falso negativo (FN) es la cantidad de positivos que fueron clasificados incorrectamente como negativos. Error tipo 2.
- Falso positivo (FP) es la cantidad de negativos que fueron clasificados incorrectamente como positivos. Error tipo 1.

La precisión (*accuracy*) se refiere a lo cerca que está el resultado de una medición del valor verdadero. Es la proporción de correctos del número total de predicciones. Se determina haciendo uso de la ecuación:

$$acc = \frac{VP + VN}{VP + VN + FP + FN}$$

4.6. Optimizadores

En la búsqueda del mínimo global, los algoritmos de optimización suelen utilizar la estrategia más clásica para obtener el gradiente conocida como Descenso de Gradiente Estocástico (SGD), mencionado en la sección 4.1.5. Como esto es computacionalmente ineficiente se ha optado por trabajar con otras variantes del mismo.

4.6.1. Descenso del Gradiente de Mini Lotes y Descenso del Gradiente Estocástico

La diferencia entre el Descenso del Gradiente de Mini Lotes y el Descenso de Gradiente Estocástico, ambos desarrollados en la sección 4.1.5, radica en el momento en el que se hace la actualización de pesos. El nuevo enfoque, en lugar de usar la suma de todos los errores de entrenamiento, como el SGD, usa el gradiente de error de una sola muestra en cada iteración [5]. Una iteración consiste en el tamaño de lote: la cantidad de muestras presentadas a la red antes de realizar el ajuste. Este tamaño es un parámetro adicional a configurar. A menudo se utiliza el término descenso del gradiente estocástico en la literatura para referirse al descenso del gradiente por lotes. Bottou [47] observó que el descenso por gradiente estocástico muestra un buen rendimiento para problemas a gran escala. El SGD es el bloque utilizado por muchos algoritmos de optimización que aplican alguna variación para lograr mejores tasas de convergencia (por ejemplo, [48][49]).

Kingma y Ba [50] observaron que SGD tiene una gran importancia práctica en muchos campos de la ciencia y la ingeniería y debido a ello propusieron un nuevo optimizador llamado Adam, un método para la optimización estocástica eficiente. Sin embargo, Wilson et al. [51] informó que las soluciones encontradas por métodos adaptativos (como Adam) tienen una generalización peor que SGD, aunque las soluciones encontradas por métodos de optimización adaptativa tienen un mejor desempeño en el conjunto de entrenamiento. Ruder [52] recomienda utilizar Adam como la mejor opción de optimización general.

4.6.2. Adam

Adam [50] es un algoritmo de optimización que se puede utilizar en lugar del procedimiento clásico de descenso de gradiente estocástico para actualizar los pesos de red de forma iterativa en función de los datos de entrenamiento. Adam fue presentado por Diederik Kingma de OpenAI y Jimmy Ba de la Universidad de Toronto en su artículo ICLR 2015 titulado *"Adam: Un método para la optimización estocástica"*. El nombre Adam se deriva de la estimación adaptativa del momento.

Adam es un algoritmo popular en el campo del aprendizaje profundo porque logra buenos resultados rápidamente. Además los resultados empíricos demuestran que Adam funciona bien en la práctica y se compara favorablemente con otros métodos de optimización estocástica. En ocasiones la utilización de este optimizador mejora los resultados frente al optimizador del descenso estocástico del gradiente [53][54].

Al presentar el algoritmo, los autores enumeran los beneficios atractivos de usar Adam en problemas de optimización no convexos, de la siguiente manera:

- Fácil de implementar.
- Computacionalmente eficiente con pequeños requisitos de memoria.

- Invariante al cambio de escala diagonal de los gradientes.
- Muy adecuado para problemas que son grandes en términos de datos y/o parámetros.
- Apropiado para objetivos no estacionarios.
- Apropiado para problemas con gradientes muy ruidosos o escasos.
- Los hiperparámetros tienen una interpretación intuitiva y generalmente requieren poca sintonización.

Adam es diferente al Descenso de Gradiente Estocástico clásico debido a que SGD mantiene una única tasa de aprendizaje (denominada alfa) para todas las actualizaciones de peso y la tasa de aprendizaje no cambia durante el entrenamiento. En cambio, con Adam se mantiene una tasa de aprendizaje para cada peso de la red (parámetro) y se adapta por separado a medida que se desarrolla el proceso de aprendizaje. El método calcula las tasas de aprendizaje adaptativo individuales a partir de estimaciones del primer y segundo momento de los gradientes.

Los autores describen a Adam como una técnica que combina ventajas de otras técnicas basadas en el descenso de gradiente estocástico como el algoritmo de gradiente adaptativo (**AdaGrad**[55]), el cual mantiene una tasa de aprendizaje por parámetro que mejora el rendimiento en problemas con gradientes escasos, y la técnica de propagación de raíz cuadrada media (**RMSProp**[56]), que mantiene tasas de aprendizaje por parámetro que se adaptan en función del promedio de magnitudes recientes de los gradientes para el peso (por ejemplo, qué tan rápido está cambiando). En lugar de adaptar las tasas de aprendizaje de los parámetros en función del primer momento promedio (la media) como en RMSProp, Adam también hace uso del promedio de los segundos momentos de los gradientes (la varianza no centrada).

En nuestras implementaciones finales para ambas arquitecturas hemos utilizado el optimizador Adam.

Capítulo 5

Plataformas de Hardware y Software para Aprendizaje Profundo

5.1. Introducción

La Inteligencia Artificial (IA) es una ciencia que no deja de deslumbrarnos por todos los desarrollos de los últimos años. No obstante, esta situación no siempre fue así. En la jerga científica se ha aceptado el término de inviernos de la IA. Ellos refieren a aquellas épocas en donde los modelos no alcanzaron las expectativas puestas en los mismos y, por tanto, se dejó de apostar en este mundo. Lo cual generó baches en el camino innovador del aprendizaje automático [57].

Una de las razones por las cuales se presentaba esta desoladora situación era que no existían los avances en la ciencia y tecnología necesarios para sustentar a los esquemas de la IA. Con el paso del tiempo y con las innovaciones modernas, se han ideado técnicas como la computación de alta performance, que trae aparejado un hardware robusto y potente, para soportar las grandes necesidades de cómputo paralelo que requieren estas metodologías. Hoy en día ha habido un enorme progreso en el diseño de las arquitecturas computacionales que han permitido que complejos esquemas del Machine Learning (ML) sean aplicados a la resolución de problemas difíciles del mundo real, utilizando esquemas de programación paralela. Se ha realizado un esfuerzo significativo para emplear tecnologías de Computación de Alto Rendimiento (HPC) en marcos de Big Data y sistemas de aprendizaje profundo que están altamente enfocados en la confiabilidad y el rendimiento [58].

5.2. Deep Learning y HPC

Los puntos clave a tener en cuenta al construir un modelo de aprendizaje profundo son la precisión, el consumo de energía, el rendimiento/latencia y el costo. En los esquemas del ML, la precisión del algoritmo debe medirse en un conjunto de datos lo suficientemente grande. Hay muchos conjuntos de datos disponibles públicamente de uso amplio que los investigadores pueden usar en sus proyectos. En nuestra implementación, por ejemplo, se utilizarán dos repositorios de datos de dominio público. La programabilidad también es un aspecto fundamental, ya que los parámetros deben ser actualizados frecuentemente. Estos dos ítems mencionados, la alta dimensionalidad y la necesidad de programabilidad implican altísimos niveles de procesamiento conjunto a numerosos movimientos de datos. Esto es un desafío en cuanto a la energía y eficiencia dado que el movimiento de datos cuesta más que el procesamiento de los mismos [59]. El rendimiento también depende de

la cantidad de cálculos paralelos que un sistema pueda realizar, los cuales crecen conforme aumenta la dimensionalidad de los datos. Finalmente, el almacenamiento es costoso y muy necesario a la hora de trabajar con modelos de DL. Todos los elementos mencionados son los que hay que tener en cuenta a la hora de comenzar con un proyecto. Una de las primeras decisiones es si el sistema utilizado será *on-premise* (en físico) u *on cloud* (en la nube) [60]. Es decir, si se va a adquirir los elementos necesarios en físico o si se emplearán servicios en la nube. Estas son decisiones de diseño que se basan en las posibilidades y magnitudes del sistema a implementar.

Independientemente de las disposiciones de operación, si se va a poseer del hardware en físico o no, siempre se deberá recaer sobre dispositivos robustos y que sean capaces de soportar el nivel de operaciones con el que se trabaja en los modelos de aprendizaje profundo. En este sentido, deben satisfacer los requerimientos de la Computación de Alto Rendimiento. La misma es cada vez más importante como herramienta de investigación en diferentes áreas. Particularmente para el DL y el entrenamiento de modelos convolucionales, esta herramienta ofrece beneficios y ventajas únicas [61]:

1. Grandes cantidades de operaciones de punto flotante: el entrenamiento de redes neuronales requiere una gran cantidad de álgebra lineal y operaciones matemáticas que requieren números decimales, como las operaciones de punto flotante. La HPC admite el rendimiento de punto flotante.

2. Baja latencia: al entrenar redes neuronales en arquitecturas de servidor tradicionales, puede causar problemas debido a las demoras entre servidores. HPC se centra en lograr interconexiones de gran ancho de banda, logrando así una baja latencia.

3. Entrada y salida en paralelo: las redes neuronales utilizan sistemas que pueden proporcionar capacidades de entrada y salida en paralelo. HPC puede realizar múltiples operaciones de entrada/salida al mismo tiempo con alto rendimiento.

Conociendo las necesidades de estos sistemas, en los siguientes apartados se comenzará con un recuento histórico explicando brevemente la evolución de las computadoras. Seguidamente se enfocará en explicar los temas relacionados al paralelismo, sus desafíos y esquemas actuales.

5.3. Evolución de las computadoras

5.3.1. Primera generación: tubos de vacío

En el contexto de la Segunda Guerra Mundial se necesitaban hacer operaciones que eran plasmadas en las tablas de tiro de las piezas de artillería y armas, las cuales llevaban mucho tiempo de elaboración manual. El proyecto que dió respuesta a esta necesidad tuvo el nombre de ENIAC (*Electronic Numerical Integrator and Computer*), y fue diseñado y construido en la Universidad de Pennsylvania bajo la supervisión de John Mauchly y John Presper Eckert. Esta máquina terminó siendo enorme, pesando treinta toneladas, ocupando 15.000 pies cuadrados y conteniendo más de 18000 tubos de vacío. El ENIAC era decimal, no binario. Su memoria consistía en veinte acumuladores, cada uno capaz de obtener un número decimal de diez dígitos. Cada dígito estaba representado por diez tubos de vacío. Uno de los mayores inconvenientes del ENIAC era que tenía que ser programado manualmente mediante conmutadores y conectando y desconectando cables.

Lamentablemente, su construcción se terminó para 1946 y era muy tarde para utilizarlo en la guerra, por lo que se aplicó a una función distinta demostrando su naturaleza de propósito general [62].

Para solucionar la tediosa tarea de cargar y modificar programas, presente en el ENIAC, se comenzó a trabajar en el concepto del programa-almacenado. Uno de los asesores del proyecto anterior, Jhon von Neumann fue el principal contribuyente al desarrollo de esta idea. La misma consiste en representar el programa de una forma adecuada, para que pueda ser guardado en la memoria junto con los datos. De esta manera, la computadora es capaz de conseguir las instrucciones leyéndolas de la memoria. En 1945 se publica la propuesta de una nueva computadora llamada EDVAC (*Electronic Discrete Variable Computer*).

En 1952 se concreta el diseño del IAS en el Instituto para Estudios Avanzados, de la mano de von Neumann y sus colegas. Esta computadora es el prototipo de toda una serie de computadoras de propósito general, conformada de una memoria principal, una unidad aritmético lógica, una unidad de control de programa y equipos de entrada y salida. Salvo raras excepciones, todas las computadoras de hoy en día tienen la misma estructura general y funcionamiento que la indicada en las máquinas von Neumann.

La memoria del IAS consistió en 1000 posiciones de almacenamiento llamadas palabras, de cuarenta dígitos binarios (bits) cada una. Tanto datos como instrucciones se almacenan ahí, por lo tanto, los números se pueden representar de forma binaria y las instrucciones también tienen un código binario. La unidad de control dirige al IAS captando instrucciones de la memoria y ejecutando una a una.

Durante los años 50, nacieron dos compañías, Sperry e IBM, que dominaron el mercado y comenzaron a fabricar computadoras comerciales. En 1947 Eckert y Mauchly formaron la Eckert-Mauchly Computer Corporation para fabricar computadoras con fines comerciales. Su primera máquina con éxito fue la UNIVAC I (*Universal Automatic Computer*). Estaba diseñada tanto para aplicaciones científicas como comerciales. A finales de los 50 salió al mercado el UNIVAC II, con más memoria y un mayor número de aplicaciones que su sucesor.

IBM, que era entonces el principal fabricante de equipos de procesamiento con tarjetas perforadas, sacó su primera computadora con programas almacenados electrónicamente, el 701 en 1953. Esta fue diseñada principalmente para aplicaciones científicas. En 1955 IBM presentó los productos 702, que tenían varias características hardware que lo hacían adecuado para aplicaciones de gestión. Dichos dispositivos fueron los primeros de una larga serie de computadoras 700/7000 que ayudaron a que IBM domine el mercado [63].

5.3.2. Segunda generación: transistores

El primer cambio importante en los computadores electrónicos vino con la sustitución de los tubos de vacío por transistores. El transistor es más pequeño, más barato, disipa menos calor y puede ser usado de la misma forma que un tubo de vacío en la construcción de computadoras. Mientras que un tubo de vacío requiere cables, placas de metal, una cápsula de cristal y vacío, el transistor es un dispositivo de estado sólido, hecho con silicio. Este dispositivo fue inventado en 1947 por los Laboratorios Bell, sin embargo, las computadoras completamente transistorizadas no estuvieron disponibles comercialmente hasta el final de los cincuenta. Los primeros en lanzar al mercado esta tecnología fueron NCR y RCA, luego los siguió IBM con la serie 7000 [63].

En la segunda generación también se introdujeron unidades lógicas y aritméticas y unidades de control más complejas, el uso de lenguajes de programación de alto nivel y se

proporcionó un software del sistema con la computadora.

En la segunda generación se destaca la aparición de la empresa Digital Equipment Corporation (DEC). DEC fue fundada en 1957 y en este año sacó su primera computadora, llamada PDP-1. Esta computadora y esta compañía iniciaron el desarrollo de los minicomputadores que fue de gran importancia en la tercera generación.

5.3.3. Tercera generación: circuitos integrados

A un transistor simple y autocontenido se le llama componente discreto. A través de los años cincuenta y principios de los sesenta, los equipos electrónicos estaban compuestos en su mayoría por componentes discretos (transistores, resistencias, capacitores, etc). Los componentes discretos se fabricaban separadamente, encapsulados en sus propios contenedores, y soldados o cableados juntos en tarjetas de circuitos en forma de panel, todo este proceso era caro y engorroso. En 1958 ocurrió un hito que revolucionó la electrónica: se inventó el circuito integrado. Este hecho define la tercera generación de computadoras y comenzó la era de la microelectrónica.

Como ya se mencionó los elementos básicos de una computadora deben proporcionar almacenamiento, procesamiento y control de funciones. Solo se requieren dos tipos fundamentales de componentes: puertas y celdas de memoria. Una puerta es un dispositivo que implementa una función lógica o booleana simple. La celda de memoria, por su parte, es un elemento que permite almacenar un dato o un bit. Interconectando muchos de estos dispositivos fundamentales, constituidos por componentes electrónicos simples, se puede construir una computadora.

Los circuitos integrados utilizaron el hecho de que componentes como transistores, resistencias y conductores podían ser fabricados a partir de un semiconductor como el silicio. En lugar de ensamblar componentes discretos hechos a partir de trozos de silicio separados, se fabrica el circuito entero en el trozo de silicio. Se pueden construir cientos incluso miles de transistores en una oblea de silicio. En la Figura 5.1 se muestran los elementos claves de un circuito integrado: una oblea de silicio se divide en una matriz de pequeñas áreas, cada una de unos pocos milímetros cuadrados. Se fabrica el mismo patrón de circuito en cada área y la oblea se divide en chips. Cada chip consiste en muchas puertas más una serie de puntos para conexiones de entrada y salida. El chip es encapsulado en una carcasa que lo protege y proporciona patas para conectar dispositivos fuera del chip.

Varios de estos elementos pueden ser interconectados en una tarjeta de circuito impreso para producir circuitos más complejos y más grandes. Inicialmente solo podían fabricarse y encapsularse juntas, con fiabilidad, unas pocas puertas o celdas de memoria. A estos primeros circuitos integrados se los conoció con el nombre de Integración a Baja Escala (SSI). A medida que fue pasando el tiempo se podían incorporar cada vez más componentes en un mismo chip.

Para 1964 IBM anunció el Sistema/360 que consistía en una familia de computadoras incompatibles a sus productos anteriores. Su objetivo era producir un sistema capaz de evolucionar conjunto a la nueva tecnología de circuitos integrados, el cual terminó siendo un éxito. Los distintos modelos de la familia eran compatibles en el sentido en que un programa escrito para un modelo debe poder ejecutarse en otro modelo de la serie, siendo la única diferencia el tiempo de ejecución. Entonces, las familias compartían un conjunto de instrucciones y un sistema operativo similar o idéntico, velocidad y memoria crecientes según el miembro que posea del Sistema/360 y, por supuesto, a mayores capacidades el costo se incrementa.

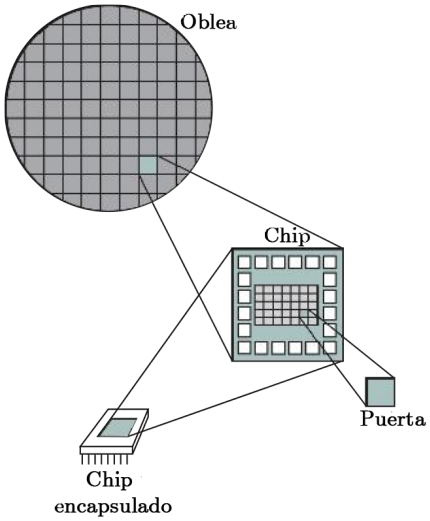


Fig. 5.1: Relación entre oblea, chip y puerta [63, Figura 1.1].

Contemporáneamente al primer Sistema/360, DEC lanzó el PDP-8, que a diferencia de la mayoría de las computadoras de la época que requerían una habitación con aire acondicionado, este era lo suficientemente pequeño para ser colocado en una mesa de laboratorio, además de que su costo no era tan elevado. La principal innovación de estas máquinas es que, en contraste con los productos de IBM, los modelos PDP-8 usaban una estructura que ahora es prácticamente universal para minicomponentes y microcomputadoras: la estructura del bus [63].

5.3.4. Ultimas generaciones

Luego de la tercera generación, no se ha llegado a un acuerdo general en las divisiones temporales entre las tecnologías existentes y esta evolución se ve reflejada en la linea temporal de la Figura 5.2. Con la Integración a Gran Escala (LSI) podía haber más de 1000 componentes en un simple chip de circuito integrado. Con la Integración a Escala Muy Grande (VLSI) se lograron más de 10.000 componentes por chip y los actuales chips de Integración a Escala Ultra Grande (ULSI) pueden contener más de un billón de elementos [63].

Con la rápida evolución de la tecnología, la tasa de introducción de nuevos productos y la importancia del software, el hardware y las comunicaciones del mundo moderno, la clasificación en generaciones es muy difusa. Los dos grandes desarrollos en las últimas generaciones incluyen la memoria semiconductora y los microprocesadores.

La primera aplicación de la tecnología de circuitos integrados fue utilizada para la construcción de procesadores (la unidad de control y la unidad aritmético-lógica) y luego se encontró que también puede utilizarse para fabricar memorias. Así nació la memoria semiconductora. En los años cincuenta y sesenta la mayoría de los dispositivos de almacenamiento se construían con pequeños anillos de material ferromagnético. Estos anillos de ferrita se insertan en mallas de finos cables engarzados en pequeños marcos dentro del computador. Se magnetizaba en un sentido el anillo (llamado núcleo) y representaba un uno; magnetizado en el otro sentido, representaba un cero. La memoria de ferrita era cara,

voluminosa y al leer su núcleo se borraban los datos almacenados en él. Era necesario disponer de circuitos que puedan recuperar el dato tan pronto como fue extraído.

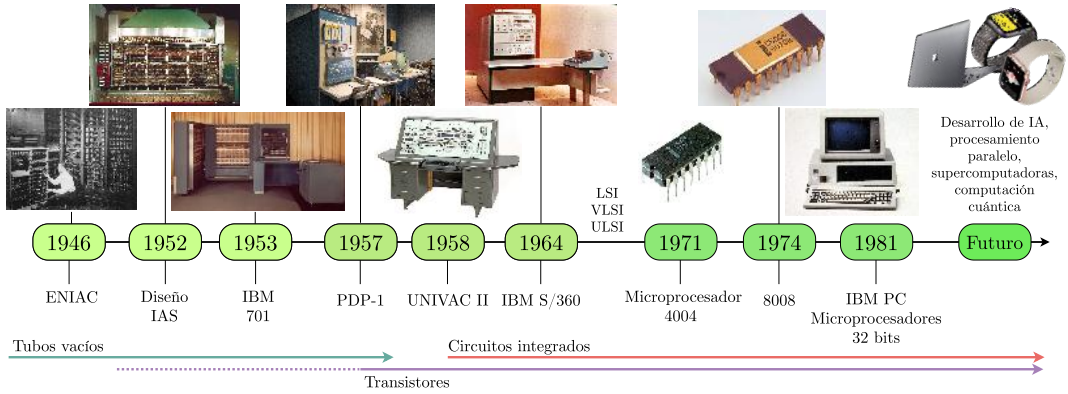


Fig. 5.2: Línea temporal de la evolución de las computadoras.

En 1970 Fairchild produjo la primera memoria semiconductora con 256 bits de memoria y tenía el tamaño de un núcleo de ferrita. Era no destructiva y más barata que un núcleo, aunque el costo por bit superaba al mismo. En 1974 el precio por bit de memoria semiconductora cayó por debajo del precio por bit de memoria de núcleo. Acompañado a este fenómeno, la densidad de memoria fue en aumento, lo que produjo que se construyan máquinas más pequeñas y rápidas con el mismo tamaño de memoria que máquinas más grandes y caras. El desarrollo de la tecnología de memorias junto con el desarrollo de la tecnología de procesadores cambiaron la naturaleza de las computadoras en menos de una década.

El aumento de la densidad de elementos en los chips de memoria se presentó también en los dispositivos de procesamiento. Conforme al paso del tiempo, cada vez se insertaban más y más elementos en cada chip. En 1971 Intel desarrolló su modelo 4004, el cual fue el primer chip que contenía todos los componentes de la CPU en un solo dispositivo. Con él nació el microprocesador. El 4004 hoy en día es muy primitivo, podía sumar dos números de cuatro bits y multiplicar sumas sucesivas. En 1972 se lanzó el Intel 8008 que era el doble de complejo que el anterior y trabajaba con 8 bits. Tanto el 8008 como el 4004 fueron diseñados para funciones específicas. En 1974 se lanza el primer microprocesador de propósito general, conocido como 8080, que también trabaja con 8 bits. Sobre la misma época se comenzaron a desarrollar los microprocesadores de 16 bits, los cuales no aparecieron hasta el final de los 70's. Luego, en 1981 los Laboratorios Bell y Hewlett-Packard desarrollaron microprocesadores de un solo chip de 32 bits.

5.4. Arquitectura de computadoras

La organización básica física de una computadora moderna se basa en la ya mencionada arquitectura von Neumann, propuesta en 1946. Este esquema consta de las siguientes unidades, las cuales se pueden estudiar en la Figura 5.3:

- Una memoria principal que almacena tanto datos como instrucciones.
- Una Unidad Aritmético-Lógica (ALU) capaz de hacer operaciones con datos binarios.

- Una unidad de control que interpreta las instrucciones en memoria y provoca su ejecución.
- Un equipo de entrada-salida dirigido por la unidad de control.

La Unidad de Procesamiento Central (CPU) comprime la unidad de control y la de aritmética lógica. El funcionamiento de una computadora es precisamente la ejecución de instrucciones para procesar datos por el CPU. Las instrucciones son las operaciones primitivas que el CPU ejecuta, como mover los contenidos de una locación en memoria (llamada registro) hacia otra locación dentro del CPU, o agregar los contenidos de dos registros del CPU. Las unidades de control recuperan los datos/instrucciones desde la memoria del sistema o memoria principal, también conocida como Memoria de Acceso Aleatorio (RAM). Los datos son luego procesados por la ALU secuencialmente, de acuerdo a las instrucciones decodificadas por la unidad de control. Almacenar tanto los datos como las instrucciones en la unidad de memoria principal es una característica esencial de la arquitectura von Neumann. Las unidades de entrada y salida actúan como la interfaz entre la computadora y el humano.

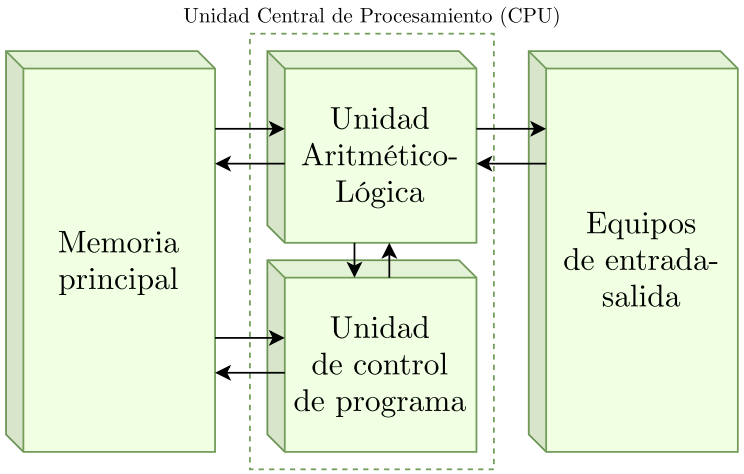


Fig. 5.3: Arquitectura Von Neumann [63].

No solo la CPU, la memoria del sistema juega un papel fundamental en la performance del sistema en general, y en el tiempo de ejecución de cada operación. La memoria de un sistema moderno es compleja. Un número de unidades más pequeñas y rápidas, llamadas memorias caché o simplemente cachés son colocadas entre la CPU y la memoria principal. Estas, que existen de varios niveles, conforman una jerarquía de almacenamiento en la que el tiempo de acceso y el tamaño aumentan a medida que están más alejadas físicamente de la propia CPU.

El objetivo de la memoria caché es colocar una parte del programa, la que es más probable que se utilice en los instantes siguientes, recuperándola de la memoria principal y colocándola en un lugar más cercano a la CPU. De esta manera, se acelera el proceso de acceso a los datos. La jerarquía de memoria (ver Figura 5.4), combina chips más pequeños y rápidos con otros más grandes y lentos pero más económicos, y permite que se pueda ver como todo un conjunto que representa una única memoria rápida y relativamente grande.

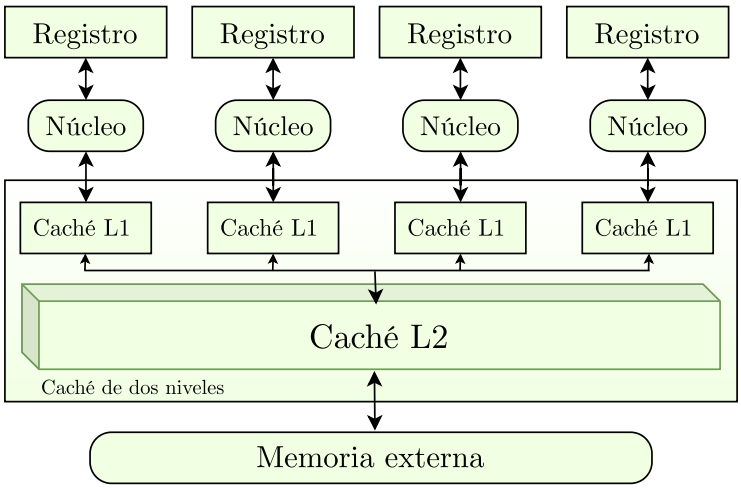


Fig. 5.4: En la imagen se observa una caché de dos niveles. L1 es mucho más pequeña que L2 pero también, notablemente más rápida en cuanto a acceso.

Una CPU moderna o microprocesador ejecuta (como mínimo) una instrucción por ciclo de reloj. Cada microprocesador posee su propio Conjunto de Instrucciones (ISA). Estas conforman el lenguaje que cada computadora comprende. Basados en el tipo de ISA, se distinguen dos arquitecturas diferentes de computadoras modernas, basadas en microprocesadores: las computadoras con un Conjunto de Instrucciones Reducido (RISC) y las computadoras con un Conjunto de Instrucciones Complejas (CISC) (ver Figura 5.5) [63].

La arquitectura CISC básica es esencialmente la arquitectura von Neumann desde el punto de vista de almacenar tanto las instrucciones como los datos dentro de una unidad común de memoria. La filosofía CISC consiste en que el ISA posee un gran número de instrucciones y modos de direccionamiento, con un número variado de ciclos de reloj requeridos en tiempo de ejecución. A su vez, ciertas instrucciones pueden ejecutar numerosas operaciones primitivas.

Por otro lado, la arquitectura básica RISC posee dos espacios completamente separados de almacenamiento para instrucciones y para datos. Esta es la característica introducida en la arquitectura Harvard para solucionar el cuello de botella en el esquema von Neumann debido a que los datos y las instrucciones compartían los mismos buses entre la CPU y la memoria. Estas arquitecturas RISC se basan en que el repertorio de instrucciones posean un número pequeño de operaciones primitivas para facilitar la manufacturación del hardware, lo que genera que cuando se deba ejecutar una operación compleja esta sea el conjunto de instrucciones simples.

Naturalmente las arquitecturas RISC son más rápidas y eficientes en comparación a las CISC. De todas formas, debido a la continua búsqueda de mejoras y mayor flexibilidad hoy en día se utiliza una combinación de ambos esquemas, las arquitecturas CISC exhiben ciertas características de las RISC y viceversa. Esto significa que es frecuente que las particularidades de cada uno se combinen. Ejemplos de arquitecturas clásicas CISC incluyen VAX (de DEC), PDP-11 (de DEC), Motorola 68000 (de Freescale/Motorola) y x86 (principalmente de Intel).

La arquitectura de CISC moderna, x86-64, que se encuentra en computadoras basadas

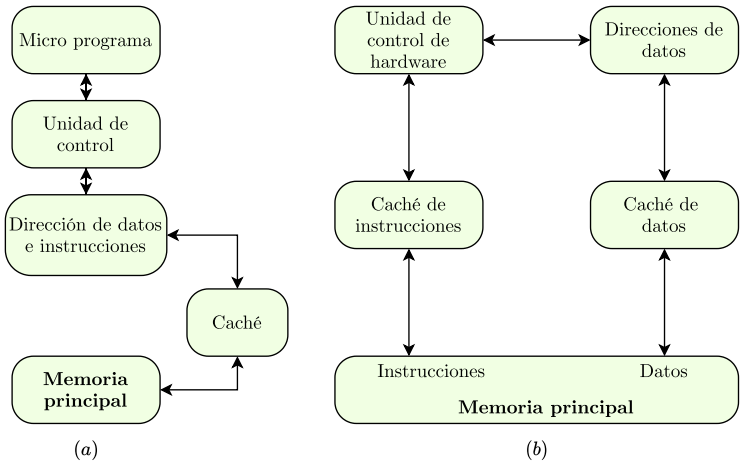


Fig. 5.5: Comparación entre las arquitecturas básicas CISC (a) y RISC (b)

en Pentium (de Intel) y Athlon (de AMD) ha evolucionado del modelo clásico x86 y exhiben numerosas características RISC (como descomponer una instrucción compleja en una típica de RISC). Actualmente Xeon (de Intel) y Opteron (de AMD) son dos íconos prominentes del mercado basados en la arquitectura x86-64. Entre las arquitecturas RISC famosas se pueden nombrar a MIPS (de Tecnologías MIP. S), Power (principalmente de IBM), SPARC (principalmente de SUN/Oracle), Alpha (de DEC) y ARM para sistemas embebidos (por ARM Ltd) [64].

Hoy en día AMD e Intel son las dos compañías líderes en la industria de los microprocesadores, cada una con su propia línea de arquitecturas CPU. Sus CPUs x86-64 en ambos casos emergieron como arquitecturas CISC y ahora incorporan características RISC, especialmente para proveer paralelismo a nivel de instrucción (ILP). Hoy en día los microprocesadores (de Intel y AMD) implementan un esquema que separa espacio de memoria para datos y para instrucciones en cachés del nivel 1 como mínimo.

Otra característica de un CPU moderno es que un número de núcleos de CPU se combinan en un mismo chip con un único controlador de memoria común integrado en el mismo chip, compartido entre todos. Inicialmente los CPUs de doble núcleo se introdujeron en el año 2005, pero a partir del 2012 CPUs de 12/16 unidades de procesamiento pueden hallarse en el mercado, a costos altos los cuales se incrementan con el número de núcleos. Más aún, lograr la mejor performance de un conjunto grande de núcleos en un CPU es una tarea desafiante, principalmente por los límites de ancho de banda de la memoria.

Otra innovación arquitectónica sofisticada en las CPUs modernas es la capacidad de multihilo dentro de un mismo núcleo. Esta consiste en proveer paralelismo entre los procesos, actuando como si fuese más de un procesador el que está haciendo el trabajo [64].

5.5. Computación de alto rendimiento y computación paralela

Tradicionalmente la computación en serie se refería a la ejecución de programas secuencialmente, de a una instrucción por vez, en un único procesador. La siempre creciente demanda de potencia de procesamiento para resolver los problemas computacionales ha servido de justificación para hallar tecnologías innovadoras para satisfacer estas necesidades, las cuales están por encima del alcance de la computación serial.

La Computación de Alto Rendimiento (HPC) es el campo que se enfoca en la búsqueda de metodologías y tecnologías innovadoras, tanto para el hardware como para el software, para responder a la gran demanda de capacidad de procesamiento. Estos son sistemas informáticos con una potencia computacional extremadamente alta capaces de resolver problemas enormemente complejos y exigentes.

La forma de cómputo que satisface las necesidades de performance es el esquema conocido como **computación paralela**. Esta se refiere a resolver un problema computacional, separándolo en partes y trabajando en ellas simultáneamente, en orden de llegar a la solución. Las unidades de procesamiento pueden ser procesadores separados, partes de un mismo procesador o, si se utiliza un sistema distribuido, desplegar las operaciones entre distintas máquinas. La computación paralela no solo provee la capacidad de concurrencia, sino que también colabora a resolver grandes problemas que serían imposibles de afrontar con un enfoque de computación serial.

En el pasado, una variedad de arquitecturas sofisticadas (distribuidas y con sistemas de memoria compartida) eran las únicas opciones de computadoras paralelas. Eran extremadamente caras y solo estaban en posesión de enormes organizaciones. Con el advenimiento de los clúster de computadoras, los cuales pueden consistir simplemente en interconectar computadoras personales compuestas de un procesador cada una, las CPUs multinúcleo, y, ahora, con los servicios en la nube, disponer de la capacidad de cómputo paralelo está al alcance de muchos usuarios. Nos encontramos, entonces, con la posibilidad de que no necesariamente la utilización de aglomeraciones físicas aisladas e independientes de recursos computacionales, lo que se conoce como paralelismo explícito, es la única manera de lograr ejecuciones paralelas de un programa. Se han desarrollado técnicas, como el paralelismo a nivel de instrucción (ILP), en donde con un solo procesador de un único núcleo se ha logrado el llamado paralelismo implícito [65].

5.5.1. Paralelismo implícito

El grado de ILP aplicable depende fuertemente de cuán relacionadas están las instrucciones del programa. Claramente, a mayor independencia entre ellas, las posibilidades de aplicar paralelismo aumentan. Esta dependencia afecta la manera en que los componentes de una aplicación (sentencias e instrucciones, iteraciones, etc) pueden ser ejecutadas ignorando la secuencia de eventos especificada por el programador, sin cambiar la salida. Entre los tipos comunes de dependencia se encuentran:

- Dependencia entre datos o real: se refiere a la situación en donde el contenido de una variable actualizado por una instrucción es utilizado por otra posterior a la operación que cambió su valor en primer lugar (casos de lecturas luego de escrituras).
- Dependencia de nombre o anti-dependencia: cuando el contenido de una variable es utilizado por una instrucción y luego dicho valor se actualiza por otra instrucción siguiente (escritura luego de lectura).
- La dependencia de control aplica a los escenarios en donde el contenido de una variable es actualizado por una instrucción y actualizado nuevamente por otra instrucción siguiente (escritura luego de escritura).

La dependencia real no puede ser eliminada porque es necesaria para el algoritmo, una dependencia de nombres, por otro lado, puede tratarse con una técnica de renombre de una variable/registro. La dependencia de control afecta el flujo de control en un

programa, al moverse de una instrucción hacia la otra, a través de la existencia de ramificaciones(condicionales if/else, switch), llamadas a funciones, etc. Los procesadores modernos incluyen predicción de ramificaciones y una característica de ejecución especulativa, que permita mayor paralelismo más allá de las limitaciones del control de flujo.

Las dependencias de los bucles, especialmente la dependencia que arrastra un bucle, en donde la iteración i puede requerir algún valor de la $(i-1)$, afectan seriamente la ILP y, en general, el paralelismo implícito. Los compiladores modernos pueden automáticamente asumir una reestructuración de código, al generar el código objeto, para enfatizar el paralelismo. Este esquema depende del nivel de privilegios que el programador le otorga al propio compilador [64].

5.5.2. Paralelismo explícito

El paralelismo explícito se caracteriza por el hecho de que el diseñador del sistema es responsable de subdividir el problema en partes más pequeñas (con respecto a datos o tareas) para su ejecución en simultáneo y, también, debe trabajar en la coordinación y sincronización de todos los subproblemas. Las formas más comunes de paralelismo explícito son [64]:

5.5.2.1. Paralelismo de memoria compartida

Desde el punto de vista del hardware, una arquitectura paralela de memoria compartida es una computadora que posee una memoria física común accesible a un número de procesadores físicos. Las dos clases de arquitecturas de memoria compartida son: Acceso Uniforme a Memoria (UMA) y Acceso No Uniforme a Memoria (NUMA) (ver Figura 5.6).

Hoy en día, la forma más común de una arquitectura UMA es el conocido multiprocesador simétrico (SMP) [66], el cual consiste en múltiples procesadores idénticos con un nivel de acceso igual y tiempos de acceso idénticos a la memoria compartida. Mientras que el esquema más común de una arquitectura NUMA son las máquinas hechas a partir de entrelazar o linkear una serie de SMPs. Se caracterizan por el hecho de que permiten que el tiempo de acceso a distintas locaciones de memoria sea variable según el procesador.

Desde el punto de vista de un programador, la forma más común de implementar la técnica de paralelismo de memoria compartida es por medio del modelo de programación multihilo. La aplicación paralela puede ejecutar múltiples hilos que comparten un mismo espacio lógico de almacenamiento. Hay que ser cuidadoso al aplicar esta técnica para que ningún proceso muera de inanición o acceda a datos inconsistentes. Algunas implementaciones estándar de hilos son POSIX Threads, Intel Threading Building Blocks, Cilk Plus y OpenMP [67].

La principal ventaja de la programación en memoria compartida es la uniformidad por el acceso compartido. No obstante, por las mismas razones, estos sistemas no son muy escalables. En este sentido, hay que combatir la congestión por el tráfico de datos, la cual ocurre cuando un gran número grande de procesadores comparten el mismo camino de acceso a la memoria global. A esto último se le suma que el costo de construir sistemas de memoria compartida, teniendo en cuenta el siempre creciente número de procesadores, aumenta exponencialmente.

El surgimiento y los rápidos avances en las CPUs multinúcleo han permitido que se las considere como una buena opción al momento de implementar plataformas de cómputo paralelas. En el mercado se encuentran tanto PCs como portátiles, que poseen dos, cuatro y

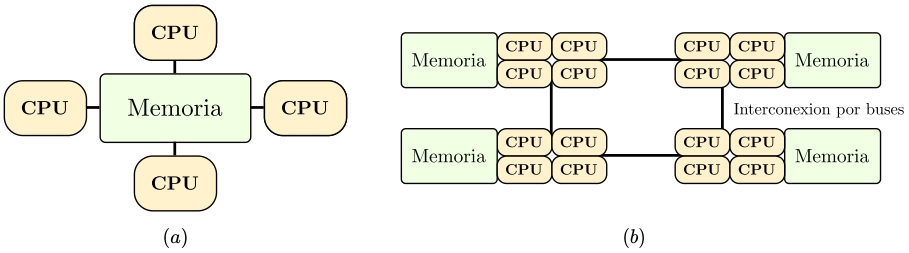


Fig. 5.6: (a) Acceso Uniforme a Memoria UMA. (b) Acceso No Uniforme a Memoria NUMA [64, Figura 4].

más núcleos. En la novena generación de procesadores Intel, el modelo Core i9-7960X posee 16 núcleos y con la tecnología hyper-threading son capaces de procesar simultáneamente dos hilos de ejecución en un mismo núcleo físico, con lo que soportan hasta 32 hilos [68].

5.5.2.2. Paralelización con memoria distribuida

Desde una perspectiva de hardware, una arquitectura paralela de memoria distribuida es una computadora que posee cierta cantidad de procesadores físicos, cada uno con sus propios recursos locales y un espacio de memoria separado, que se comunica por medio de una red de interconexión para coordinar los accesos a memoria de todos los elementos (ver Figura 5.7) [64].

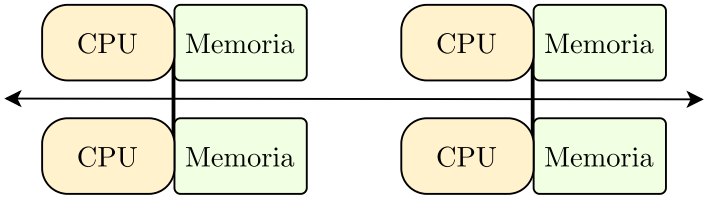


Fig. 5.7: Diagrama de bloques de una arquitectura con memoria distribuida [64, Figura 4].

Desde la perspectiva del programador, la manera más adecuada de enfrentar el paralelismo en espacios distribuidos de memorias compartidas es por medio del multiprocesamiento. Múltiples procesos, cada uno alocado con un subproblema, son mapeados con procesadores diferentes para resolver lo que les fue asignado. La comunicación interprocesos (para coordinación mutua y sincronización) es realizada utilizando el modelo de paso de mensajes. De acuerdo al cual los procesos (cada uno teniendo un espacio de memoria lógica separado) envían y reciben mensajes para compartir datos. Esto se logra de manera cooperativa. Cualquier mensaje SEND emitido por un proceso debe tener su correspondiente RECEIVE emitido por el proceso receptor. La implementación de este modelo es una librería de subrutinas que funciona en conjunto con los típicos compiladores de C, C++ y Fortran. Si bien existen algunas librerías que se pueden poner en práctica, la implementación más utilizada es la interfaz de paso de mensajes (MPI) [67]. Esta incluye una variedad de rutinas tanto comunicación punto a punto (peer to peer) como colectivas. La conexión que solo incluye un emisor y un receptor se denomina peer to peer. En contraste a la colectiva o grupal, que puede ser uno a todos, todos a uno o todos con todos. MPI se ha convertido en un estándar de facto para programación portable y escalable en sistemas con arquitecturas de memoria distribuida.

Las ventajas y desventajas asociadas a los esquemas de paralelismo en memorias distribuidas se contrastan con aquellos presentes en el paralelismo en memorias compartidas. Los sistemas distribuidos son altamente escalables y menos costosos. Más aún, computadoras sencillas y no muy potentes pueden ser utilizadas para construir un sistema con una buena performance, lo que se denomina un clúster (más adelante detallaremos los mismos). Por otro lado, la programación en sistemas distribuidos es más desafiante y requiere cuidados más finos e intensos, principalmente en aquellas áreas que despliegan las estructuras de datos entre los distintos espacios separados de memoria de los procesos paralelos y, también, todo lo relacionado a la comunicación cooperativa entre los procesos. En principio las aplicaciones de memoria distribuida MPI pueden ejecutarse en cualquier arquitectura, incluso para un esquema de memoria compartida, cada proceso posee su propio espacio de almacenamiento aislado del resto de los procesos.

5.5.2.3. Paralelismo en sistemas híbridos

Las arquitecturas híbridas paralelas (ver Figura 5.8) hacen referencia a los sistemas que consisten en un número de computadoras interconectadas mediante una red de datos con un sistema de memoria distribuida, en el cual cada máquina actúa como una computadora con memoria compartida (como SMP). Entonces, un sistema híbrido de memoria compartida puede ser construido al interconectar un conjunto de máquinas SMP en una red. Los clusters de computadoras de alta gama, por ejemplo, siguen un modelo de memoria híbrida [69].

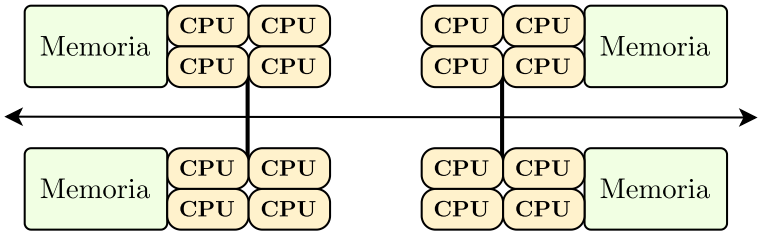


Fig. 5.8: Arquitectura híbrida: distribuida y con memoria compartida [64, Figura 5].

5.5.3. Arquitecturas paralelas actuales

La performance de un programa paralelo y de alto rendimiento en un sistema de memoria distribuida o híbrido depende de muchos factores. Entre ellos están la unidad de procesamiento, la memoria, la interconexión (latencias, anchos de banda), las capacidades del entorno (sistema operativo, compiladores, implementación, librería) y los algoritmos, las estructuras de datos y el patrón de acceso a ellos. Los esquemas que se utilizan para ejecutar una aplicación paralela son diversos. A continuación se mencionan algunos enfoques usualmente adoptados.

5.5.3.1. Computación en cluster

La computación en clúster es considerado como un sistema de procesamiento paralelo y distribuido que consiste en una colección de computadoras con hardware similar interconectadas por una red local trabajando de manera cooperativa como un recurso informático

único e integrado. El objetivo de esta arquitectura es poder utilizar el 100 % de su poder de cómputo en un solo problema.

Respecto al uso se pueden distinguir: los clústers de cómputo, los clusters de alta disponibilidad y aquellos clústers de balanceo de carga. En el presente trabajo se hablará de clusters de cómputo, ya que las otras opciones se refieren a campos relacionados con el área de sistemas distribuidos, pensados para dotar a estos últimos de características como tolerancia a fallos o proveer disponibilidad.

Volviendo a los clústers de cómputo han probado ser arquitecturas paralelas de memoria distribuidas muy prometedoras. Pueden ser vistos como un sistema paralelo híbrido, dado que cada computadora conectada conforma un SMP. A los elementos que constituyen el cluster se les denomina nodos y se les denomina nodos de cómputo a aquellos en los cuales se ejecutan las operaciones paralelas, mientras que los conocidos como nodos principales son los responsables del monitoreo global del estado del sistema [70].

Usualmente, el usuario del cluster se loguea en el nodo principal del cluster y compila sus programas utilizando el entorno que desee (de los que estén disponibles) y carga en dicho elemento el lote de trabajo que desee ejecutar. Este nodo gestiona las colas de solicitudes de todos los usuarios y planifica la ejecución de las mismas, desplegándose en los nodos de cómputo de acuerdo a mecanismos internos de prioridad preespecificados que utilizan un scheduler particular.

En grandes clústers, que poseen cientos o miles de nodos, las responsabilidades del nodo principal se distribuyen en otros tipos de nodos: un administrador, uno encargado de los ingresos de usuarios al sistema y uno o más que actúen como nodos de entrada y salida (para manejos eficiente de lotes grandes de datos). La interconexión entre todos los nodos puede ser simplemente utilizando Fast Ethernet (de 100 Mbps de ancho de banda) o Gigabit Ethernet (de 1000Mbps de ancho de banda). El clúster puede también estar equipado con una tecnología de conexión especializada de mucho ancho de banda y/o baja latencia, como 10 Gigabit Ethernet, Infiniband, Myrinet o Quadrics.

Si una red especializada está presente en el clúster, entonces se configura para que sea utilizada para comunicaciones MPI entre los procesos paralelos que han sido mapeados en diferentes nodos. En tal caso, la red Ethernet es utilizada para cuestiones de monitoreo. Los dos ítems a tener en cuenta al interconectar elementos son el ancho de banda y la latencia. El ancho de banda es la medida de transferencia de datos, es decir, la cantidad de información transmitida por unidad de tiempo, usualmente expresada en Mbps. La latencia se refiere al tiempo de transferencia para datos mínimos (cero bytes) entre dos puntos, generalmente expresada en microsegundos. Un ejemplo de un framework open source que facilita la distribución de problemas muy densos en datos a través de clústers de computadoras es Apache Hadoop [71].

5.5.3.2. Computación en grilla y en la nube

Existen aplicaciones paralelas, basadas en modelos de memoria distribuida, que pueden ser categorizadas como débilmente acopladas o fuertemente acopladas. Las primeras requieren muy poca o virtualmente ninguna comunicación entre los procesos paralelos y por lo tanto pueden ser desplegadas de una manera muy dispersa geográficamente. Dicha aglomeración de computadoras remotas interconectadas, posiblemente mediante Internet, se denomina **computación en grilla**. Esta es muy útil para compartir y facilitar el acceso a recursos computacionales remotos. Algunas características de las grillas son [72]:

- Gran escala: tienen que ser capaces de organizar un gran número de recursos sin que la performance del sistema decaiga a medida que el tamaño de la grilla aumenta.
- Distribución geográfica: los recursos de la grilla pueden encontrarse muy alejados geográficamente entre sí.
- Heterogeneidad: una grilla posee una gran variedad de recursos incluyendo software como de hardware que abarcan desde sensores hasta archivos.
- Intercambio de recursos: los recursos en una grilla pertenecen a un gran número de organizaciones las cuales permiten que otras organizaciones (los usuarios) puedan acceder a ellos. Los recursos no locales pueden entonces ser utilizados por aplicaciones, promoviendo la eficiencia y reduciendo los costos.
- Múltiples administradores: cada organización puede establecer diferentes políticas de seguridad y administrativas, bajo las cuales los recursos pueden ser accedidos y utilizados. Como resultado, la ya complicada seguridad de la red se dificulta aún más dado que se deben tener en cuenta las distintas facetas existentes.
- Coordinación de recursos: los recursos en una grilla deben ser coordinados de manera que no se produzcan fenómenos de inanición o largas esperas para los usuarios.
- Acceso transparente: una grilla debe verse como una única computadora virtual.
- Acceso confiable: una grilla debe asegurar que podrá satisfacer la demanda de todos los usuarios sin bajar la performance del sistema.

Otro tipo de sistemas, conocidos como **computación en la nube** o *cloud computing* es moneda corriente en casi todas las compañías hoy en día. A pesar de que no hay ninguna definición precisa de este concepto, se puede decir que se trata de un modelo que permite el acceso a una red a pedido, conveniente y ubicuo a un grupo compartido de recursos informáticos configurables (servidores, almacenamiento, aplicaciones y servicios) los cuales pueden ser contratados rápidamente y utilizados con muy pocos esfuerzos administrativos o con una mínima interacción con el proveedor de los mismos. El Instituto Nacional de Ciencia y Tecnología de los Estados Unidos [73] separa a la computación en la nube entre modelos de servicios y modelos de despliegue:

5.5.3.3. Modelos de despliegue

Estos modelos definen la naturaleza de la nube y abarcan las siguientes cuestiones:

- Public clouds: infraestructura disponible para el uso público y es propiedad de una organización que vende servicios en la nube.
- Private clouds: poseen un uso exclusivo de una organización. La nube puede estar manejada por dicha empresa o por un tercero.
- Hybrid clouds: combina distintos tipos: privada, community o pública, en donde todas las piezas conservan sus identidades únicas, pero están unidas como una unidad. Una nube híbrida puede ofrecer acceso estandarizado o propietario a datos y aplicaciones.

- Community cloud: es aquella en donde la nube ha sido organizada para dar servicio a una función específica o un propósito definido. Puede ser destinada a una organización a un número de organizaciones, pero todas comparten intereses similares, como la misión, políticas, seguridad, necesidades de cumplimiento normativo, etc. Una nube comunitaria puede ser administrada por las organizaciones integrantes o por un tercero.

5.5.3.4. Modelos de servicio

Los modelos de servicio comprenden la siguiente clasificación:

- Servicios de infraestructura: ofrecen grandes recursos computacionales como la capacidad de procesamiento o almacenamiento. Un usuario que paga por este servicio no tiene noción del lugar físico en donde se están guardando sus datos o procesando sus solicitudes. Ejemplos de estos son Google Compute Engine (GCE) u OpenStack.
- Servicios de plataforma: generalmente se abstrae de la infraestructura y estos servicios se enfocan en servir de soporte para el desarrollo de aplicaciones. Es el punto medio entre el hardware y la aplicación. Muchas compañías están tratando de dominar el área de los servicios de plataforma. Google Colab es un ejemplo, ya que es la base de desarrollo de infinitos modelos de Data Science. Microsoft Azure es otro PaaS muy conocido actualmente.
- Servicios de software: ellos prometen reemplazar las aplicaciones instaladas en las computadoras. En lugar de comprar el software, se seguirá un esquema de pago por uso. Este esquema es muy atractivo, y hay mucho software que funciona muy bien de esta manera como Gmail, Google Docs y Dropbox. Sin embargo, con las latencias de las redes actuales se dificulta muchísimo para aplicaciones a tiempo real como juegos en 3D.

5.5.4. CPUs y GPUs

La mayoría de CPUs modernas utilizadas en HPC son esquemas muy complejos. Contienen algunas de las estrategias ya mencionadas como la ejecución fuera de orden, la predicción de ramas, la técnica de precargar datos en memoria y la jerarquía de caches. El propósito de estas optimizaciones es mejorar la performance al recibir una serie de instrucciones. Anteriormente se ha tratado de adecuar las CPUs a los requerimientos de cómputo modernos. Esto se lograba aumentando la frecuencia de funcionamiento para acelerarlas [74]. Sin embargo, hoy en día ya se ha llegado a un punto en donde no se pueden acelerar más físicamente porque produciría problemas en el funcionamiento interno de estos elementos. Dicha situación sumada a la complejidad de las arquitecturas heterogéneas actuales basadas sólo en CPUs, sumado a la aceptación de CUDA como modelo de programación paralela de propósito general [75], han hecho que la GPU sea una arquitectura paralela opcional para la resolución de problemas de alta demanda computacional.

Si bien las Unidad de Procesamiento Gráfico (GPU) fueron concebidas para dedicarlas al procesamiento gráfico, actualmente son muchas las aplicaciones que se han adaptado a esta arquitectura, alcanzando las mismas una aceleración significativa en el ámbito de HPC. La velocidad de procesamiento que se puede obtener con estas nuevas plataformas paralelas es indiscutible debido a su diseño. Las GPUs, a diferencia de las CPUs, tienen

una mayor cantidad de transistores dedicados al procesamiento de datos, en cambio las CPUs dedican muchos de sus transistores para flujo de control y grandes cachés.

Si bien hasta el presente, el conjunto de problemas de propósito general resolubles en placas gráficas es menor que aquellos que pueden resolverse computacionalmente en las CPUs, estas aplicaciones se han visto altamente beneficiadas disminuyendo el tiempo de procesamiento utilizando sólo una GPU. Por otra parte, a medida que las GPUs se instalaron como una nueva arquitectura paralela han surgido los cluster de GPU [76] para satisfacer las demandas de altísimas necesidades de cómputo.

Las GPUs pequeñas se observan principalmente en sistemas embebidos, como tablets y smartphones, en donde el chip se compone de una cantidad de núcleos mucho menor (números de un solo dígito) y se combinan con Unidad de Procesamiento Central en el mismo integrado. En los últimos años, las GPUs han sido implementadas en numerosas áreas, como la bioinformática, la dinámica molecular, las finanzas computacionales y el campo que estudiamos actualmente, la Inteligencia Artificial. Por estos grandes campo de aplicación, se comenzó a utilizar el término computación de propósito general en unidades de procesamiento gráfico (GPGPU) para referirse a todas aquellas actividades que estudian cómo aprovechar las capacidades de cómputo de una GPU, con el objetivo de que pueda ejecutar tareas diseñadas tradicionalmente para una CPU. Las principales razones para la migración de aplicaciones altamente paralelizables, de naturaleza secuencial, es debido a el desarrollo de los lenguajes de programación en GPUs, como CUDA de NVIDIA y OpenCL del Grupo Khronos, también algunas modificaciones pequeñas a la arquitectura original de la GPU facilitaron el procesamiento de propósito general sumado a que el costo de adquirir esta tecnología no es tan excesivo y brinda una alta performance.

5.5.4.1. Latencia y rendimiento

En la Figura 5.9 se explaya la principal diferencia entre una CPU y una GPU. Las CPUs suelen tener menos núcleos, siendo lo normal para una computadora de trabajo promedio, tener cuatro o seis con un número mayor de hilos que pueden ejecutar de manera concurrente. Se dice que siguen una arquitectura multicores, y el agregado de núcleos es con el objetivo de acelerar las aplicaciones existentes. Cada instrucción de CPU puede generar muchos cambios trabajando de manera independiente. Las GPUs, por otro lado, poseen miles de núcleos, son arquitecturas many cores. La desventaja es que cada uno corre a una menor frecuencia y no son tan poderosos como el anterior. Los núcleos de una GPU no operan de manera independiente, la unión de todos ellos hace que puedan paralelizar tareas, distribuyendolas a través de las distintas unidades.

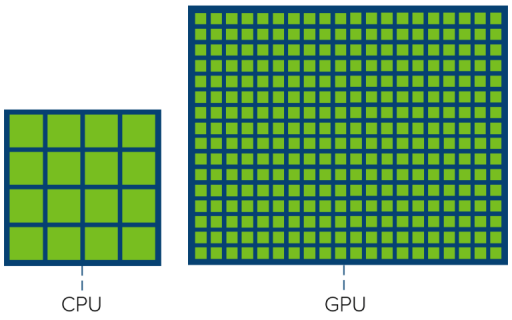


Fig. 5.9: Comparación de densidad de núcleos por pastilla de CPU y GPU [77].

5.5.4.2. Diferencia y similitudes

Al observar las Figuras 5.10 y 5.11 podemos notar numerosas similitudes entre una CPU y una GPU. Ambas utilizan la memorias caché, un controlador de memoria y una memoria global. Una visión de alto nivel de un CPU moderno indica que está optimizado para disminuir las latencias de acceso a memoria por la utilización de numerosas capas de caché.

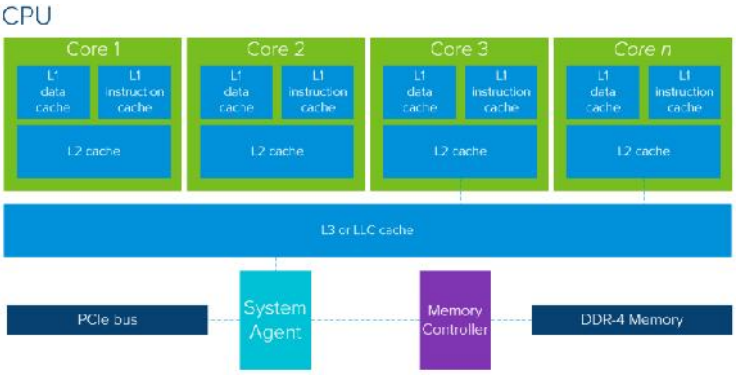


Fig. 5.10: Arquitectura a alto nivel de una CPU moderna [77].

Como se observa, la Figura 5.10 es una visión a alto nivel de una CPU moderna. Está compuesta de varios núcleos, cada uno soportado con su propia caché. Si los datos que necesita no están en estas últimas, deberá buscarlo en la caché compartida y caso contrario en la memoria (DDR-4 Memory).

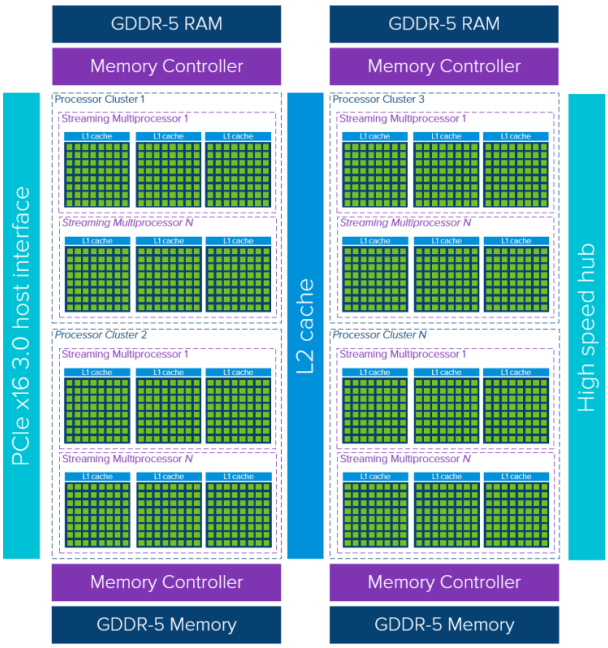


Fig. 5.11: Arquitectura a alto nivel de una GPU moderna [77].

Una sola GPU consiste de numerosos clusters de procesos los cuales a su vez contienen varios Streaming Multiprocesors (SM). Cada SM comparte una capa de caché L1 con los

núcleos agrupados en dicho espacio. Típicamente un SM recurre a su caché L1 y luego a una caché compartida L2 antes de ir a buscar datos a la memoria global GDR-5 . Esta arquitectura es tolerante a la latencia de memoria.

Una GPU funciona con una menor cantidad de memorias caché, las cuales son relativamente más pequeñas que las de una CPU. La razón es que una GPU posee una mayor cantidad de transistores dedicados al procesamiento lo que hace que no esté focalizada en cuanto tarda en recuperar datos de la memoria. Sin embargo, a pesar de que las memorias son más pequeñas, una Unidad de Procesamiento Gráfico (GPU) posee un mayor ancho de banda para recuperar los datos de la misma, que ayuda a equilibrar las latencias.

	CPU	**GPU**
Memoria	6 - 64 GB	768 MB - 6 GB
Ancho de memoria	24 - 32 GB/s	100 - 200 GB/s
Caché L2	8 - 15 MB	512 - 768 kB
Caché L1	256 - 512 kB	16 - 48 kB

Tab. 5.1: Comparación prestaciones de CPU vs. GPU [78]

Las Unidad de Procesamiento Gráfico (GPU) son procesadores paralelos dedicados, optimizados y completamente programables [79]. Por esta razón son perfectas para aplicarlas a la computación de alto rendimiento, ya que son una herramienta que colabora al paralelismo de aplicaciones avanzadas y se ajustan a las altas cargas de trabajo típicas de dichos sistemas. Un problema que es resuelto de manera muy eficiente por una Unidad de Procesamiento Gráfico (GPU) es la multiplicación de matrices. Mientras que una CPU este cálculo es resuelto en términos secuenciales, en una Unidad de Procesamiento Gráfico (GPU) se distribuyen todas las filas para computarlas de manera independiente y así obtener el resultado más rápido. Este caso se puede abstraer y comparar con una convolución, en donde tenemos tensores de entradas, de pesos y el tensor de salida, el cual es una combinación entre los dos primeros elementos. Por esta razón es aconsejable el uso de Unidad de Procesamiento Gráfico (GPU)s frente a las CPUs para entrenar redes neuronales [80].

5.6. Métrica de performance

En la literatura se encuentran distintas métricas para cuantificar la performance de los programas paralelos. Estas métricas incluyen el tiempo total de ejecución, el *relative speedup* (o solo *speedup*) y la eficiencia relativa (*efficiency*). El tiempo de ejecución considera los tiempos de comunicación y de cálculo. Es todo el tiempo que pasa desde que se comenzó a ejecutar el primer proceso de un programa paralelo hasta que termina la ejecución del último. Relative Speedup S de un programa paralelo es la relación entre el tiempo que transcurre para que un proceso resuelva un problema, τ_1, y el tiempo para que ese mismo problema sea resuelto por n procesos [81]:

$$S = \frac{\tau_1}{\tau_n}$$

La eficiencia relativa se define como:

$$\varepsilon = \frac{S}{n}$$

En general, la medida de *speedup* es un valor menor a n y la eficiencia está dentro del rango $[0, 1]$.

5.7. Lista TOP 500

Las estadísticas sobre la computación de alto rendimiento son de gran interés para los fabricantes y usuarios. Estas pueden facilitar el establecimiento de colaboraciones y el intercambio de datos, además de ayudar a una mejor comprensión del mercado.

El proyecto TOP500 se inició en 1993 para proporcionar una base confiable para rastrear y detectar las tendencias en el área de la computación de alto rendimiento. Dos veces al año, se recopila y publica una lista durante la conferencia ISC High Performance, con la ayuda de expertos en computación de alto rendimiento, científicos computacionales, fabricantes y la comunidad de Internet en general.

La tabla TOP500 muestra los 500 sistemas informáticos comercialmente de propósito general más potentes que se encuentran disponibles actualmente. En esta se enumeran las computadoras clasificadas por su desempeño en el LINPACK Benchmark. La lista contiene una variedad de información, incluidas las especificaciones del sistema, sus principales áreas de aplicación, además de la ubicación de las mismas [82].

En noviembre de 2020, para la 56ª edición del TOP500, se estableció que la supercomputadora más rápida es Fugaku de Japón (ver Figura 5.12), con una puntuación de referencia Linpack de alto rendimiento de 442 petaflops, superando al ahora segundo sistema Summit por un factor de 2,8 veces. Una descripción de los primeros 5 puestos se desarrolló en la Tabla 5.2, donde Estados Unidos tiene tres de los puestos, mientras que China y Japón tienen uno cada uno.

Fig. 5.12: Las dos supercomputadoras más potentes a la fecha de noviembre de 2020, donde a la izquierda se encuentra Fugaku y a la derecha Summit [83].

Cómo se puede apreciar en la Tabla 5.2, Fugaku, funciona con el SoC A64FX de 48 núcleos de Fujitsu, convirtiéndose en el primer sistema número uno en la lista que funciona con procesadores ARM. En precisión simple o reducida, un método popular para reducir la demanda de ancho de banda de memoria de las redes neuronales haciendo uso de números de coma flotante de 16 u 8 bits, utilizado en aplicaciones de IA y aprendizaje automático, el rendimiento máximo de Fugaku es de más de 1000 petaflops (1 exaflops). El nuevo sistema está instalado en el Centro RIKEN de Ciencias Computacionales en Kobe, Japón [83].

	Puesto	País	Núcleos	Memoria [GB]	Procesador	Performance [TFlop/s]	Fabricante
Fugaku	1	Japón	7.630.848	5.087.232	A64FX 48C 2.2GHz	442.010	Fujitsu
Summit	2	Estados Unidos	2.414.592	2.801.664	IBM POWER9 22C 3.07GHz	148.600	IBM
Sierra	3	Estados Unidos	1.572.480	1.382.400	IBM POWER9 22C 3.1GHz	94.640	IBM NVIDIA Mellanox
Sunway TaihuLight	4	China	10.649.600	1.310.720	Sunway SW26010 260C 1.45GHz	93.014	NRCPC
Selene	5	Estados Unidos	555.520	1.120.000	AMD EPYC 7742 64C 2.25GHz	63.460	NVIDIA

Tab. 5.2: Primeros 5 puestos en la lista TOP500

5.8. Librerías de Deep Learning

5.8.1. Introducción a Caffe

Caffe es un framework de código libre para trabajar con técnicas de Deep Learning originalmente desarrollado por Berkeley AI Research (BAIR) / Berkeley Vision and Learning Center (BVLC) y contribuidores de la comunidad. Está escrito en C++ y tiene una interfaz en Python.

A diferencia de otros códigos de investigación que había por esos tiempos, Caffe era una librería de DL bien diseñada, que hacía fácil construir una red a partir de un archivo de texto prototxt. Su diseño era modular, haciendo fácil que otros investigadores pudieran agregar nuevas capas y algoritmos de entrenamiento. Esto hizo que Caffe fuese popular durante el periodo de 2012 hasta 2016. La mayoría de las redes y modelos importantes en el campo de reconocimiento de imágenes fueron lanzadas en Caffe [84].

Mientras tanto, había un interés creciente en alternativas a Caffe. Esto se debía a las limitaciones que estaba empezando a mostrar Caffe. La librería estaba principalmente escrita con C++, lo que significaba tiempo lentos de experimentación y desarrollo. Otras características útiles como la utilización de diferentes tipos de datos de precisión, cuantización y el entrenamiento haciendo uso de múltiples GPU no estaban presente en Caffe. Estos problemas resultaron en una nueva camada de librerías de DL que fueron escritas pensando en el caso de uso, entrenamiento distribuido y la customización. Estos incluyen TensorFlow y PyTorch.

En abril de 2017 Facebook anunció Caffe2 que incluía nuevas características cómo redes neuronales recurrentes. A finales de marzo de 2018 Caffe2 fue fusionado con PyTorch.

5.8.2. Diferencias entre TensorFlow y PyTorch

TensorFlow [85] es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google. Permite construir y entrenar redes neuronales. Actualmente es utilizado tanto en la investigación como en los productos de Google.

TensorFlow fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto Apache 2.0 el 9 de noviembre de 2015. TensorFlow provee APIs estables para Python y C++. Esta

librería también goza de una comunidad de usuarios grande y tremendamente activa, que aporta constantes contribuciones de código y resolución de problemas en GitHub. El marco de TensorFlow tiene un amplio respaldo en la industria y se ha convertido en una opción válida para la investigación de aprendizaje profundo y el desarrollo de aplicaciones, especialmente en ámbitos como la visión artificial, la comprensión de lenguaje natural y la traducción de voz.

TensorFlow es el sistema de aprendizaje automático de segunda generación de Google Brain, liberado como software de código abierto en 9 de noviembre de 2015. Mientas la implementación de referencia se ejecuta en dispositivos aislados, TensorFlow puede correr en múltiple CPUs y GPUs. TensorFlow está disponible para Windows, Linux, macOS, y plataformas móviles que incluyen Android e iOS.

Por otro lado PyTorch [86] es un paquete de Python pensado en la comunidad científica que busca un reemplazo de numpy y/o en el desarrollo de redes neuronales. Basada en la librería Torch, PyTorch es una librería de aprendizaje de máquina de código abierto. Facebook desarrolla PyTorch. También está diseñado para realizar cálculos numéricos haciendo uso de la programación de tensores. Además permite su ejecución en GPU para acelerar los cálculos.

Tanto la librería de TensorFlow como PyTorch se basan en el uso de grafos para realizar sus cómputos. En matemáticas, un grafo es un conjunto de nodos unidos por enlaces llamados aristas que permiten representar relaciones entre elementos de un conjunto.

En TensorFlow todos los códigos contienen dos partes importantes, que son el grafo y la sesión. El grafo computacional es una serie de operaciones de TensorFlow organizadas en un grafo de nodos. Cada grafo en esta librería contiene un conjunto de objetos *tf.Operation*, que representan unidades de cómputo y objetos *tf.Tensor*, que representan las unidades de datos que fluyen entre las operaciones [87]. En cada nodo de un grafo en TensorFlow toma cero o más tensores como entrada y producen un tensor como salida.

En particular esta librería hace uso de grafos estáticos donde primero debemos definirlo, indicando que se quiere hacer con los datos, y después utilizar una sesión para calcular los resultados de los tensores. Este tipo de procedimiento dificulta la depuración de código y hace más tediosa su implementación. Una ventaja con respecto a TensorFlow y sus grafos es que provee de una herramienta conocida como TensorBoard para visualizar a los grafos.

PyTorch también modela sus operaciones en un grafo de nodos, y también utiliza tensores cómo estructura de datos. En PyTorch un tensor es muy similar a un array numpy, excepto que está diseñado para trabajar de forma paralela y usar las ventajas de las capacidades brindadas por una GPU.

Al contrario que en el paquetes de Tensorflow, PyTorch trabaja con grafos dinámicos en vez de estáticos. Esto significa que en tiempo de ejecución se pueden ir modificando operaciones y el cálculo del gradiente variará con estas modificaciones de operaciones.

Otra diferencia es que PyTorch trabaja de forma directa con tensores sin la necesidad de una librería a un nivel superior como pueda ser Keras para Theano o Tensorflow, aunque en el caso de TensorFlow con su nueva versión 2.0 se está buscando consolidar el uso de esta API intuitiva de alto nivel basada en Keras [88].

Por último PyTorch ofrece una gran variedad de modelos ya entrenados dentro de su módulo de torchvision. El paquete de modelos torchvision.models contiene definiciones para diferentes tareas incluyendo la clasificación de imágenes, detección de objetos y clasificación de vídeo entre otras. Dentro del área que nos compete esta librería nos provee de arquitecturas conocidas como AlexNet [4], VGG [21], ResNet [15] y GoogLeNet [23] entre

otros.

	PyTorch	**TensorFlow**
Desarrollado por	Facebook	Google
Grafos	Dinámicos	Estáticos
Característica distinguida	Soporte para CUDA	TensorBoard
Curva de aprendizaje	Fácil de aprender	Curva de aprendizaje empinada
Comunidad	Pequeña en comparación	Grande
Desarrollo	Menos soporte	Da apoyo
Orientado	Industria	Comunidad y trabajo científico

Tab. 5.3: Comparación entre las librerías de DL TensorFlow y PyTorch

5.9. Plataformas web con GPUs

5.9.1. Ventajas de uso

Debido al alto costo de los componentes de hardware vistos en el apartado anterior, es una buena decisión tratar de hacer uso de ellos a través de los servicios en la nube. Este tipo de plataformas en la nube ofrecen una innovación más rápida y facilitan el acceso a recursos flexibles. Lo habitual es pagar sólo por los servicios en la nube utilizados, de tal forma que ayude a reducir los costos operativos, evitando adquirir un hardware que quedará obsoleto en poco tiempo, a ejecutar la infraestructura con más eficacia y a escalar a medida que cambian las necesidades.

Para nuestro problema particular del tratamiento de señales, la ejecución de muchas de las técnicas de Deep Learning actuales requieren de un gran poder de cómputo del cual no disponemos por lo que estas plataformas con disponibilidad de aceleración de hardware por medio de GPU son de gran ayuda.

A la hora de comparar las distintas opciones se tuvieron en cuenta los productos que ofrecen las plataformas, el precio y la memoria que brinda para el cómputo.

5.9.2. Colaboratory

Colaboratory, o Colab para abreviar, es un producto de Google Research. Colab permite que todos puedan escribir y ejecutar código arbitrario de Python en el navegador. Es ideal para aplicarlo en proyectos de aprendizaje automático, análisis de datos y educación. Técnicamente, Colab es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube.

Los recursos de Colab no están garantizados ni son ilimitados y, en ocasiones, los límites de uso fluctúan. Esto es necesario para que Colab pueda ofrecer los recursos en forma gratuita. Los tipos de GPU disponibles en Colab varían con el tiempo. Esto es para que Colab pueda ofrecer un acceso gratuito a los recursos. Las GPU disponibles en Colab a menudo incluyen K80, T4, P4 y P100 de NVIDIA [89].

No hay una forma de elegir el tipo de GPU a la que te puedes conectar en Colab. A veces, las GPU se priorizan para los usuarios que utilizan Colab de manera interactiva, en lugar de los que ejecutan cálculos de larga duración o de aquellos que recientemente usaron menos recursos en Colab. Como resultado, los usuarios que utilizan Colab para

ejecutar cálculos de larga duración o los que usaron más recursos últimamente tienen más probabilidades de encontrar límites de uso o de que se restrinja de manera temporal su acceso a las GPU. Los usuarios que buscan tener un acceso más confiable a las GPU más veloces de Colab pueden probar Colab Pro.

5.9.3. Colab Pro

Colab Pro es un servicio pago con el cual se tiene acceso prioritario a GPU más veloces. Por ejemplo, se puede acceder a la a GPU T4 y P100 en momentos en los que los usuarios no suscritos acceden a K80. También se tiene acceso prioritario a TPU. Sin embargo, sigue habiendo límites de uso en Colab Pro, y los tipos de GPU y TPU disponibles pueden variar con el paso del tiempo. En la versión gratuita de Colab, el acceso a las GPU más rápidas es muy limitado, y los límites de uso son mucho más bajos que en Colab Pro.

Las diferencias fundamentales para ambas plataformas son:

- Precio: Colab es gratuito y su versión Pro cuesta 9.99 dolares por mes.
- Tiempo: en la versión Colab los notebooks se conectan a máquinas virtuales que tienen una vida útil máxima de hasta 12 horas. Además, se desconectan los notebooks de las VM cuando permanecen inactivos durante un período prolongado. La vida útil máxima de las VM y el tiempo de espera de inactividad pueden variar con el tiempo o en función de tu uso. En Colab Pro los notebooks pueden permanecer conectados hasta 24 horas y los tiempos de espera de inactividad son relativamente flexibles, comparándolo con la versión gratuita. Sin embargo, las duraciones no están garantizadas y es posible que esos tiempos varíen.
- Memoria disponible: en la versión gratuita Colab la cantidad de memoria disponible en las máquinas virtuales varía con el tiempo, pero es estable durante el transcurso de la vida útil de la VM. Se asigna automáticamente una VM con memoria adicional cuando Colab detecte que probablemente la necesites. En Colab Pro se tiene acceso prioritario a una VM con memoria alta. Estas VM generalmente tienen el doble de CPU y de memoria que las estándar de Colab. Es posible que se te asigne automáticamente una VM con memoria alta cuando Colab detecte que es probable que la necesites. No obstante, los recursos no están garantizados y existen límites de uso de las VM con memoria alta.

5.9.4. Google Cloud

Google Cloud es una plataforma que es utilizada para crear ciertos tipos de soluciones a través de la tecnología almacenada en la nube y permite al igual que muchas de las tecnologías de esta índole en la nube la rapidez y la escalabilidad de su infraestructura.

En el apartado de la IA y el aprendizaje profundo Google Cloud ofrece para nuestras necesidades imágenes de máquina virtual preconfiguradas para aplicaciones de aprendizaje profundo. Esta permite aprovisionar una máquina virtual de forma rápida con todo lo necesario para iniciar un proyecto de aprendizaje profundo en Google Cloud. Permite lanzar instancias de Compute Engine preinstaladas con frameworks de aprendizaje automático muy conocidos (como TensorFlow, PyTorch o scikit-learn) y hacer que sean compatibles con GPU y TPU de Cloud con sólo un clic [90].

Respecto a la memoria disponible por la plataforma, Google Cloud pone a disposición máquinas virtuales equipadas con GPU capaces de ofrecer un rendimiento por instancia de hasta 960 teraflops. Gracias a las GPU NVIDIA Tesla K80, P4, T4, P100 y V100, se agilizan tareas como el aprendizaje profundo, la simulación física y la creación de modelos moleculares.

Por último relacionado al precio esta plataforma ofrece empezar a usar cualquiera de los productos con un crédito gratuito de 300 dolares.

5.9.5. Amazon Web Services

Amazon Web Services (AWS) es una plataforma en la nube que ofrece más de 175 servicios integrales de centros de datos a nivel global. Entre otros productos esta plataforma ofrece los servicios de AWS mediante una experiencia de TensorFlow completamente administrada a través de Amazon SageMaker, una plataforma diseñada para crear, entrenar e implementar modelos de aprendizaje automático a escala. O bien, puede usar las AMI de aprendizaje profundo de AWS para crear flujos de trabajo y entornos personalizados con TensorFlow y otros marcos conocidos, como Apache MXNet, PyTorch, Caffe, Caffe2, Chainer, Gluon, Keras y Microsoft Cognitive Toolkit.

Amazon SageMaker Studio proporciona una única interfaz virtual basada en la web donde puede realizar todos los pasos de desarrollo de aprendizaje automático. SageMaker Studio le brinda acceso completo, control y visibilidad en todos los pasos requeridos para crear, entrenar e implementar modelos. Puede cargar datos, crear blocs de notas nuevos, entrenar y ajustar modelos, retroceder y avanzar entre los pasos para ajustar experimentos, comparar resultados e implementar modelos para la producción, todo de forma rápida y en un solo lugar, lo cual aumenta su productividad.

Con respecto al precio, Amazon ofrece dentro de su capa gratuita una prueba de su producto Amazon SageMaker por 250 horas al mes de uso del cuaderno de notas t2.medium (2 CPU virtuales y 4 GiB de memoria virtual) durante los primeros dos meses, o 50 horas por mes de m4.xlarge (4 CPU virtuales y 16 GiB de memoria virtual) para entrenamiento durante los dos primeros meses. La capa gratuita comienza el primer mes, cuando crea su primer recurso de SageMaker [91].

5.9.6. Microsoft Azure

Microsoft Azure es un servicio de computación en la nube creado por Microsoft e introducido al mercado en octubre de 2008 para construir, probar, desplegar y administrar aplicaciones y servicios mediante el uso de sus centros de datos. Proporciona software como servicio, plataforma como servicio e infraestructura como servicio y es compatible con muchos lenguajes, herramientas y marcos de programación diferentes. Esta plataforma utiliza principalmente un modelo de precios de pago 'como va', que se carga basado en el uso.

Dentro de los distintos productos que ofrece la plataforma, el que más nos interesa es Azure Machine Learning, un servicio de aprendizaje automático de nivel empresarial para crear e implementar modelos con más rapidez. También ofrece compatibilidad integrada con herramientas y plataformas de código abierto para la inferencia y el entrenamiento de modelos de Machine Learning como PyTorch, TensorFlow y scikit-learn, o el formato abierto e interoperable ONNX [92].

Plataforma	Poder cómputo (GPU)	Precio	Memoria
Colaboratory	K80 (3 teraflops)	Gratuito	12 GB
Colab Pro	GPUs de alta gama como T4 y P100 (21 teraflops)	9.99 USD/ mes	25 GB
Amazon SageMaker Instancias P4	8 GPUs NVIDIA A100	desde 94 USD/año hasta 2367 USD/año	1152 GB
Azure Notebooks	4 núcleos CPU o 2 núcleos CPU y una GPU	Gratuito	4 GB

Tab. 5.4: Comparación entre distintas plataformas web con aceleración de GPU disponibles.

Machine Learning Studio se ofrece en dos niveles: Gratis y Estándar. La versión basic es para desarrollo de código abierto en la nube con una experiencia de programación básica. El nivel de enterprise ofrece todos los servicios del nivel básico más características de IU y ademas administración segura y exhaustiva del ciclo de vida del aprendizaje automático para todos los niveles de conocimiento.

5.9.7. Comparación de plataformas web

La siguiente Tabla fue desarrollada a modo de comparación entre algunas de las distintas plataformas web que se pueden adquirir hoy en día para la implementación de modelos de aprendizaje profundo:

Capítulo 6

Teledetección o sensado remoto

El sensado remoto o teledetección (*remote sensing* en inglés) es la ciencia de adquirir información sobre la superficie de la Tierra sin estar en contacto con la misma. Esto se hace mediante una variedad de sensores que detectan la energía reflejada o emitida por la superficie de la Tierra y luego es procesada de manera conveniente [93].

Los satélites utilizados para la observación de la Tierra son capaces de decodificar mucha más información electromagnética que la que se encuentra en el rango visible. Algunos satélites, como los hiperespectrales, tienen hasta más de 256 sensores con los que forman una señal de 256 capas de información, la cual no se podría obtener fácilmente sólo con el estudio del color visual estándar [94]. Algunas veces estos **datos satelitales** capturados por los sensores ubicados a bordo se pueden ver como una representación visual georreferenciada de información.

Una imagen puede ser definida cómo una función de dos dimensiones, $f(x, y)$, donde x e y son las coordenadas espaciales, y la amplitud de f en cualquier par de coordenadas (x, y) es la intensidad del nivel de gris de la imagen en ese punto. Luego cuando x, y y los valores de amplitud de f son todos finitos, en cantidades discretas, hablamos de una **imagen digital**. Esta imagen está compuesta por un número finito de elementos o pixeles, cada uno con una ubicación particular y un valor discreto, generalmente entre 0 y 255 para imágenes de 8 bits [95]. A su vez, las imágenes contenidas en las bases de datos que se utilizaron para este trabajo son imágenes digitales de tres dimensiones, también conocidas cómo RGB, donde una de estas variables representa los tres canales de colores existentes (rojo, verde y azul) que los seres humanos podemos decodificar con nuestros propios ojos.

Las imágenes satelitales son una herramienta eficaz y de gran utilidad para el estudio del clima, los océanos, los vientos, la vegetación, y también en el campo militar para distintas tareas de reconocimiento. Sus aplicaciones en distintas áreas, cómo en la agricultura, conservación de la biodiversidad, geología y uso de la tierra, entre otros, serán detalladas en la sección 6.4.

Históricamente, sólo los gobiernos y las grandes corporaciones han tenido acceso a imágenes satelitales de calidad, sin embargo en los últimos años, existen cada vez más conjuntos de datos de este tipo de imágenes disponibles para cualquier persona con una computadora y una conexión a Internet. La cantidad y la calidad de estos conjuntos de datos de imágenes mejora continua y aceleradamente, y además hay muchas plataformas gratuitas y comerciales a disposición para adquirir estos datos. En el ámbito comercial, los precios de las imágenes van desde unos pocos dólares por una sola imagen hasta aproximadamente los miles de dólares por la imagen de mayor calidad posible.

Debido a su importancia, este tipo de imágenes de la superficie terrestre son consideradas de utilidad pública, por lo cual muchos países mantienen programas de investigación de estas imágenes. Estados Unidos ha liderado el camino para hacer que estos datos estén disponibles gratuitamente para uso científico.

El tipo de imágenes que emplearemos para el entrenamiento y posterior evaluación de los modelos planteados son generadas por la combinación de una fuente de iluminación y la reflexión o absorción de energía de la fuente por los elementos de la escena a capturar [95]. La información provista por estos sensores puede ser procesada, analizada y aplicada a distintas cuestiones. Los siguientes siete dispositivos, que se observan en la Figura 6.2 son partícipes y comprenden el proceso de sensado remoto de principio a fin, los cuales implican una interacción entre la radiación incidente y los objetivos de interés [96]:

1. **Fuente de energía o iluminación**: el primer requisito para el sensado remoto es tener una fuente de energía electromagnética que ilumine al objetivo de interés. Generalmente hablamos del Sol cómo fuente principal de energía para este proceso. Sin embargo como se desarrollará en la sección 6.2.1, los sensores activo proporcionan sus propias fuentes de energía.

2. **Radiación y atmósfera**: la energía viaja desde una fuente emisora hasta el objetivo, entrando en contacto e interactuando con la atmósfera que atraviesa. Esta interacción puede tener lugar una segunda vez cómo la energía que viaja desde el objetivo hasta el sensor. Entonces antes de que la radiación utilizada para el proceso de sensado remoto llegue a nuestra superficie, debe viajar a través de la atmósfera terrestre por cierta distancia. En este viaje existen partículas y gases que componen la atmósfera que pueden afectar la luz y la radiación entrantes provocando efectos la dispersión y absorción, los cuales son observables en la Figura 6.1.

   La dispersión (*scattering*) ocurre cuando las partículas de gas presentes en la atmósfera interactúan y hacen que la radiación electromagnética se desvíe de su camino original. La absorción es el otro mecanismo principal en funcionamiento, el cual se da cuando las moléculas de la atmósfera absorben energía en varias longitudes de onda. El ozono, dióxido de carbono y el vapor de agua son los tres principales constituyentes atmosféricos que absorben radiaciones.

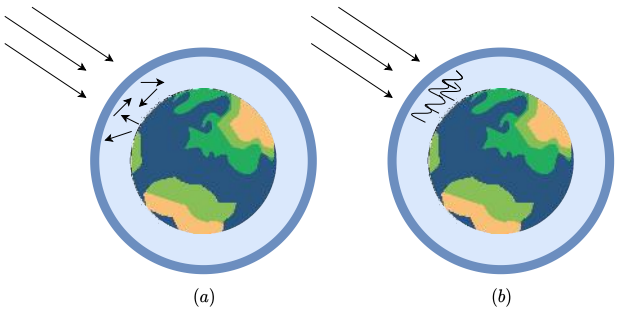


Fig. 6.1: (*a*) Dispersión; (*b*) Absorción.

3. **Interacción con el objetivo**: una vez que la energía llega al objetivo, esta interactúa con el objetivo dependiendo de las propiedades tanto del mismo como de la radiación en sí. La radiación que no se absorbe ni se dispersa en la atmósfera puede

alcanzar e interactuar con la superficie terrestre de tres formas: absorción, transmisión y reflexión. Las proporciones de cada una dependerá de la longitud de onda del energía y el material del objetivo en cuestión.

4. **Registro de energía por el sensor**: después de que la energía ha sido dispersada por, o emitida desde el objetivo, se necesita de un sensor remoto para recolectar y registrar la radiación electromagnética.

5. **Transmisión, recepción y procesamiento**: la energía registrada por el sensor es transmitida, a menudo en forma electrónica, a una estación de recepción y procesamiento donde el los datos se procesan en una imagen.

6. **Interpretación y análisis**: la imagen procesada se interpreta, visualmente y/o digital o electrónicamente, para extraer información sobre el objetivo que fue iluminado.

7. **Aplicación**: el elemento final del proceso de sensado remoto se logra cuando se aplica la información extraída de las imágenes sobre el objetivo para comprenderlo mejor, revelar información nueva o ayudar a resolver un problema en particular. En nuestro caso particular utilizaremos estas imágenes para entrenar distintos modelos de CNN, los cuales luego de ser implementados podrán ser capaces de clasificar entre distintas categorías de escenarios urbanos.

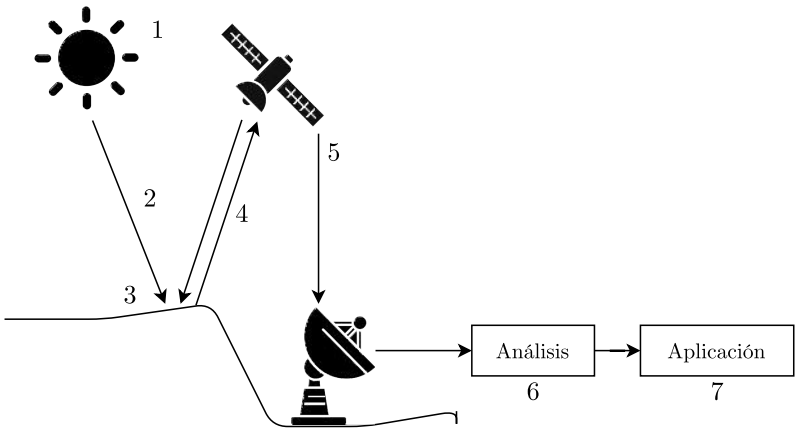


Fig. 6.2: Elementos del proceso de teledetección o sensado remoto.

6.1. Imágenes y fotografías

La energía electromagnética puede ser detectada de manera fotográfica o electrónicamente. En el proceso fotográfico se utiliza reacciones químicas en la superficie de un rollo de película sensible a la luz para detectar y registrar variaciones de energía.

En el área de sensado remoto es importante poder distinguir entre lo que son las imágenes y las fotografías. Una imagen se refiere a cualquier representación pictórica, independientemente de qué longitudes de onda o qué dispositivo se haya utilizado para detectar y registrar la energía electromagnética. En cambio, una fotografía se refiere especialmente a imágenes que han sido detectadas y grabadas en films de fotografía. Se puede

decir que todas las fotografías son imágenes, pero no todas las imágenes son fotografías. En el proceso de sensado, los sensores registran electrónicamente la energía como una matriz de números en formato digital.

Nuestros ojos son capaces de detectar una sección de longitudes de ondas en un rango visible, observable en la Figura 6.3, y el cerebro luego procesa esta información en colores separados. Los sensores sólo pueden ver rangos de longitudes de onda estrechos donde la información de estos rangos se recopila y almacena en un canal, también a veces referido como una banda. Luego se suele combinar y mostrar canales de información de forma digital utilizando tres colores cómo se mencionó anteriormente. Los datos de cada canal se representan como uno de los colores y, dependiendo del valor digital de cada píxel en cada canal, los colores rojos, azules y verdes se combinan en diferentes proporciones para representar otros colores distintos, formando una imagen a color.

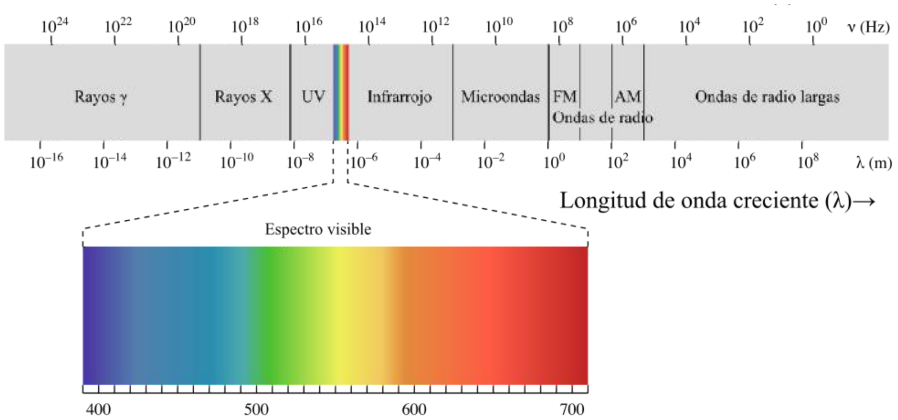


Fig. 6.3: Espectro electromagnético que incluye un diagrama de luz visible.

Teniendo en cuenta diferentes autores, la resolución de un sistema sensor se puede definir como la habilidad para registrar información de detalle [96]. Mientras que la disposición de los píxeles describe la estructura espacial de una imagen, las características radiométricas describen el contenido de información real en una imagen. Cada vez que una imagen es adquirida por un sensor, su sensibilidad a la magnitud de la energía electromagnética determina la resolución radiométrica. Por tal razón, esta resolución radiométrica describe la capacidad para discriminar diferencias muy leves de energía. Cuanto más fina es la resolución de un sensor, más sensible es a detectar pequeñas diferencias en la energía reflejada o emitida.

Otra resolución a tener en cuenta es la espectral, la cual indica el número y anchura de las bandas en las que el sensor puede captar radiación electromagnética. En principio cuanta más bandas incluya un sensor mejor, ya que cada banda constituye una variable para caracterizar la superficie captada.

Los datos de estas imágenes están representados por números digitales positivos que varían de 0 a una potencia dada de 2. Este rango corresponde al número de bits utilizados para codificar números en formato binario. En consecuencia, la cantidad máxima de niveles de brillo disponibles depende de la cantidad de bits utilizados para representar la energía registrada.

6.1.1. Elementos visuales

Nuestras implementaciones de clasificadores de escenarios urbanos aprenden a partir de las imágenes que componen el conjunto de entrenamiento. En estos datos se pueden observar los siguientes elementos visuales característicos de una imagen satelital [93][96]:

- Tono: se refiere al brillo o color relativo de los objetos en una imagen. Generalmente, el tono es el elemento fundamental para distinguir entre diferentes objetivos o características.
- Forma o morfología: se refiere a la estructura o contorno general del objeto en particular. Por un lado las formas de borde recto representan típicamente zonas urbanas o agrícolas (de campo), mientras que los escenarios naturales, como los bosques, contienen formas más irregulares.
- Tamaño: el tamaño de un objeto depende de la escala de la imagen. Es por ello que es importante evaluar el tamaño de un objetivo relativo a otros objetos en una escena.
- Patrón: se refiere a la disposición espacial de objetos visiblemente discernibles. Normalmente, una repetición ordenada de tonos y texturas similares producirá una patrón distintivo y finalmente reconocible. Algunos ejemplos para los escenarios urbanos son las zonas urbanas con calles y casas espaciadas de manera regular.
- Textura: es la frecuencia de variación tonal en áreas particulares de una imagen y es uno de los elementos más importantes para distinguir este tipo de imágenes. Las texturas ásperas se forman con niveles de grises que cambian abruptamente en una pequeña área, mientras que las texturas suaves tienen muy poca variación en el tono. Estas texturas suaves suelen ser el resultado de superficies uniformes como campos, asfalto o pastizales.
- Sombra: puede proporcionar una idea del perfil y altura relativa de un objetivo. Sin embargo, este elemento también puede reducir o eliminar la interpretación en su área de influencia, ya que los objetivos dentro de las sombras son mucho menos discernibles.
- Asociación: este elemento tiene en cuenta la relación entre otros objetos o características reconocibles en las proximidades del objeto de interés. Por ejemplo, un lago está asociado con botes y un posible puerto.

6.2. Sensores satelitales y aerotransportados

Un sensor es el aparato que reúne la tecnología necesaria para captar imágenes a distancia y para que pueda recopilar y registrar la energía que es reflejada o emitida desde un objetivo o superficie, el sensor debe residir en una plataforma estable que se encuentre lo suficientemente alejada del objetivo a observar.

El sensor en cualquier plataforma *ve* una cierta porción de la superficie. La zona de la superficie capturada se conoce como franja (*swath*) y se ve representada en la Figura 6.4. Esta ventana de observación puede variar entre decenas y cientos de kilómetros de ancho.

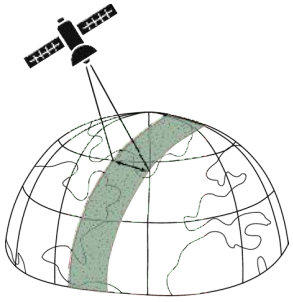


Fig. 6.4: Franja o ventana de observación de un sensor.

Una de las misiones espaciales que más suministros de imágenes satelitales ha proporcionado es el Landsat, que desde los años 70 viene formando un conjunto histórico. Estas señales son adquiridas a través de varios sensores a bordo de la flota Landsat con finalidades específicas [97]:

- Multispectral Scanner Sensor (MSS): eran dispositivos de escaneo de líneas que observaban la Tierra perpendicular a la trayectoria orbital. Este sensor tiene un rango espectral de 0.5 a 1.1 μm, una detección de cuatro bandas espectrales y un tamaño de imagen de 185×185 km. También utiliza una ventana de observación de 185 km.
- Thematic Mapper (TM): es un sensor de recursos terrestres de escaneo multiespectral avanzado diseñado para lograr una resolución de imagen más alta, una separación espectral más nítida, una fidelidad geométrica mejorada y una mayor precisión y resolución radiométrica que el sensor MSS. Los datos se detectan en siete bandas espectrales simultáneamente, también con una ventana de 185 km de tamaño.
- Operational Land Imager (OLI): instrumento de barrido multicanal que opera en nueve longitudes de onda en el rango de 0.433 a 2.300 μm y proporciona imágenes con una resolución máxima de 15 m. Proporciona dos nuevos rangos de longitud de onda, que son cruciales para la evaluación de la calidad de las aguas de lagos y zonas costeras.
- Thermal Infrared Sensor (TIRS): registra de 10.30 μm a 12.50 μm y proporciona la capacidad de realizar observaciones en dos combinaciones de 8 bandas de longitud de onda infrarroja. La resolución espacial de las imágenes obtenidas con el instrumento TIRS es de 100 m. Su objetivo principal es obtener las características de temperatura de la superficie y estudiar el proceso de transferencia de calor y humedad en el ámbito de la agricultura, la gestión del agua, etc.

Otro sensor a comparar es el que lleva los satélites Spot (*Satellite Pour l'Observation de la Terre*) de observación del suelo terrestre. Sus sensores de Alta Resolución Espacial, son similares en muchos aspectos a los MSS y TM mencionados anteriormente. Este sensor tiene una cobertura en tres bandas espectrales con una resolución espacial mucho más alta que el MSS, 20 m versus 80 m.

6.2.1. Sensores pasivos y activos

Uno de los criterios que se tienen en cuenta para clasificar los sensores remotos es la forma de recibir la energía procedente de las distintas cubiertas y en este sentido, existen dos tipos de sensores.

La mayor parte de los sistemas de teledetección capturan la energía que está naturalmente disponible ya sea provista por el Sol, el infrarrojo térmico o por la radiación de microondas natural de la Tierra, a los cuales se los denomina sensores pasivos. Esto tiene lugar cuando el Sol está iluminando la Tierra de día. Sin embargo, la energía que se emite naturalmente, cómo la infrarroja, se puede detectar ya sea de día o de noche.

A diferencia de los sensores pasivos los sensores activos proporcionan sus propias fuentes de energía para la iluminación, es decir que emiten de manera activa radiación que se dirige hacia el objetivo a investigar. La ventaja principal de estos sensores es la capacidad de obtener mediciones en cualquier momento, independientemente de la hora del día, la cobertura de nubes o la temporada.

Los sensores activos utilizan frecuencia en microondas de longitudes de onda diferentes a las que emite el sol. Sin embargo, estos sistemas requieren la generación de una gran cantidad de energía para iluminar adecuadamente los objetivos. Algunos ejemplos de sensores activos son el láser fluorosensor, el cual se utiliza para detectar películas de aceite muy delgadas, para clasificar el tipo de aceite y para analizar descargas deliberadas de petróleo en el mar. Otro ejemplo de sensores activos son los radares de apertura sintética o SAR del inglés Synthetic Aperture Radar.

6.3. Plataformas de teledetección

Los instrumentos de sensado remoto pueden ubicarse en una variedad de plataformas para ver y obtener las imágenes. Estas plataformas pueden estar situadas en el suelo, o alguna otra plataforma dentro de la atmósfera, cómo un avión, o en una nave espacial o satélite fuera de la atmósfera terrestre. La Tabla 6.1 realiza un análisis comparativo entre algunas de las plataformas de sensores disponibles hoy en día.

Factores a tener en cuenta para elegir entre distintas plataformas: el costo, área de superficie a recolectar información, frecuencia, la necesidad que sea necesario efectuar la recolección de datos en momentos programados por fuera de órbitas de satélites; todos ello son un factor importante a la hora de elegir entre las distintas plataformas disponibles. Hoy en día los satélites constituyen una de las plataformas más utilizadas. En este momento hay más de 4500 satélites en órbita alrededor de nuestro planeta, y más de 600 de ellos están recolectando información de gran variedad de sensores de manera constante.

6.3.1. Satélites, evolución

Los satélites son objetos que giran alrededor de otro objeto, en este caso, sobre la Tierra y esta tecnología está creciendo aceleradamente y el resultado de ello es la gran cantidad que existen en órbita.

La historia de los satélites cómo herramienta para la teledetección comienza en 1957 cuando la Unión Soviética lanza el primer satélite artificial de la historia, el Sputnik. El mismo año los soviéticos también lanzaron un satélite aún más masivo, el Sputnik 2, que transportaba al famoso perro, Laika. El primer satélite de los Estados Unidos fue el Explorer 1 el cual fue lanzado un año más tarde en 1958 y su aplicación era para las

	Características	Desventajas
Avión	Optimiza la adquisición de datos al proveer al operador con un amplio rango de parámetros de adquisición. Una misión de percepción remota puede realizarse sobre un área particular en un momento específico y puede ser repetido bajo condiciones controladas. Los aviones comerciales disponibles pueden alcanzar una altitud de 15 km. Utilizado principalmente para misiones críticas en el tiempo.	Inestabilidad de la plataforma, cobertura geográfica limitada debido a la relativamente baja altitud del avión, alto costo y dependencia de las condiciones ambientales.
Satélite	Tienen la capacidad de monitorear el total de la superficie de la Tierra sobre bases periódicas, cubriendo una sección suficientemente grande en cada revolución.	Los datos no son muestras directas del fenómeno e involucra mucho tiempo. Los datos deben corregirse geométricamente.
UAV	Un vehículo aéreo no tripulado, también conocido con el nombre de dron, es reutilizable y capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido. Este es propulsado por un motor de explosión, eléctrico o de reacción. Permite su uso en áreas de alto riesgo o de difícil acceso.	Los fenómenos físicos, como la actividad solar, mal tiempo o tormentas de rayos pueden provocar una influencia negativa en el funcionamiento. Tienen una capacidad de vuelo limitada por el tipo de combustible, fuente de energía, tamaño, alcance y su sistema de navegación. Alto coste de adquisición y mantenimiento.

Tab. 6.1: Distintas plataformas de sensores y sus características principales.

Ciencias de la Tierra. Este satélite tenía sólo un 2 por ciento de la masa del Sputnik 2, con 13 kg. Los Sputniks y el Explorer 1 se convirtieron en los primeros planos de una carrera espacial entre Estados Unidos y la Unión Soviética que duró al menos hasta finales de la década de 1960. En el año 1965 Asterix se convirtió en el primer satélite artificial de Francia. Tres años después tanto Japón cómo China lanzaron sus primeros satélites artificiales.

Otro hito importante en la evolución de los satélites en la historia son los Landsat, una serie de satélites construidos y puestos en órbita por Estados Unidos para la observación en alta resolución de la superficie terrestre. El primer satélite Landsat (en principio denominado ERTS-1) fue lanzado el 23 de julio de 1972, y el último de la serie es el Landsat 8, enviado al espacio en el año 2013. Los satélites de observación de la Tierra, como la serie Landsat, rastrean los cambios en los bosques, el agua y otras partes de la superficie de la Tierra a lo largo del tiempo.

Con respecto a la historia argentina, esta comienza en 1990 con el Lusat 1, el primer satélite argentino el cual fue un proyecto de radioaficionados. Luego le siguió Victor 1 en el año 1996, el cual fue el primer satélite artificial que fue concebido, diseñado, calificado e integrado en Argentina, con el propósito de ser un satélite experimental con fines educativos y como demostrador tecnológico. Llevaba a bordo dos cámaras para tomar imágenes de la Tierra, una de campo amplio y otra de campo estrecho.

Luego le siguieron los satélites SAC, los cuales tenían cómo misión el estudio y monitoreo del ambiente, el primero (SAC-A) lanzado en el año 1998 y el ultimo (SAC-D) en

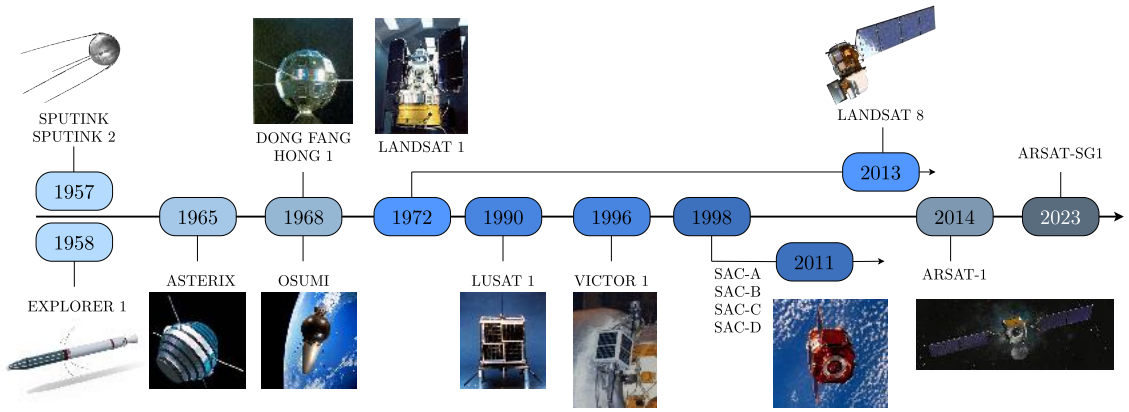


Fig. 6.5: Línea del tiempo de los primeros satélites artificiales y satélites argentinos.

2011. También le siguieron los satélites ARSAT, los cuales fueron fabricados totalmente en el país (ARSAT-1 en 2014 y ARSAT-2 en 2015). A estos le sigue ARSAT-SG1, conocido como ARSAT-3, satélite geoestacionario con fecha de lanzamiento para 2023, del cual se espera que ofrecerá una amplia gama de servicios de telecomunicaciones, tales como la transmisión de datos, Internet y televisión.

En términos de países con más satélites, Estados Unidos lidera significativamente el camino con 859 satélites, China ocupa el segundo lugar con 250 y Rusia el tercero con 146. A continuación, le siguen India (118), Japón (72) y el Reino Unido (52), mientras que desde 1990 la Argentina puso 17 satélites en órbita.

6.3.1.1. Órbitas

La trayectoria seguida por un satélite se conoce como órbita. Estas órbitas se adaptan a la capacidad de los sensores y al objetivo o superficie de interés. La selección de la órbita puede variar en términos de altitud, orientación y rotación relativa a la Tierra.

Los satélites ubicados en altitudes muy altas, de aproximadamente 36000 kilómetros en lo que se conoce como órbita de Clark, provocan que para un observador en la Tierra se lo vea como un punto fijo. Estas órbitas también denominadas geoestacionarias observan en todo momento la misma porción de la superficie terrestre y logran este resultado desplazándose a velocidades que coinciden con la rotación de la Tierra, y debido a la mecánica de la órbita, en esa altura en particular, la fuerza gravitacional es compensada por la fuerza centrífuga y de esta manera logran permanecer fijos en relación con la superficie. Cómo resultado se puede observar y recopilar información de forma continua sobre áreas concretas. Los satélites meteorológicos cómo el satélite GOES y de comunicaciones, cómo Intelsat, ARSAT, entre otros, suelen tener este tipo de órbitas.

La órbita geoestacionaria no es de las más utilizadas por los satélites de observación de la Tierra. En general se emplean órbitas polares con una inclinación del plano de órbita cercana a los 90 grados, relativo a una línea que corre entre los polos norte y sur, lo que permite que el satélite vea virtualmente cada parte de la Tierra. Estos satélites que tardan aproximadamente 90 minutos en completar una órbita viajan hacia el norte en un lado de la Tierra y luego hacia el polo sur en la segunda mitad de su órbita. Esto se conoce cómo pasajes ascendentes y descendentes, respectivamente. La mayoría de las plataformas satelitales actuales están en este tipo de órbitas. De los tres tipos de órbitas

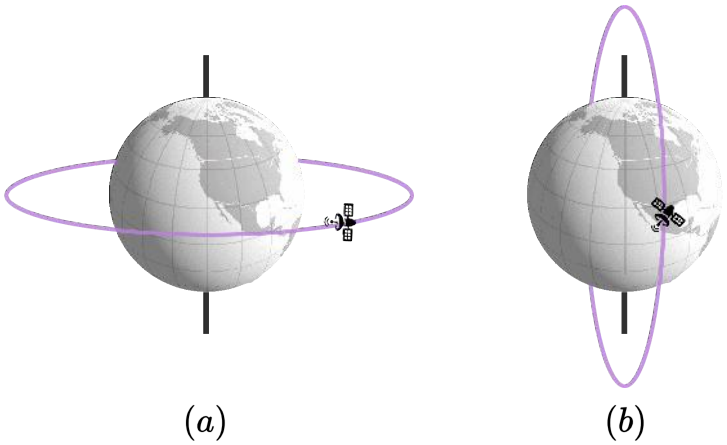


Fig. 6.6: (*a*) Órbita geoestacionaria ; (*b*) Órbita polar.

(órbita terrestre baja, media y alta), las órbitas polares a menudo caen en órbitas terrestres bajas.

Aproximadamente el 63 % de los satélites operativos están en una órbita terrestre baja, el 6 % están en órbita terrestre media (a 20.000 km), el 29 % están en órbita geoestacionaria (a 36.000 km) y el 2 % restante están en órbita elíptica.

6.4. Aplicaciones de la teledetección

Las aplicaciones de sensado remoto abarcan una gran cantidad de disciplinas y es transversal a muchas especialidades. A continuación, se detallan alguna de estas.

6.4.1. Agricultura, silvicultura y geologia

La agricultura juega un papel dominante en las economías de los países desarrollados y subdesarrollados. Gracias al uso del sensado remoto los agricultores pueden tener una visión general de sus cultivos junto a información de la salud y daños. Otro recurso valioso son los bosques, los cuales proporcionan alimento, refugio, hábitat de vida silvestre, combustible y suministros diarios además de jugar un papel importante en equilibrar el suministro e intercambio de CO_2 de la Tierra. Por último otra área que hace un gran uso de la teledetección es la geología, que basa sus estudios de peligros y catástrofes potenciales como la erupción de volcanes, deslizamientos de tierra y los terremotos. A su vez, los estudios geológicos no se limitan a la Tierra, el sensado remoto también se emplea para examinar la composición y estructura de otros planetas y lunas.

Las imágenes satelitales en todas estas áreas se emplean para [94]:

- Clasificación, evaluación del estado y salud del cultivo.
- Estimación del rendimiento de los cultivos.
- Mapeo de las características y prácticas de manejo del suelo.
- Mapeo de reconocimiento: incluye la actualización, el seguimiento del agotamiento y la medición de las propiedades biofísicas de rodales forestales.

- Monitoreo ambiental: las autoridades de conservación se preocupan por monitorear la cantidad, salud, y diversidad de los bosques de la Tierra.
- Mapeo de depósitos superficiales y lecho rocoso.
- Exploración y explotación de arena, minerales e hidrocarburos.
- Movimientos de terreno y desplazamientos por terremotos.

6.4.2. Uso de la tierra

Poder identificar, delimitar y mapear la cubierta terrestre es importante para estudios globales de seguimiento, gestión y planificación de recursos naturales. El término **uso de la tierra** o *land use* se refiere al propósito para el que la tierra sirve.

Las aplicaciones de uso de la tierra involucran tanto el mapeo como el monitoreo, ya que se requiere información oportuna para saber qué cantidad actual de tierra se encuentra en qué tipo de uso e identificar los cambios de uso de la tierra de un año a otro. Los estudios de cobertura y uso del suelo son de naturaleza multidisciplinaria y, por lo tanto, los participantes de tales trabajos son numerosos y variados entornos, que van desde investigadores gubernamentales y empresas forestales a fundaciones de conservación.

Además de facilitar la gestión sostenible del suelo, el uso del sensado remoto permite la obtención de información que puede usarse para planificar, monitorear y evaluar el desarrollo, la actividad industrial, o la recuperación. La detección de cambios a largo plazo en la cobertura terrestre puede revelar una respuesta a un cambio en las condiciones climáticas locales o regionales, la base de la globalización terrestre vigilancia.

Las aplicaciones de sensado remoto para el conocimiento del uso de la tierra incluyen las siguientes:

- Manejo de recursos naturales.
- Protección del hábitat de la vida silvestre.
- Expansión e invasión urbana.
- Planificación de rutas y logística para actividades sísmicas, de exploración y extracción de recursos.
- Delineación de daños producidos por catástrofes naturales como tornados, inundaciones, volcánicas, sísmicas, incendios.
- Detección de objetivos: identificación de pistas de aterrizaje, carreteras, claros, puentes, etc.

Finalmente en particular, nuestro trabajo entra dentro de la categoría de *land use* o uso de la tierra, donde implementamos CNNs con el propósito y aplicación de poder hacer una detección de escenarios urbanos.

Capítulo 7

Implementación de los Clasificadores

Se procedió a implementar la red de convolución AlexNet descrita en el Capítulo 3, y una nueva arquitectura basada en la anterior, conocida como AlexNet-SPP. Estas implementaciones fueron perfeccionadas en la tarea de clasificación de imágenes de alta resolución espacial (high spatial resolution) sensadas de manera remota. Para tal fin se utilizaron dos conjuntos de datos, por un lado la base de datos UC Merced Land Use [9] y el set WHU-RS [8].

El poder de generalización de un método de aprendizaje se mide en base a su capacidad de predecir frente a datos independientes de prueba. Alcanzar una buena performance es extremadamente importante ya que de ello depende tanto la selección del modelo como la posterior tasa de error final del mismo [98]. La construcción del modelo conlleva un arduo trabajo y para que este sea hecho de manera eficiente es crucial el proceso de validación del mismo. Para construir un modelo con buena generalización y rendimiento, se debe tener una estrategia de división de datos sensata [99].

Hay distintas maneras de evitar el overfitting, en este informe implementamos el mecanismo de early stopping explicado en la sección 7.2.3.1. También hay diversos métodos para estimar el error de prueba o test esperado de un modelo como las técnicas de cross validation y Montecarlo Cross Validation (MCCV) desarrolladas en las sección 4.4.1 y 4.4.2 respectivamente.

Es importante destacar que el presente capítulo conlleva dos partes diferentes pero dependientes e igual de importantes:

- La selección del modelo: estimar la performance de diferentes modelos en orden de escoger el mejor. Esto se desarrollará en las secciones de **Hiperparámetros**.
- La evaluación del modelo: habiendo pasado por el punto anterior, se estima el error de predicción o generalización sobre datos nuevos (conjunto de prueba o test), explicado en las secciones de **Resultados por escenario**.

7.1. Conjuntos de Datos

Un conjunto de datos o *dataset*, es una colección o recopilación de información que se corresponde al contenido de una tabla de base de datos única, donde cada columna de la tabla representa una variable particular y cada fila corresponde a un miembro o instancia. Cada instancia corresponde a uno de los datos de los que se disponen para hacer un análisis que a su vez, está compuesta de características que la describen. Luego en el aprendizaje

supervisado el conjunto de datos establecido tiene características etiquetadas que definen el significado de los datos.

En nuestro caso la tabla no sería bidimensional, sino que las filas son etiquetas de clase que agrupen a las imágenes que pertenecen a ellas. Entonces estamos trabajando con matrices con valores enteros entre 0 y 255, de tamaño $3 \times H \times W$ donde existen los tres canales de colores (R, G, B) y $H \times W$ es la altura y ancho de la imagen. Estas instancias están acompañadas de sus etiquetas correspondientes.

En la actualidad cualquier proyecto de machine learning requiere de estos conjuntos de datos como la materia prima para el modelo de predicción y su entrenamiento. El aprendizaje de estos modelos depende en primera medida de los datos ya que sin ellos es imposible que estos modelos aprendan. Entonces hablamos de que estos conjuntos son el aspecto más crucial que hace posible el entrenamiento de estas redes de convolución.

7.1.1. UC Merced Land Use

El conjunto de datos de la UC Merced Land Use publicado en 2010 [9], contiene 21 clases distintivas de imágenes ópticas (espacio de color RGB) de alta resolución que muestran distintos usos de la tierra. Para la formación de este conjunto fueron seleccionadas 2100 imágenes de 256×256 píxeles en total, 100 imágenes por clase, y se etiquetaron manualmente. Los datos fueron manualmente extraídos de la colección de imágenes disponibles públicamente del Servicio Geológico de los Estados Unidos (USGS) para varias áreas urbanas de USA [100].

Los datos de imágenes aerofotográficas provistos por el USGS generalmente son imágenes aéreas de alta resolución que combinan atributos visuales de una fotografía aérea con la precisión espacial de un mapa planimétrico (esto implica conseguir la representación a escala de todos los detalles interesantes del terreno sobre una superficie plana). Las imágenes digitales provistas varían en resolución de 15.24 cm a 1 metro y se ofrecen descargas gratis de dominio público cómo es el caso de la UC Merced Land Use.

Las clases de las que se compone este conjunto son las siguientes: agricultura, avión, campo de béisbol, playa, edificios, chaparral, zona residencial densa, bosque, autopista, cancha de golf, puerto, cruce, zona residencial media, parque de casa moviles, paso elevado, estacionamiento, rio, pista de aterrizaje, zona residencial esparcida, tanques de almacenamiento y cancha de tenis, cómo se muestran en la Figura 7.1.

También se puede observar que el conjunto de datos UC Merced tiene muchas clases similares o superpuestas, mostrando una diversidad inter-clase muy baja en algunas de las categorías que la componen, esto significa que comparten objetos similares además de texturas y patrones, lo que la hace un set de datos bastante desafiante [101]. Un ejemplo son las clase de zona residencial ('sparse residential', 'medium residential', 'dense residential'), donde también se ve reflejado claramente en las matrices de confusión de los resultados obtenidos en las secciones 7.2.4 y 7.3.4.

Todo esto además de la pequeña cantidad de muestras por clase hace que UC Merced sea una base de datos desafiante para la clasificación de imágenes y se refleja en los resultados obtenidos a través de las experimentaciones realizadas. Debido a su naturaleza y su resolución relativamente alta, estas imágenes comparten muchas características de bajo nivel con imágenes ópticas de propósito general, similares a las del conjunto ImageNet. Estas cuestiones las convierte en buenas candidatas para ser utilizadas con la técnica de fine-tuning en una red convolucional previamente entrenada. En los últimos años, muchos

investigadores han utilizado este conjunto de datos, lo que permite una amplia comparación de los resultados con la literatura [8][11][101][102].

7.1.2. WHU-RS

La base de datos WHU-RS [8] fue recolectada a partir de imágenes satelitales de Google Earth (Google Inc., Mountain View, CA, USA). Este conjunto de datos recientemente disponible de manera pública y gratuita, consiste en 950 imágenes con un tamaño de 600×600 píxeles uniformemente distribuidas en 19 categorías de escenas, es decir 50 imágenes correspondientes a cada categoría. La resolución espacial de estas imágenes de satélite es de hasta 0.5 m, y las bandas espectrales son roja, verde y azul.

Las clases incluyen aeropuerto, playa, puente, area comercial, desierto, tierras de cultivo, campo de fútbol, bosque, zona industrial, prado, montaña, parque, estacionamiento, estanque, puerto, estación de tren, zona residencial, río y viaducto. Ejemplos de muestras para estas clases se pueden observar en la Figura 7.2.

Una problemática de esta base de datos es la baja cantidad de muestras disponibles para realizar el entrenamiento, es por ello que se aplicó la técnica de Data Augmentation para sortear este inconveniente. Se puede observar que la variación de iluminación, escala, resolución y el punto de vista en algunas de las categorías de imágenes hace que este conjunto presenta complicaciones a la hora del entrenamiento, he incluso sea más complicada de entrenar a la red.

Fig. 7.1: Imágenes representativas de las 21 categorías de clases de la base de datos UC Merced.

Fig. 7.2: Ejemplos representativos de las categorías que conforman la base de datos WHU-RS.

7.2. Modelo AlexNet

7.2.1. Arquitectura

Se utilizó el modelo pre-entrenado de AlexNet propuesto por Alex Krizhevsky et al. en el 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2012) [4]. La idea es utilizar esta arquitectura para mejorarla en la tarea de clasificación para escenarios urbanos, por ser una de las redes convolucionales clásicas con gran protagonismo en la bibliografía. Esto es posible ya que las características aprendidas en las capas profundas de cualquier red convolucional pre-entrenada, como bordes o manchas de color, son lo suficientemente generales para ser trasladadas a otras bases de datos. El éxito de esta estrategia depende de numerosos factores, siendo el más importante la diferencia entre la tarea original para la cual la red fue entrenada y el nuevo enfoque que se le desea dar.

Como se explico en el Capítulo 3, AlexNet se compone de cinco capas de convolución, donde la distribución comienza con las primeras dos capas convolucionales, seguidas por una capa de agrupación (pooling) cada una, luego tres capas de convolución, otra capa de pooling y tres capas completamente conectadas (ver Figura 7.3. Para la arquitectura AlexNet, los filtros convolucionales se extraen durante el procedimiento de retropropagación del error optimizando la función de costo.

Generalmente, las capas convolucionales actúan sobre los mapas de características de entrada con los filtros convolucionales deslizantes para generar los mapas de características, estos, al pasar por las capas de pooling se alteran de manera que se extrae la información más importante con una operación de agrupación máxima o la operación de agrupación promedio.

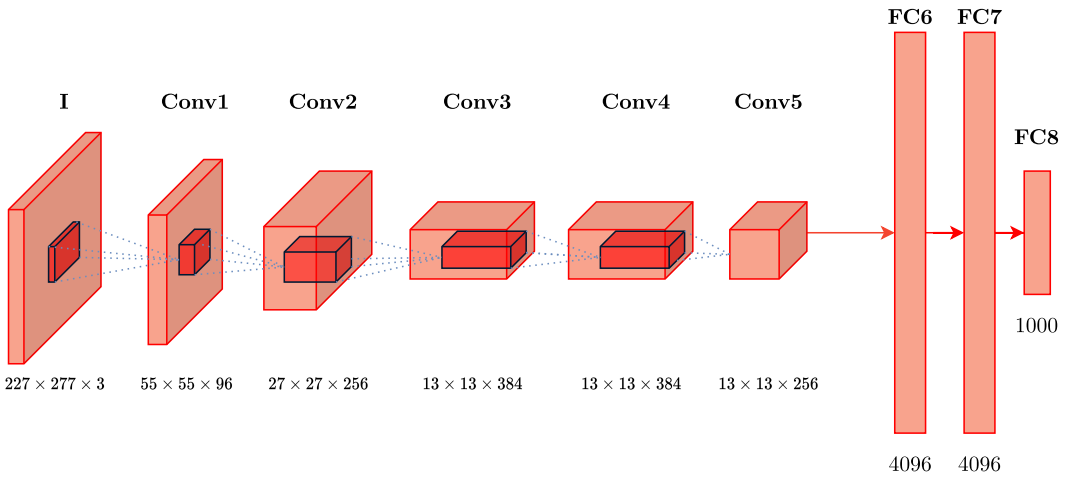


Fig. 7.3: Arquitectura original AlexNet.

La red está configurada para aceptar entradas de 227×227, y en el paper original [4] figura un tamaño de 224×224 píxeles. Realizando un analisis se puede deducir que el tamaño de entrada 224×224 es inapropiado al considerar que la primera convolución consta de 96 filtros de 11×11 con un stride de 4 y 0 de padding. Al evaluar si la imagen se ajusta a estas dimensiones nos damos cuenta que los filtros no cubren apropiadamente la entrada de 224×224 debido a que la ecuación :

$$volumenDeSalida = (tamImagen - tamFiltro)/stride + 1$$

da un número no entero por lo tanto las neuronas no cubren ordenada y simétricamente la entrada [103]. Esta situación es muy restrictiva y numerosas veces dicho condicionamiento provoca la pérdida de información útil para el entrenamiento. Esto se vio resuelto cuando se optó por cambiar las capas de pooling luego de la quinta convolución por un módulo de Pooling Piramidal Espacial (SPP), definiendo así una nueva arquitectura utilizada en la sección 7.3.

7.2.2. Hiperparámetros

Antes de comenzar con el entrenamiento de los escenarios a plantear, se realizaron distintas pruebas para seleccionar el modelo que mejor se comportaba a nuestro problema en particular utilizando la base de datos UC Merced Land Use.

7.2.2.1. Determinación inicial

Los parámetros de ajuste o hiperparámetros están estrictamente vinculados a la complejidad del modelo. Una mala elección de estos nos puede llevar a una red en donde el entrenamiento se dificulta excesivamente. Lo ideal es encontrar aquellos que minimicen el error.

En nuestro caso los hiperparámetros asociados a la arquitectura de la red están definidos ya que hicimos uso de una red pre-entrenada, en este caso AlexNet. No tendría sentido modificar estos valores si queremos analizar el rendimiento de este modelo con el conjunto de UC Merced Land Use. Para nuestro estudio nos van a interesar establecer los hiperparámetros relacionados al proceso y a los algoritmos de aprendizaje.

Para sortear esta problemática y lograr la elección inicial de hiperparámetros, se realizaron 162 pruebas en total a través de una búsqueda por grilla. Se tomó un 90 % del conjunto total para entrenamiento y un 5 % como conjunto de validación. Naturalmente, se ajustó a la red pre-entrenada de AlexNet utilizando transfer learning, y, en esta etapa de la implementación, se seleccionó el método del descenso del gradiente estocástico durante 40 épocas variando los hiperparámetros: tasa de aprendizaje, momentum y tamaño de lote.

El subconjunto del espacio de hiperparámetros definido para la búsqueda sobre cada combinación posible de hiperparámetros, es decir la grilla, es el siguiente:

- Tamaño de lote = [1, 2, 4, 8, 16, 32, 64, 128, 256]
- Tasa de aprendizaje = [0.1, 0.01, 0.005, 0.001, 0.0005, 0.0001]
- Momentum = [0.9, 0.95, 0.99]

En el paper [11] se sostiene la idea de que hay un vínculo entre el tamaño de batch y la tasa de aprendizaje. Específicamente se demuestra que decrementando esta última se obtiene el mismo efecto que incrementando el tamaño de batch. Nuestros resultados arrojaron los peores valores para actualizaciones consecutivas, es decir, modificar los parámetros de la red siempre que se presente una muestra. Sin embargo, como se comenta en [11], a menores tasas de entrenamiento y tamaños de batch con valor igual a 1, sí se notaron

N°	batch_size	learning rate	momentum	train_acc	val_acc	train_loss	val_loss
1	256	0.005	0.9	100.00 %	95.24 %	0.00072	0.306
2	256	0.001	0.99	99.15 %	95.24 %	0.02315	0.245
3	**16**	**0.0001**	**0.95**	**100.00 %**	**95.24 %**	**0.00247**	**0.166**
4	32	0.0001	0.99	99.05 %	95.24 %	0.02358	0.250
5	64	0.0001	0.99	99.84 %	95.24 %	0.00436	0.242
6	256	0.01	0.9	100.00 %	94.29 %	0.00080	0.277
7	2	0.0001	0.95	100.00 %	94.29 %	0.00024	0.216
8	4	0.0001	0.95	100.00 %	94.29 %	0.00071	0.241
9	16	0.0005	0.9	99.95 %	94.29 %	0.00253	0.170
10	32	0.001	0.95	99.95 %	94.29 %	0.00315	0.241
11	32	0.0005	0.9	100.00 %	94.29 %	0.00124	0.208
12	64	0.001	0.95	99.05 %	94.29 %	0.03030	0.326
13	128	0.0001	0.99	99.84 %	94.29 %	0.00263	0.218

Tab. 7.1: Los mejores resultados de las 162 pruebas llevadas a cabo mediante una búsqueda por grilla.

resultados llamativos, alcanzando el 91.4 % para el conjunto de validación con 0.0001 de tasa de aprendizaje que se excluyeron de la Tabla 7.1 por cuestiones de espacio.

A medida que los tamaños de batch aumentaban, la performance, en general, mejoraba. Exceptuando el caso del valor 0.1, el cual no está expuesto en la Tabla 7.1 porque demostró ser insatisfactorio en todas las pruebas, dando resultados inexactos alejándose muchísimo de lo esperado del modelo.

Alex Krizhevsky [4] entrenó a AlexNet desde cero con un tamaño de batch de 128 imágenes, momento de 0.9 e inicializando la tasa de aprendizaje a 0.01. Esta última era disminuida manualmente al dividirla por 10 cada vez que el error de validación se estancaba. Dentro de nuestra búsqueda por grilla, correspondiente a la prueba número 130, se sometieron a pruebas estos valores mencionados (128; 0.01; 0.9) para ver cómo se comparaba con nuestra implementación. En esa prueba, manteniendo la tasa de aprendizaje constante, alcanzamos resultados buenos, con una precisión de validación de 93.33 %.

Sin embargo el mejor resultado de la prueba se dió para una tasa de 0.0001 con $momentum = 0{,}95$, obteniendo un 93.33 % de exactitud en la validación y un 97.14 % de exactitud en el set de prueba. Esto tiene sentido ya que estamos utilizando la red preentrenada, entonces, en concordancia con la aplicación de fine-tuning, la tasa de aprendizaje debe ser mantenida a valores bajos, ya que no estamos llevando a cabo un aprendizaje radical. Además, esto tiene sentido debido a que las redes convolucionales entrenadas en el set ImageNet poseen fuertes similitudes de bajo nivel con las imágenes ópticas de la UC Merced [104].

7.2.2.2. Selección final

Cuando se evalúan diferentes opciones de hiperparámetros hay un riesgo de sobreentrenamiento en el conjunto de validación debido a que los parámetros son ajustados hasta que los resultados son buenos. De esta manera, el conocimiento del conjunto de prueba puede filtrarse al modelo, y luego las métricas de evaluación ya no reportan de manera fehaciente el rendimiento en cuanto a generalización.

Una solución a este problema es realizar el procedimiento de validación cruzada de k iteraciones, mencionado en la sección 4.4.1, donde un conjunto de prueba sigue siendo separado para la evaluación final. Para la selección final de parámetros, hicimos uso de la

técnica de validación cruzada con $k = 10$, donde 9 folds o 90 % del conjunto de datos es utilizado para el entrenamiento.

Para ello tomamos los trece mejores conjuntos de hiperparámetros obtenidos en la búsqueda por grilla previa, los cuales se pueden observar en la Tabla 7.1. Por último el rendimiento se reporta realizando un promedio de los valores obtenidos en las $k = 10$ iteraciones, contenidos en la Tabla 7.2.

batch size	lr	momentum	train_acc	train_loss	val_acc	val_loss	time[min]
256	0.005	0.9	97.70 %	0.06816	90.32 %	0.31452	115.8
256	0.001	0.99	88.31 %	0.36848	83.70 %	0.54736	116.4
16	**0.0001**	**0.95**	**98.08 %**	**0.06666**	**92.22 %**	**0.26394**	**113.8**
32	0.0001	0.99	96.05 %	0.12322	89.37 %	0.36073	110.8
64	0.0001	0.99	97.52 %	0.07881	89.47 %	0.37478	99.5
256	0.01	0.9	97.26 %	0.08072	89.84 %	0.33773	108.1
2	0.0001	0.95	95.80 %	0.12846	89.89 %	0.32275	282.7
4	0.0001	0.95	98.98 %	0.03215	89.74 %	0.30737	109.1
16	0.0005	0.9	98.73 %	0.04064	91.27 %	0.29289	86.2
32	0.001	0.95	96.75 %	0.10440	90.11 %	0.32746	82.9
32	0.0005	0.9	98.45 %	0.05556	90.05 %	0.30749	83.8
64	0.001	0.95	97.76 %	0.06840	91.64 %	0.30440	76.3
128	0.0001	0.99	95.40 %	0.14800	88.15 %	0.40027	79.4

Tab. 7.2: Resultados promedio de entrenamientos haciendo uso de los mejores hiperparámetros con validación cruzada 10-fold.

Finalmente luego de la obtención de los resultados de validación y entrenamiento, y una siguiente examinación, los hiperparámetros óptimos a tener en cuenta son:

- Tamaño de lote = 16
- Tasa de aprendizaje = 0.0001
- Momentum = 0.95

7.2.3. Entrenamiento

El procedimiento para lograr una buena clasificación consistió en transferir el entrenamiento de AlexNet, es decir se empleó la técnica de *transfer learning* descrita en la sección 3.6, y luego se realizó fine-tuning en la base de datos de interés. Esto es mucho más rápido que entrenar la red desde cero [11]. También se mantuvieron descongeladas todas las capas de la arquitectura de la red. Esto se debe a que en pruebas preliminares realizadas se obtuvieron mejores resultados cuando todas las capas aprendían la nueva tarea logrando, de esta manera, una mejor adaptación para las señales de entrada utilizadas.

Para realizar el entrenamiento se optó por utilizar la librería de DL PyTorch comparada en la sección 5.8.2. Al comienzo se intentó trabajar con con la librería TensorFlow para el modelado de redes convolucionales, pero debido a que esta no provee de un modelo de AlexNet con los pesos pre-entrenados se optó por esta primera alternativa. Finalmente, se utilizó la API provista por esta librería para Python y se corrió sobre la plataforma web Colaboratory usando aceleración por hardware de GPU.

Al llevar a cabo el proceso de entrenamiento de una red se debe ir evaluando el desempeño de la misma en la tarea de clasificación, haciendo uso de una función de costo. Esto es importante debido a que con la misma se actualizan los pesos del modelo en la etapa de

retropropagación. En nuestro caso utilizamos el criterio de la entropía cruzada categórica o cross entropy, medida de precisión para variables categóricas.

Este error, conocido también como logarítmico, mide el rendimiento del modelo de clasificación cuya salida es un valor de probabilidad entre 0 y 1, y es indicativo de cuánto ha aprendido el modelo a partir de nuestras muestras. El error de entropía cruzada se incrementa a medida que la probabilidad predecida diverge de la verdadera etiqueta o clase de la muestra, penalizando con valores cercanos a uno según el nivel de error. Luego se esperaria de un modelo muy bueno valores de error cercano a 0. Se calcula cómo [24]:

$$CrossEntropyLoss = -\sum_{i=1}^{n} L_i \cdot log(S_i)$$

donde S_i el valor de probabilidad predecida, L_i es el valor real y se aplica el logaritmo en base 2. Por ejemplo, suponiendo un problema con 3 clases, si a partir de una muestra se predice por la salida de la capa softmax $S = [0.7, 0.2, 0.1]$ siendo la etiqueta original $L = [1, 0, 0]$, entonces el error se determina de la siguiente manera:

$$CrossEntropyLoss = -(1 \cdot log(0.7) + 0 \cdot log(0.2) + 0 \cdot log(0.1)) = 0.5145$$

Otra problemática surgida del entrenamiento es que los conjuntos de imágenes trabajados son algo pequeños, y es por esto que se adoptó una táctica para sortear las dificultades que esta situación podría conllevar. En este sentido fue determinante para alcanzar los resultados obtenidos la utilización de la conocida técnica de Data Augmentation descrita en la sección 3.4.5. Esto nos permitió aumentar la cantidad de muestras que utilizamos para generar los sets de datos, logrando un entrenamiento más enriquecedor.

7.2.3.1. Early Stopping

Se establecieron varios criterios a la hora de decidir en qué momento el proceso de entrenamiento se detiene. Entonces nuestra primera condición de parada es que el error de entrenamiento llegue a cierto valor bajo prefijado, en nuestro caso, 0.01. Este error se calcula haciendo la división entre cantidad de imágenes clasificadas correctamente sobre el número total de muestras utilizadas.

El error de entrenamiento no es el único indicativo para detener el proceso de entrenamiento del modelo. Prestar atención a esto solamente puede llevarnos a sobreentrenar al mismo, logrando así que el modelo pierda la capacidad de generalizar frente a nuevas muestras. Es por ello que a su vez utilizamos el conjunto de validación para determinar cuándo corresponde parar definitivamente el entrenamiento, estrategia llamada early stopping. La curva de validación describe los errores de validación o generalización obtenidos a lo largo del entrenamiento del modelo. Existen varios criterios de early stopping y la mayoría están basados en el uso de la curva de validación [24][105].

Un criterio es detener el entrenamiento del modelo cuando el error de validación no ha mejorado, comparado con el mejor error, por cierta cantidad de épocas [24][105]. Otro criterio adopta la idea de detener el entrenamiento cuando el error de validación es más bajo por cierto margen, que el error de validación mínimo registrado, mientras que el error de entrenamiento ya se haya estancado. Otros cómo Duvenaud et al [106] y Mahsereci et al [107] proponen criterios basados en cómputos de gradientes y estimación de márgenes logarítmicos, logrando así la no necesidad del uso de un conjunto de validación.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Fig. 7.4: Pseudocódigo del algoritmo de early stopping para determinar el tiempo óptimo de entrenamiento. [24, Algoritmo 7.1]

En *Early Stopping - But When?* [105] se definen tres criterios de detención los cuales han sido probados en nuestra implementación dando resultados no favorables: el entrenamiento paraba en alguna de las primeras 10 épocas, que es demasiado pronto. Analizando los resultados obtenidos se decidió adoptar una estrategia diferente a los criterios mencionados, pero que tiene algunos puntos en común. Lutz Prechelt [105] sostiene que hay ocasiones en las el error de entrenamiento está progresando rápidamente mientras que los errores de generalización demuestran lo contrario, entonces considera que el error de validación se corregirá en el momento en el que el entrenamiento empieza a decrecer lentamente. Es en este último momento en el que comienza el peligro de overfitting y es ahí en donde hay que guiarse por el error de generalización.

Luego de realizar y analizar numerosas pruebas se optó por activar el early stopping cuando el error de entrenamiento decae a 0.001. Basándonos en el pseudocódigo extraído del Deep Learning Book (ver Figura 7.4)[24], se determina una paciencia de 5 y un delta mínimo de 0.001 [13]. El delta mínimo determina un margen en base al cual se determina si ha ocurrido una mejora, donde un cambio menor que delta mínimo no cuenta como progreso. La paciencia define el número de épocas en las cuales no ocurrió una mejora, luego una vez que se supera estas épocas de paciencia el early stopping se activa y el proceso de entrenamiento es detenido.

7.2.4. Resultados por escenario

Se define el *escenario A* a partir de los resultados obtenidos para el conjunto de datos de la UC Merced Land Use, caracterizada en la sección 7.1.1.

Como se menciono anteriormente en la sección 4.5, la precisión de clasificación *accuracy* se mide mediante la ecuación $A = N_c/N_t$, donde N_c denota el número de muestras clasificadas de manera correcta en el conjunto de prueba y N_t define el número total

de muestras en dicho subconjunto. Evaluamos el rendimiento final de clasificación con el promedio de precisión obtenido en las corridas de entrenamiento.

7.2.4.1. Escenario A.1

Se comenzó dividiendo el total de muestras en diez particiones iguales, es decir, $k = 10$ con el objetivo de aplicar la técnica de validación cruzada $k-$fold. De esta manera se tomó una partición para prueba, media para validación y los restantes para entrenamiento, logrando entonces un 10 % para prueba, 5 % para validación y 85 % para entrenamiento. Entonces, de las 2100 imágenes se reservan 210 para el conjunto de prueba y 1890 para entrenar, dentro de las cuales 105 son para validar el modelo.

En este escenario se optó por activar la técnica de early stopping con los criterios descritos en la sección 7.2.3.1.

Para incrementar la diversidad de las imágenes censadas se utilizó la técnica de data augmentation incorporando la rotación en ángulos aleatorios en el rango $-45°$ a $45°$ sumado a una rotación horizontal completa con probabilidad de ocurrencia de 0.5. Las capas de todo el modelo se mantuvieron descongeladas en todo el proceso de entrenamiento.

Se utilizaron los hiperparámetros obtenidos en la sección 7.2.2.2, valor inicial de tasa de aprendizaje de 0.0001 y un momentum de 0.95, haciendo uso del optimizador SGD. El entrenamiento se realizó por 100 épocas, con un tamaño de lote de 16 muestras. Con 10 muestras por clase para conformar el conjunto de prueba, manteniendo equilibrio entre clase, se obtiene un resultado promedio de $\mathbf{(90.5 \pm 2.2)\,\%}$ de precisión en el conjunto de prueba. Los resultados de las 10 iteraciones se encuentran en la Tabla 7.3.

ki	época parada	train acc	train loss	validation acc	validation loss	test acc	
0	53	99.89 %	0.00857	91.43 %	0.43572	91.43 %	
1	61	99.50 %	0.01737	96.19 %	0.08036	90.48 %	
2	64	99.44 %	0.01537	94.29 %	0.21460	90.00 %	
3	70	99.61 %	0.01212	91.43 %	0.51899	88.57 %	
4	58	99.78 %	0.00996	91.43 %	0.31333	92.86 %	
5	48	99.89 %	0.00922	96.19 %	0.09082	95.24 %	
6	62	99.61 %	0.00873	93.33 %	0.17681	88.10 %	
7	60	99.72 %	0.00922	91.43 %	0.17765	88.57 %	
8	57	99.16 %	0.01681	96.19 %	0.20878	89.05 %	
9	60	99.72 %	0.00744	92.38 %	0.30532	90.95 %	
Final	-	99.63 %	0.01148	93.43 %	0.25224	**90.5 %**	**2.2 %**

Tab. 7.3: Resultados obtenidos para el escenario A.1.

Se puede observar de la matriz de confusión (ver Figura 7.5) que las clases más conflictivas ya mencionadas anteriormente que son las zonas residenciales.

7.2.4.2. Escenario A.2

Debido a que en la prueba anterior (*Escenario A.1 - sección 7.2.4.1*) se observó que para las épocas de entrenamiento, el error de validación se mantenía fluctuando en un rango acotado y que además las muestras del conjunto de validación eran muy pocas, se optó en este nuevo escenario, no utilizar un conjunto de validación, logrando así también aumentar las muestras utilizadas para el entrenamiento, las cuales son escasas.

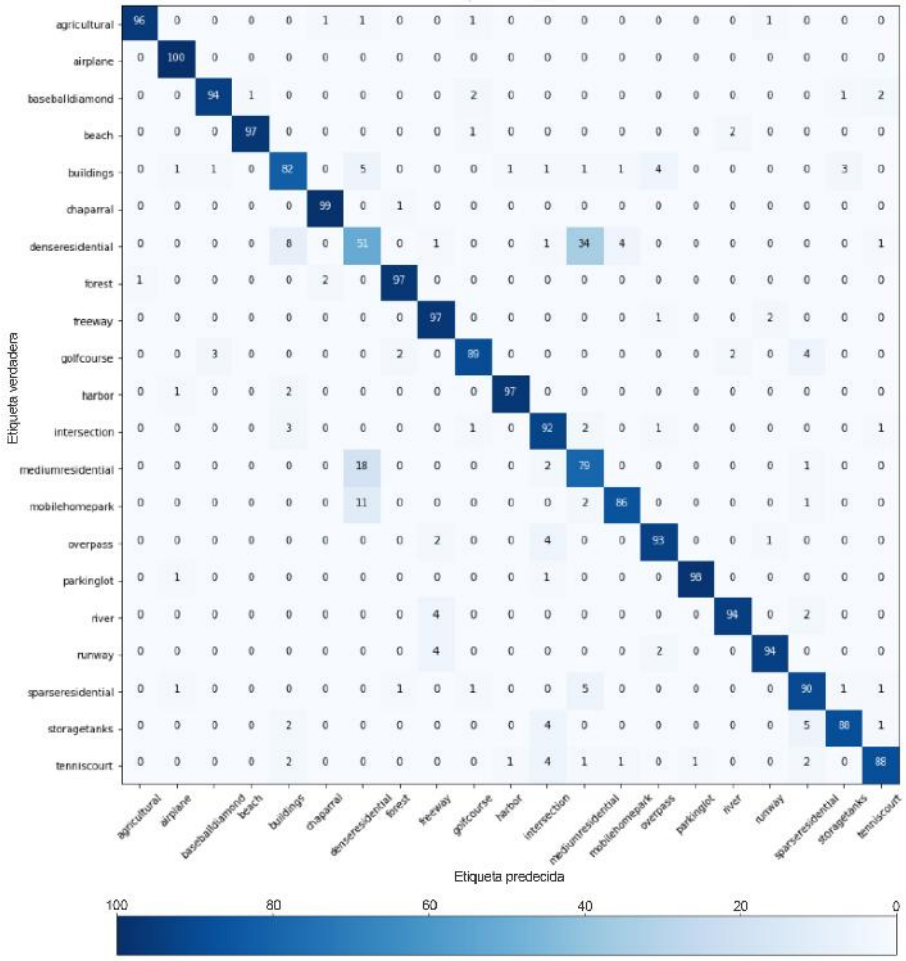


Fig. 7.5: Matriz de confusión total para el escenario A.1.

En base a que ya se tenía un conocimiento de la performance de la red, considerando las numerosas pruebas realizadas previamente, podíamos asegurar que la misma no iba a diverger. Aplicando esta técnica se emplean un 90 % de muestras para entrenar y un 10 % para el proceso de prueba o test, con $k = 10$.

Se utilizaron las mismas técnicas de data augmentation y se corrió cada entrenamiento por 150 épocas. Al igual que en el escenario anterior (*sección 7.2.4.1*) todas las capas se mantuvieron descongeladas para todo el proceso de entrenamiento.

También se mantuvo el uso del optimizados SGD con los hiperparámetros definidos por la búsqueda por grilla de la seccion 7.2.2.2. Con 10 muestras por clase (manteniendo equilibrio entre clase) para conformar el conjunto de prueba se obtiene un resultado promedio superador de $\mathbf{(92.2 \pm 1.7)\,\%}$ de precisión. Los resultados de las 10 iteraciones se encuentran en la tabla.

ki	época parada	train acc	train loss	test acc	
0	91	99.79 %	0.00391	92.86 %	
1	89	99.89 %	0.00405	90.00 %	
2	99	99.89 %	0.00255	92.38 %	
3	90	99.79 %	0.00487	92.86 %	
4	89	99.95 %	0.00282	94.29 %	
5	90	99.95 %	0.00434	94.76 %	
6	79	100.00 %	0.00375	90.48 %	
7	98	100.00 %	0.00114	90.95 %	
8	97	99.89 %	0.00402	90.48 %	
9	96	100.00 %	0.00136	92.86 %	
Final	-	99.92 %	0.00328	**92.2 %**	**1.7 %**

Tab. 7.4: Resultados obtenidos para el escenario A.2.

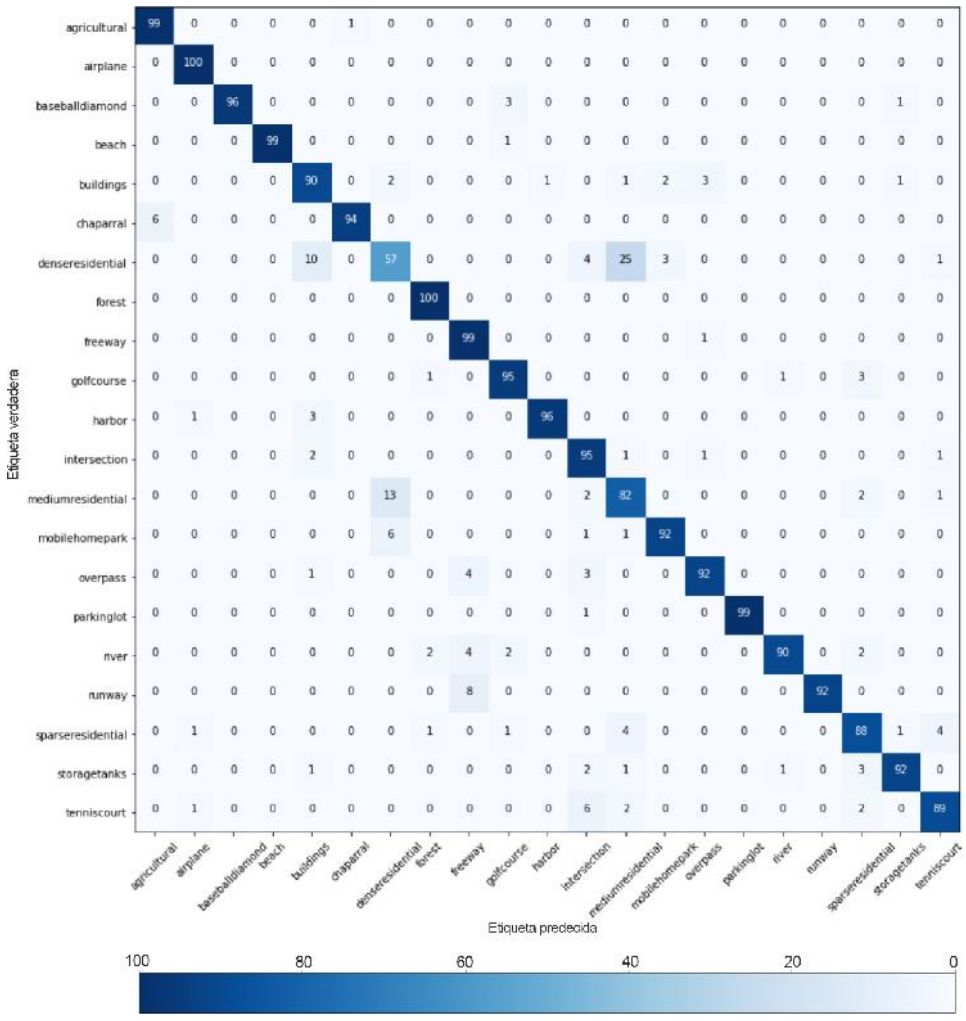


Fig. 7.6: Matriz de confusión total para el escenario A.2.

7.3. Modelo AlexNet+SPP

7.3.1. Arquitectura

Las redes neuronales clásicas aceptan entradas de un tamaño prefijado. Este condicionamiento no se debe a las convoluciones de las capas iniciales, ya que ellas generan mapas de características independientemente del tamaño de las señales. Solamente basta recorrer todo el volumen de la imagen con los filtros de la capa en cuestión. El problema reside en las capas totalmente conectadas o FC, las cuales necesitan una entrada de un tamaño determinado, es decir que admiten un vector de dimensiones fijas, ya que en ellas se realiza un producto escalar, operación que implica la restricción comentada y obliga a modificar la imagen original para que cumpla los requerimientos de estas últimas capas.

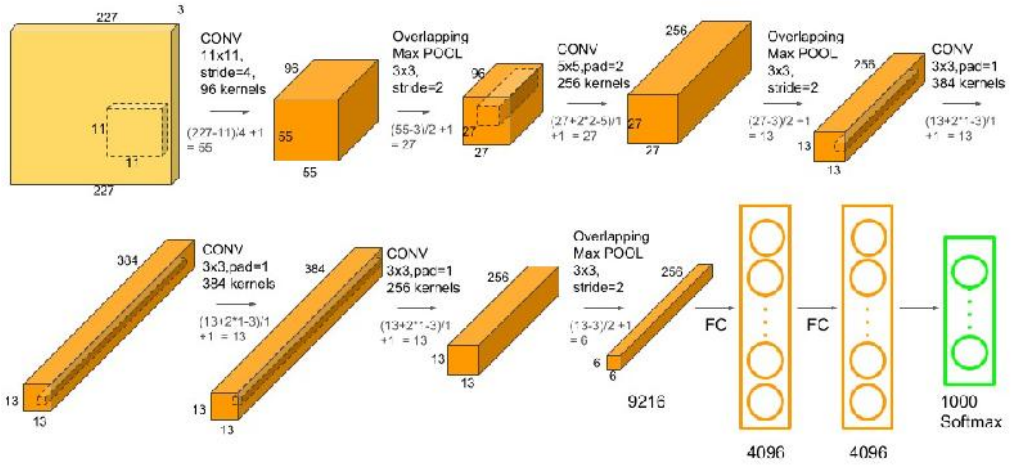


Fig. 7.7: Arquitectura de AlexNet detallada en donde se observan las dimensiones de entrada y salida de las capas que componen el modelo.

En la arquitectura de AlexNet luego de la quinta convolución se encuentra una capa de max pooling que espera como entrada los mapas de convoluciones de $13 \times 13 \times 256$ y aplicará la operación de pooling como una ventana deslizante de dimensiones de $3 \times 3 \times 256$ con un stride de 2. Luego se obtendrán 256 mapas de características de $6 \times 6 (6 \times 6 \times 256)$. Seguida de ella hallamos a la primera capa totalmente conectada. Esta última espera un vector unidimensional de 9216 elementos obtenidos de aplanar la salida de la capa de max pooling. Aquí es donde subyace la problemática que impide entrar a la red con imágenes de tamaño variable. Esta capa necesita una entrada de tamaño 9216×1, la cual únicamente se consigue alimentando a la red con imágenes de tamaño 227×227. Para sortear este inconveniente se intercambia la capa de max pooling luego de la quinta convolución por el módulo de *pooling piramidal espacial*.

El Pooling Piramidal Espacial (SPP), añade una nueva capa entre las convoluciones y previo a las capas totalmente conectadas. Es su trabajo mapear cualquiera sea el tamaño de la entrada a una salida de tamaño fijo. Este modulo SPP divide los mapas de características de la última convolución, la cual en AlexNet es la convolución número 5, en un número de bins espaciales con tamaños proporcionales al tamaño de la imagen. Entonces, el número

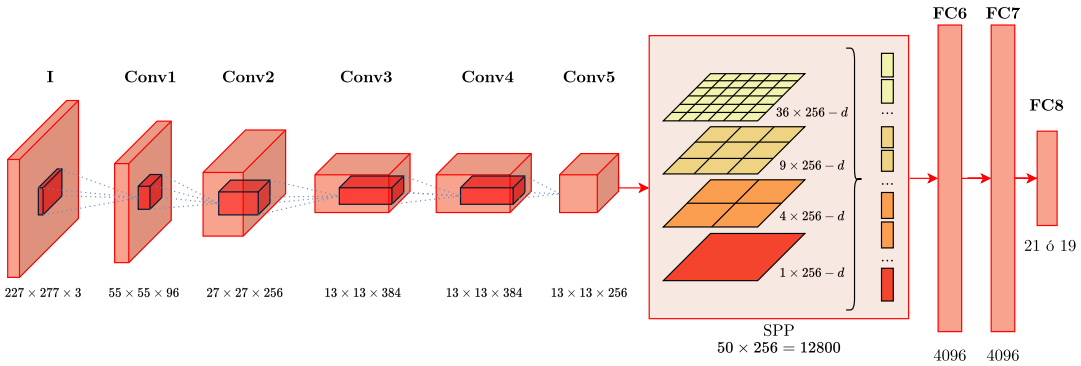


Fig. 7.8: Arquitectura propuesta AlexNet con uso del modulo SPP.

de bins es fijo a pesar del tamaño de la entrada. Los bins se capturan en diferentes niveles de granularidad, por ejemplo una capa de 16 bins divide la imagen en una grilla de 4×4, otra capa de 4 bins la particiona en 2×2 y una capa final comprende la imagen en su totalidad. En cada bin espacial a las activaciones de cada filtro se les aplica max pooling.

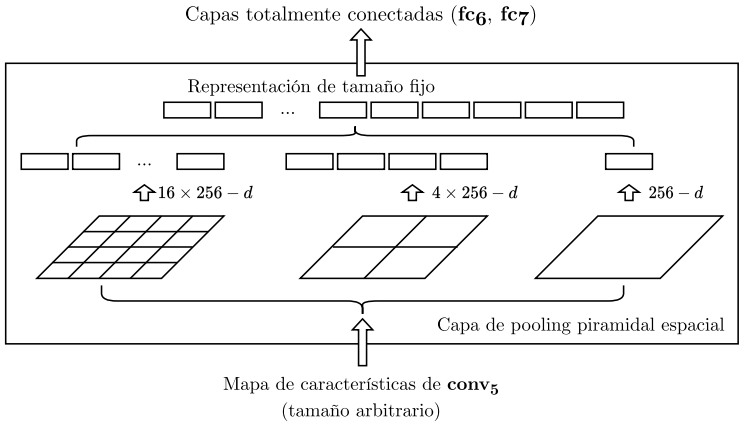


Fig. 7.9: La estructura de una red con una capa de pooling espacial piramidal.

Como el número de bins es conocido, se puede concatenar las salidas del módulo SPP para brindar una representación de tamaño fijo y así obtener la entrada necesaria para las capas FC, evitando de esta manera la pérdida de información. En la figura 7.9, se utiliza un SPP de tres niveles donde la última capa de convolución entrega 256 mapas de características, en el módulo de SPP:

1. A cada mapa de características se le realiza un pool extrayendo un único valor y formando un vector de $256 - d$.

2. Se aplica otro pool de 2×2 a cada mapa de características y se obtienen cuatro valores por cada uno de estos, luego se forma un vector de $4 \times 256 - d$.

3. Nuevamente, a cada mapa de características se le extraen 16 valores formando un vector de $16 \times 256 - d$.

4. Los tres vectores que se obtienen del paso anterior se concatenan para formar un

vector de una dimensión. Finalmente este vector $1 - d$, el cual ya no tiene una dimensión de 9216, es la entrada a las capas FC.

No implementamos ni el pooling clásico ni el que acabamos de explayar de 3 niveles, sino que dividimos el espacio en cuatro, generando un pooling de cuatro niveles de 1, 4, 9 y 36 bloques también propuesto por los diseñadores de este modulo [12] y aplicado en otros trabajos [108][109], de modo que se obtienen 50 subregiones y se logran extraer 50 características. Luego se alimentan las capas totalmente conectadas con las salidas obtenidas.

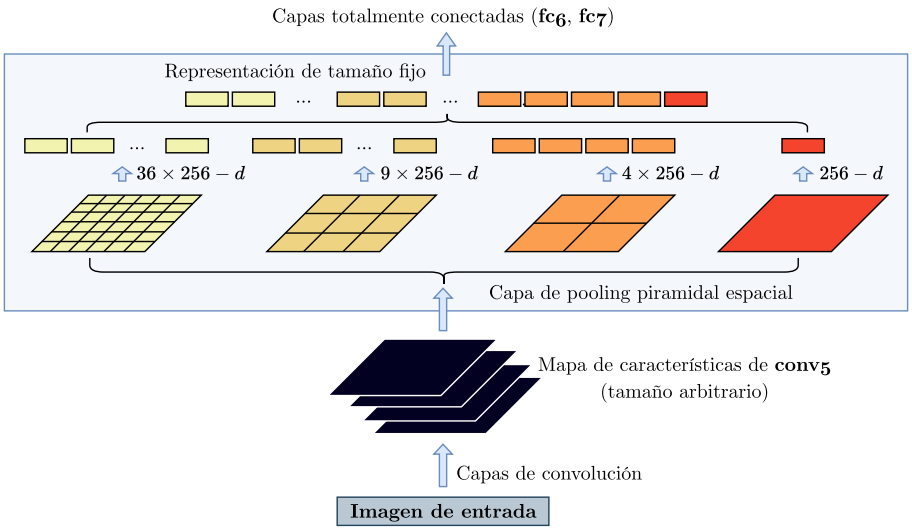


Fig. 7.10: La estructura propuesta para la experimentación con una capa de pooling espacial piramidal.

Es importante notar que al adicionar el módulo SPP se va a afectar la entrada a la primera capa FC. Dependiendo de la cantidad de niveles de pooling que se decida adicionar, la entrada a dicha capa tendrá más o menos parámetros.

A modo de ejemplo y tomando nuestro caso de mapas de convolución de entrada de $13 \times 13 \times 256$, luego de la quinta capa de convolución:

- En un modelo de tres niveles tendremos $(1 + 4 + 16) \times 256$ lo que genera un vector de 5376 en lugar de los 9216 elementos que se poseían con anterioridad.
- Si se aplican cuatro niveles, como el que diseñamos en este trabajo, se obtienen $(1 + 4 + 9 + 36) \times 256$ valores dando lugar a un vector unidimensional de 12800 elementos.

En la arquitectura utilizada tendremos una mayor cantidad de parámetros. Enfocándonos en la zona analizada, con la capa de max pooling tradicional en la primer capa FC la cantidad de parámetros se obtienen haciendo: 9216×4096 (ver Figura 7.7, en FC1), alcanzando una cifra de 37.748.736 parámetros en esa capa. Al reemplazar esta capa y luego de agregar el módulo SPP de cuatro niveles dicha cifra se ve modificada, tomando el valor de 52.428.800 parámetros. Si bien disponemos de una cantidad mayor de parámetros, los autores [12] aclaran que las mejoras del pooling espacial piramidal no se deben simplemente a este aumento de parámetros. Los beneficios del mismo radican en el pooling

multinivel, el cual es robusto frente a varianzas en las deformaciones de los objetos y su disposición espacial.

Cambiando el eje, falta mencionar la decisión de especificar como función de pooling tomar el máximo en lugar del promedio. Lo preferimos ya que el max pooling tiende a trabajar mejor para tareas de clasificación visual. Esto se debe a que en los elementos de los mapas de características se codifican las características que indican la presencia espacial de algún patrón o concepto presente en la imagen. Al analizarlos es más informativo ver la presencia máxima de las diferentes características frente a su influencia promediada [24].

7.3.2. Hiperparámetros

Nuevamente los hiperparámetros asociados a la arquitectura de la red están definidos ya que hicimos uso de la red pre-entrenada AlexNet. Con respecto a los otros hiperparámetros relacionados al proceso de entrenamiento y a los algoritmos de aprendizaje, se mantuvo la tasa de aprendizaje en $1e-4$ y se utilizó un valor de weight decay, desarrollado en la sección 4.2, de 0.0005 por lo experimentado y sugerido en [102]. Se definió una tasa adaptativa para el aprendizaje, la cual disminuye en 0.1 cada 15 épocas.

Con respecto al tamaño de lotes, este se mantuvo en un valor de 16 pero debido al uso de la nueva técnica de transformación *Five Crop*, desarrollada en la sección 7.3.3, este hiperparámetro toma un nuevo tamaño de $16 \times 5 = 80$.

Otra diferencia con los escenarios presentadas en la sección 7.2.4 es el uso del optimizador Adam, donde se utilizaron los hiperparámetros por defecto propuestos por la librería de PyTorch:

- $beta1(0{,}9)$: la tasa de caída exponencial (*exponential decay rate*) para las estimaciones del primer momento.
- $beta2(0{,}999)$: la tasa de caída exponencial para las estimaciones de segundo momento.
- $epsilon(1e-8)$: una pequeña constante utilizada para la estabilidad numérica de la optimización y para evitar cualquier división por cero en la implementación.

7.3.3. Entrenamiento

Para el entrenamiento se mantuvieron varias de las configuraciones comentadas en la sección 7.2.3. El procedimiento de entrenamiento consistió nuevamente en transferir lo aprendido por AlexNet empleando la técnica de *transfer learning* descrita en la sección 3.6, y luego se realizó fine-tuning en ambas bases de datos estudiadas. Los pesos y biases de todas las capas de la nueva arquitectura se mantuvieron descongelados, permitiendo la adaptación de los mismos a las nuevas señales de entrada.

Se mantuvo el uso de la librería de Deep Learning PyTorch, puntualizada en la sección 5.8.2, para realizar el entrenamiento, y debido al buen rendimiento en los escenarios anteriores se procedió a correr nuevamente sobre la plataforma web Colaboratory, la cual hace uso de aceleración por hardware de GPU, descrita en el Capítulo 5.

También se decidió seguir usando la función de costo con el criterio de la entropía cruzada categórica o cross entropy, medida de precisión para variables categóricas.

Al volver a trabajar con los mismos conjuntos de imágenes pequeños, la problemática que surge a partir de esto sigue en pie. Es por esto que se volvió a adoptar la técnica de

Data Augmentation, desarrollada en la sección 3.4.5, permitiéndonos aumentar la cantidad de muestras entregadas a la entrada del modelo en el proceso de entrenamiento.

Otra problemática surgida del entrenamiento es que los conjuntos de imágenes trabajados son algo pequeños, y es por esto que se adoptó una táctica para sortear las dificultades que esta situación podría conllevar. En este sentido fue determinante para alcanzar los resultados obtenidos la utilización de la conocida técnica de Data Augmentation. Esto nos permitió aumentar la cantidad de muestras que utilizamos para generar los sets de datos, logrando un entrenamiento más enriquecedor.

Dentro del proceso de entrenamiento entonces, el ítem fundamental fue agrandar el conjunto de datos con la función de transformación *five crop*. Esta consiste en realizar cinco recortes de las imágenes de entrada (centro, esquina superior derecha, esquina inferior derecha, esquina superior izquierda, esquina inferior izquierda). Al realizar esta operación logramos alcanzar resultados superadores a trabajos de referencia, los cuales se pueden observar en la Tabla 7.5.

7.3.4. Resultados por escenario

Se definen dos escenarios de estudios: el escenario A corresponde al entrenamiento con los datos del conjunto de la UC Merced Land Use, y el escenario B plantea el uso del conjunto de datos WHU-RS. También se utiliza el mismo criterio para medir la precisión de clasificación en los resultados.

7.3.4.1. Escenario A

Para este escenario no se utiliza la técnica de validación cruzada k-fold sino que se seleccionó la técnica de MCCV detallada en la sección 4.4.2. De esta manera, en cada iteración se formaron tres subconjuntos disjuntos y aleatorios sin reposición, respetando el espacio completo de imágenes disponibles e igual número de muestras por clase.

Los poolings utilizados fueron los cuatro: $\{(1 \times 1),\ (2 \times 2),\ (3 \times 3),\ (6 \times 6)\}$ mencionados en la sección 7.3.1. Otra diferencia con los escenarios presentadas en la sección 7.2.4 fue el uso del optimizador Adam con la configuración de hiperparámetros descrita en la sección 7.3.2.

El entrenamiento se realizó por 30 épocas, utilizando cómo condición de parada las siguientes cuestiones:

- El error de entrenamiento está por debajo de 0.001.
- Se utiliza la técnica de early stopping desarrollada para los escenarios de implementación de AlexNet en la sección 7.2.3.1, definiendo una paciencia de 5 épocas y un delta mínimo de 0.001.

Se aplica una nueva transformación conocida cómo five crop al conjunto de entrenamiento y validación, la cual permite aumentar el número de muestras cómo se puede advertir en la Figura . Se utilizó un tamaño de corte de 227×227 píxeles. También se incorporó la rotación en ángulos aleatorios en el rango $-45°$ a $45°$ sumado a una rotación horizontal completa con probabilidad de ocurrencia de 0.5. Las capas de todo el modelo se mantuvieron descongeladas en todo el proceso de entrenamiento.

Los conjuntos conformados de manera aleatoria con la técnica de MCCV representan un 75 % del conjunto total para entrenamiento, 5 % para validación y 20 % para conformar

Fig. 7.11: Imagen de la categoría 'airplane' de la UC Merced y sus 5 recortes obtenidos luego de aplicar la transformación.

el conjunto de prueba o test. Luego de aplicar las transformaciones descriptas, los conjuntos quedan conformados por 7875 muestras para entrenamiento, 525 para validación y 420 para el proceso de prueba. Para medir la estabilidad propuesta del modelo pre-entrenado AlexNet-SPP, los experimentos fueron ejecutados 50 veces para obtener resultados convincentes. Se adoptaron los valores de valor medio y la desviación estándar cómo indicadores de los experimentos realizados. Los resultados obtenidos para la escenario A fueron de $\mathbf{(96.0 \pm 1.1)\,\%}$.

A partir de la matriz de confusión total reflejada en la Figura 7.12 se puede observar que únicamente 2 de las 21 clases tienen menos de un 90 % de exactitud en la clasificación. Esto implica que el 90 % de las clases han sido etiquetadas casi perfectamente, alcanzando en algunos casos valores del 100 % de exactitud.

Entre las que menores dificultades han presentado se encuentran las categorías *beach*, *chaparral*, *airplane*, *forest*, *harbor* y *parking lot*. En las dos primeras no hubo ningún error y esto puede deberse a que ambas tienen elementos espectrales, como colores, que son reiterativos en todas las imágenes de la clase. En el caso en el que se debía distinguir la presencia de aviones en la fotografía, el modelo respondió favorablemente, ya que esta categoría es bastante distante de las otras, presentando un objeto bien definido en todos los casos (las aeronaves).

Distinto es el caso de los casos de, por ejemplo, *dense residential*, la cual ha presentado las mayores dificultades. En ella se nota que algunos elementos poseen un tono más sepia y en otros resaltan los colores verdes. Tanto en *beach* como en *chaparral* el margen de variaciones es mucho menor y no hay tanta distancia entre las diferentes imágenes, lo cual facilita la tarea del clasificador. Lo mismo pasa con *forest*, donde predomina el color verde en los 100 casos.

También como se observa en la Figura 7.13 las clases más sensibles de obtener etiquetas erradas son las llamadas: *dense residential* y *medium residential*. De la matriz de confusión (*Figura 7.12*) sabemos que la equivocación en estas clases se da entre ellas mismas: es decir la mayoría de los errores en *dense residential* son etiquetados como *medium residential* y viceversa. Las clases problemáticas en nuestro escenario son las mismas que causan problemas en los estudios similares [11][102]. En algunos casos, hasta a un ser humano le costaría distinguir correctamente las imágenes de estas clases [11][110].

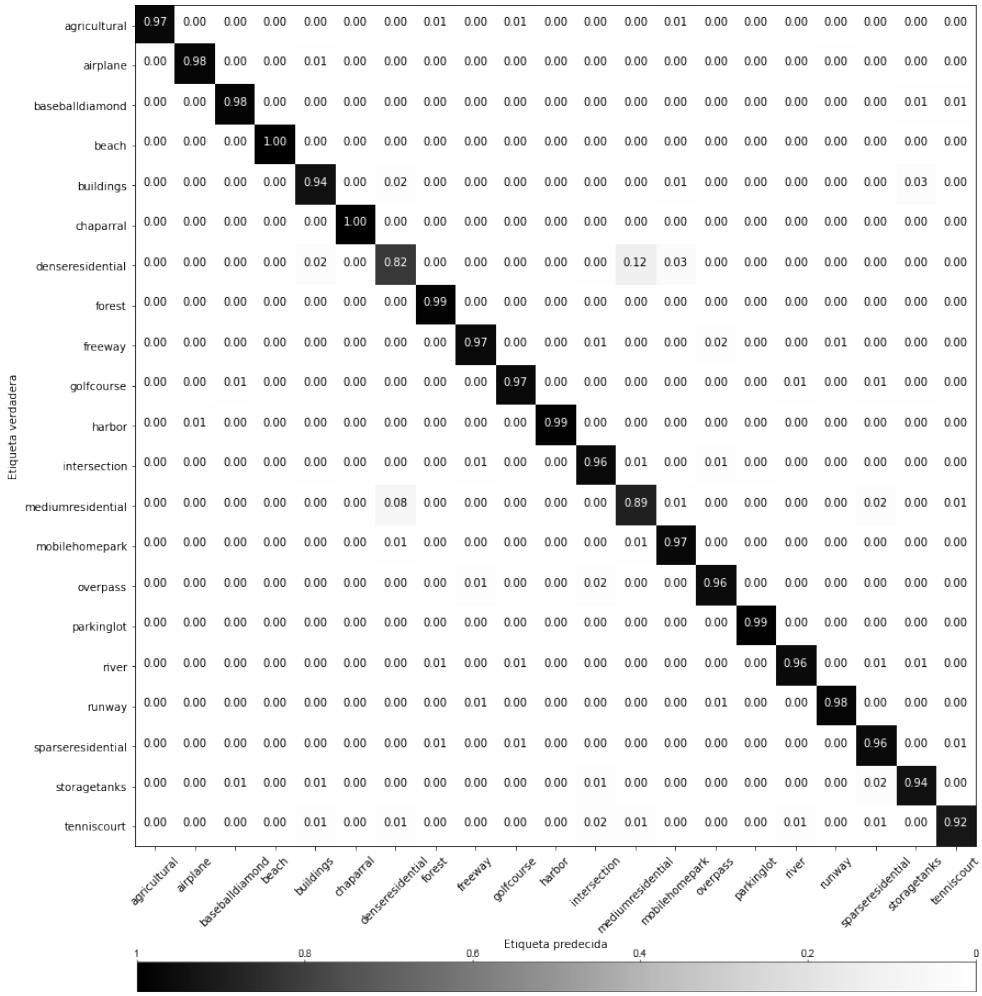


Fig. 7.12: Matriz de confusión total para el escenario A.

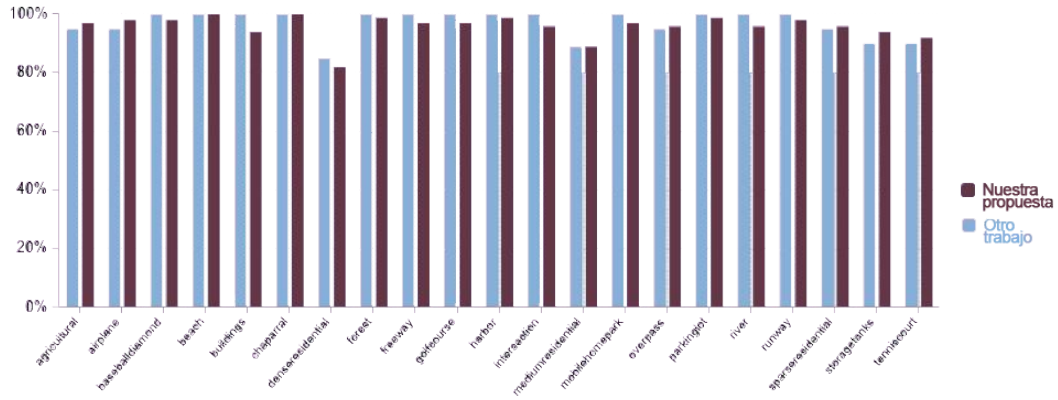


Fig. 7.13: Resultados de la clasificación del modelo propuesto en el escenario A vs el escenario del trabajo [102] en donde utilizan la misma arquitectura pero realizando 10 iteraciones de entrenamientos.

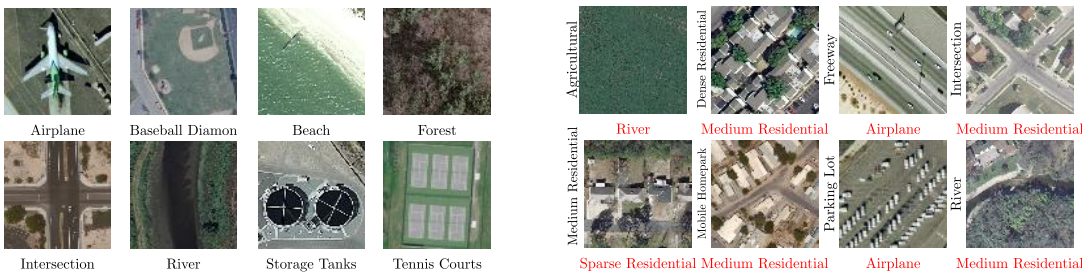


Fig. 7.14: (a) Algunas de las imágenes clasificadas de manera correcta para una de las 50 pruebas realizadas en el escenario A. (b) Varias imágenes clasificadas de manera incorrecta para una prueba de las 50 realizadas en total. En negro las etiquetas verdaderas y en rojo las predicciones erróneas realizadas por el modelo.

Luego la Figura 7.14 refleja el rendimiento del modelo entrenado para el escenario A. Por un lado la Figura 7.14-(a) muestra algunas de las imágenes que fueron clasificadas de manera correcta por el modelo entrenado, es decir, que las etiquetas precedidas por la red coinciden con la clase verdadera, mientras que la Figura 7.14-(b) expone las dificultades del modelo a la hora de clasificar algunas de las clases que componen esta base de datos.

La Figura 7.15 muestra para una iteración en particular, el comportamiento de la curva de error de entrenamiento (decreciente) y el error de validación (fluctuante) a medida que avanzan las épocas en el proceso de entrenamiento, y la época de parada. Allí se puede observar cómo actúa el algoritmo de early stopping el cual se activa una vez que el error de entrenamiento es lo suficientemente pequeño y el error de validación deja de mejorar.

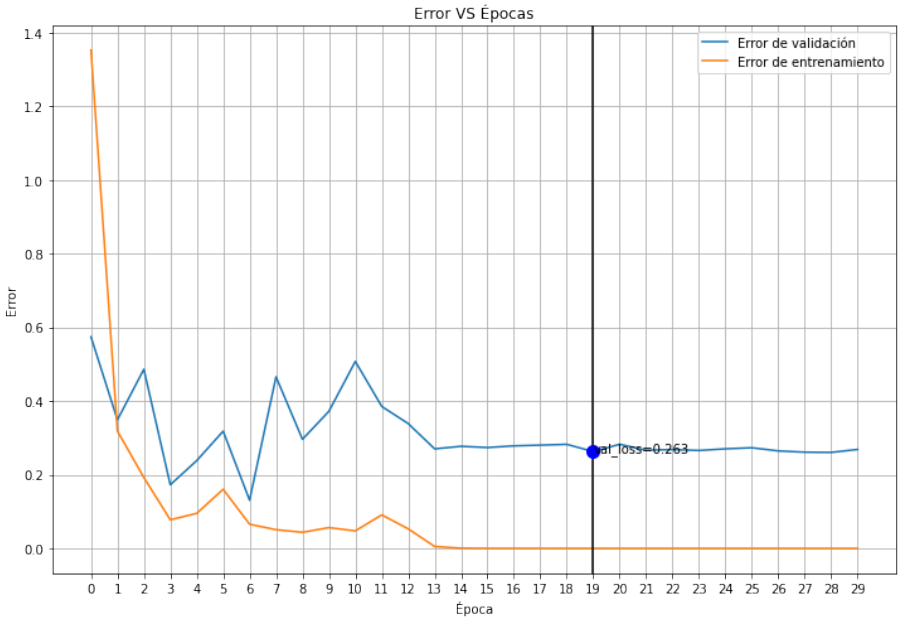


Fig. 7.15: Error de entrenamiento y error de validación VS. épocas transcurridas en el proceso de entrenamiento para el escenario A.

7.3.4.2. Escenario B

Debido a los buenos resultados obtenidos para *escenario A* con el uso del módulo SPP y el optimizador Adam, se optó por extender el uso de los mismos este nuevo escenario. Principalmente porque el banco de imágenes WHU-RS presenta muestras muy grandes, de 600×600 píxeles. Si no se adopta una estrategia como SPP, que permitiera conservar las dimensiones originales, habría que haber recortado las imágenes de manera tal que se hubiese perdido mucha información importante.

Nuevamente se hizo uso de la técnica data augmentation, y la función five crop, con la cual extrajeron a partir de una misma imagen cinco recortes de 535×535 píxeles cómo se puede observar en la Figura 7.16. Sumado al five crop, a cada partición obtenida se les sumó una rotación horizontal aleatoria, cada imagen tenía una probabilidad de 0.5 de sufrir esta última transformación.

Fig. 7.16: Imagen de la categoría *mountain* de la WHU-RS y sus 5 recortes obtenidos luego de aplicar la transformación.

El entrenamiento se realizó por 45 épocas, utilizando cómo condición de parada los siguientes parámetros:

- El error de entrenamiento está por debajo de 0.001.
- Se utiliza la técnica de early stopping basada en el algoritmo descrito en la sección 7.2.3.1, definiendo una paciencia de 7 épocas y un delta mínimo de 0.001.

Para cada entrenamiento se utilizó MCCV cómo método de fraccionamiento de los datos. Se armaron los tres conjuntos necesarios con igual número de muestras por clase, siempre considerando todo el conjunto inicial de muestras. Estos representan un 44 % del conjunto total para entrenamiento, 6 % para validación y 50 % para test. Luego de aplicar las transformaciones mencionadas los tamaños son: 2090 muestras para el conjunto de entrenamiento, 285 para validación y 475 muestras para el conjunto de prueba.

Para medir la estabilidad del modelo propuesto de AlexNet-SPP, los experimentos fueron ejecutados 50 veces para obtener resultados convincentes. Se adoptaron los valores de valor medio y la desviación estándar cómo indicadores de los experimentos realizados. Luego se obtiene un resultado de $\mathbf{(95.6 \pm 1.0)\,\%}$ de precisión.

Al observar la matriz de confusión de la Figura 7.17 podemos detectar que las categorías que mayores errores han presentado en el dataset WHU-RS incluyen las clases de *viaduct*,

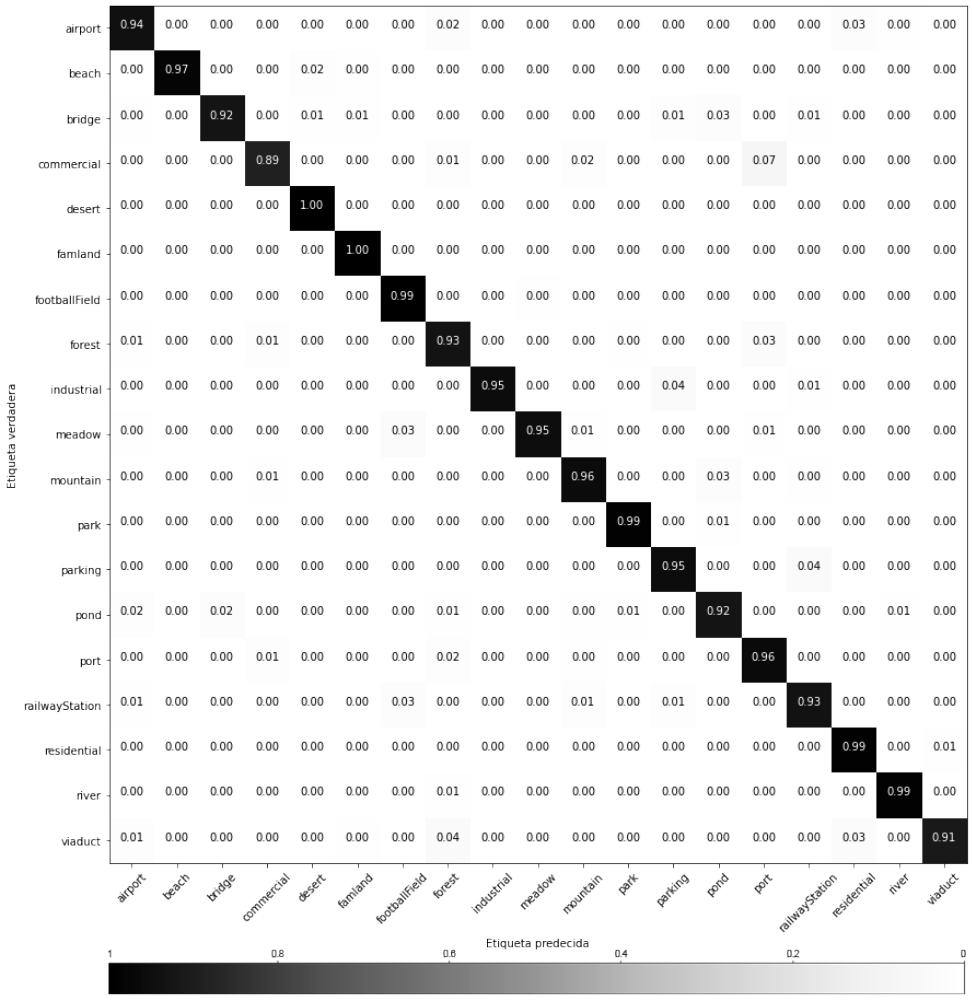
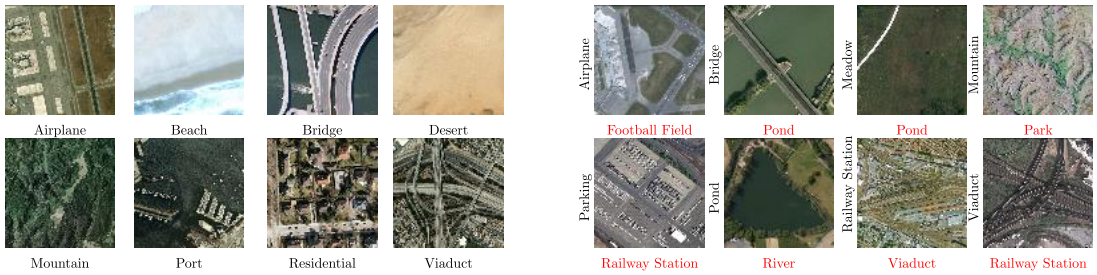


Fig. 7.17: Matriz de confusión total obtenida a partir de las 50 corridas sobre la WHU-RS.

parking, *industrial* y *bridge*. *Viaduct* fue la más sensible a errores, llevándose en su mayoría las etiquetas de *forest* y *residential* y, en menor medida, las de *airport* y *mountain*.

Las malas clasificaciones de las imágenes de *viaduct* pueden deberse a que dentro de esa clase hay numerosas imágenes que contienen colores y patrones repetidos en las otras categorías lo que presta a confusión. En *residential*, por ejemplo, las casas están rodeadas de calles las cuales en algunos casos se observa mucho verde, y creemos que ahí radica la confusión. Por otro lado, en la clase *forest* la mayoría de los ejemplos son la perspectiva superior de un territorio totalmente cubierto de árboles. Hay algunos casos en donde se presentan porciones con tierra o caminos, pero son los menos. Esto también hace que el modelo no aprenda las particularidades de la clase, considerando que la cantidad de ejemplos totales en la WHU-RS escasean, habiendo sólo 50 fotografías por etiqueta.

Otra clase que también exhibió confusiones fue *parking lot*. Casualmente, la combinación de colores en esta última es considerablemente similar a la que presenta *viaduct*. Las clasificaciones etiquetadas erróneamente, en este caso, presentan la categoría de *railway station*. Cruzando las imágenes esto puede deberse a que, teniendo en cuenta que todas



(a) *(b)*

Fig. 7.18: (a) Algunas de las imágenes clasificadas de manera correcta para una de las 50 pruebas realizadas en el escenario B. (b) Varias imágenes clasificadas de manera incorrecta para una prueba de las 50 realizadas en total. En negro las etiquetas verdaderas y en rojo las predicciones erróneas realizadas por el modelo.

son vistas superiores de tanto un estacionamiento de autos al aire libre y una estación de ferrocarril, la perspectiva superior en ambos casos ve los techos de tanto automóviles como trenes. Muchos vehículos estacionados forman una hilera que puede confundir a la red y hacer que la etiquete como un ferrocarril. En las dos situaciones los alrededores son mayoritariamente césped, lo que no ayuda en la discriminación entre las mencionadas clases.

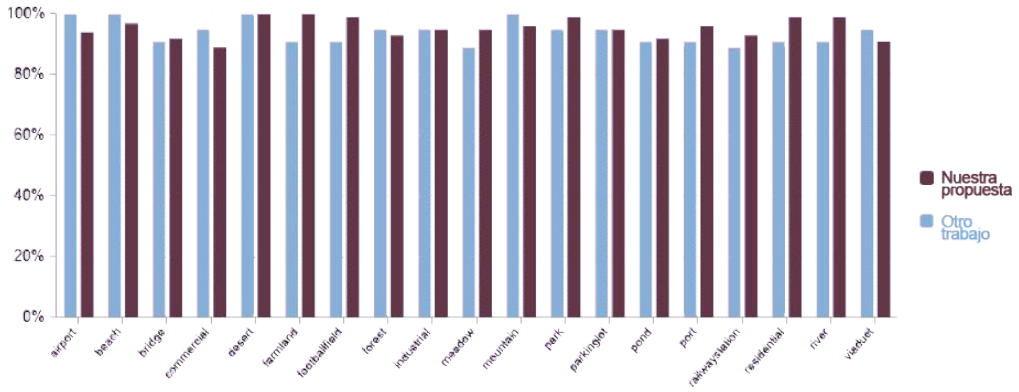


Fig. 7.19: Resultados de la clasificación del escenario propuesto B vs el escenario del trabajo [102] en donde utilizan la misma arquitectura pero realizando 10 iteraciones de entrenamientos.

Se alcanzó el 100 % de exactitud en las clases de *desert* y *famland*. La que presentó menor dificultad para el clasificador fue *desert*, esto es porque en dicha categoría todas las imágenes son bastante similares entre sí y analizándolas a simple vista se puede notar que en todas se destacan porciones enormes correspondientes a la arena de zonas desérticas.

Por último se adjunta la Figura 7.15 la cual expone para una iteración en particular, el comportamiento de la curva de error de entrenamiento (decreciente) y el error de validación (fluctuante) a medida que avanzan las épocas en el proceso de entrenamiento, y la época de parada. Allí se puede observar cómo llega un punto, aproximadamente a partir de la época 14, donde el error de entrenamiento es muy diminuto, y el error de validación comienza a estancarse. Es ahí donde entra en juego la técnica de early stopping, la cual detiene al proceso de entrenamiento en la época adecuada.

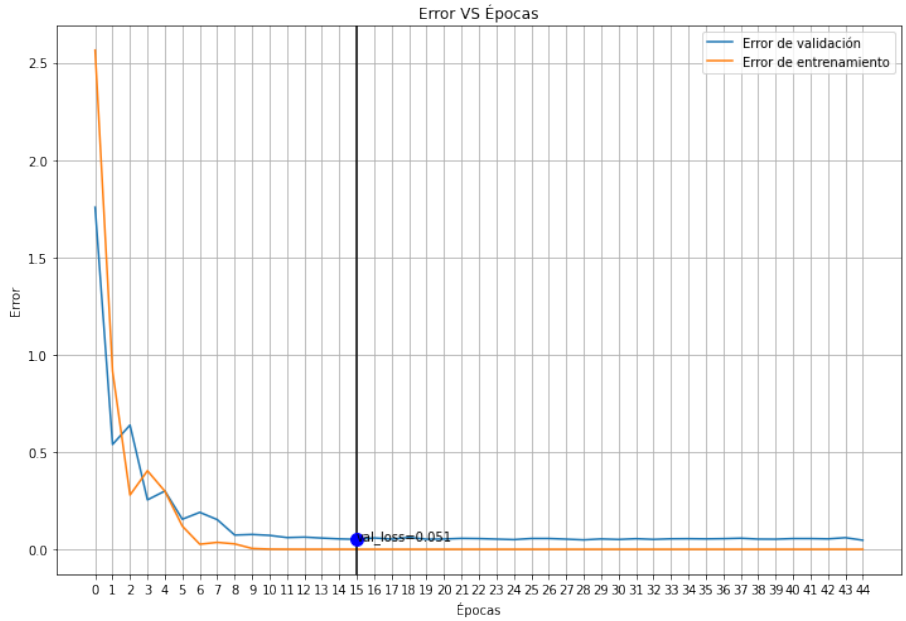


Fig. 7.20: Error de entrenamiento y error de validación VS. épocas transcurridas en el proceso de entrenamiento para el escenario B.

7.4. Comparación con otros trabajos

A continuación se mencionan algunos de los trabajos considerados para las experimentaciones mencionadas donde se pueden observar implementaciones similares:

1. Pre-Trained AlexNet Architecture with Pyramid Pooling and Supervision for High Spatial Resolution Remote Sensing Image Scene Classification [102].

2. Transferring Deep Convolutional Neural Networks for the Scene Classification of High-Resolution Remote Sensing Imagery [8].

3. Deep Learning for Satellite Image Classification [10].

4. Land Use Classification in Remote Sensing Images by Convolutional Neural Networks [11].

5. Land Use Change Detection with Convolutional Neural Network Methods [111].

En conclusión, se condujeron distintas pruebas de escenarios posibles para determinar el rendimiento y la habilidad de generalización del modelo de red convolucional AlexNet. Para ello se empleo la técnica de **transfer learning**, utilizando el modelo pre-entrenado disponible a través de la librería de Deep Learning PyTorch.

Observando la Tabla 7.5 de resultados finales, nos sentimos muy conformes con los valores obtenidos para ambos escenarios experimentales planteados. La fila que representa los modelos que utilizan las estrategias implementadas en el presente trabajo es la número tres.

Para el primer escenario trabajando con la UC Merced Land Use logramos superar los valores propuestos en la bibliografía y trabajos mencionados. Con la WHU-RS, si bien

#	Características	UC Merced	WHU-RS
1	Se respetó la arquitectura de AlexNet, pero no se utilizó la técnica de Transfer Learning. Se entrenó la red para la tarea de clasificación.	90.21 ± 1.17 %	86.32 ± 1.86 %
1	Implementación de la red AlexNet pre-entrenada.	95.00 ± 0.72 %	94.32 ± 1.54 %
1	A la arquitectura AlexNet pre-entrenada se le adiciona el módulo de pooling piramidal (SPP).	95.95 ± 1.01 %	94.73 ± 1.09 %
1	Se utilizó la arquitectura AlexNet pre-entrenada con la estrategia de Side Supervision.	95.71 ± 1.21 %	94.28 ± 2.10 %
1	A la arquitectura AlexNet pre-entrenada junto a la estrategia de Side Supervision se le incorpora el módulo SPP.	96.67 ± 0.94 %	95.00 ± 1.12
2	Se puso en práctica la técnica de feature extraction junto a la red AlexNet pre-entrenada. Técnica de Ten Crop.	95.20 %	95.49 %
3	Se utilizó la técnica de feature extraction, con un clasificador SVM multiclase, junto a la red AlexNet pre-entrenada.	∼94 %	-
4	La técnica llevada a cabo fue fine tuning, donde, con una red pre-entrenada, se adaptan algunas de las capas de alto nivel.	95.48 %	-
5	Emplearon una mezcla de la UC Merced junto a imágenes de 700 x 700 píxeles manualmente extraídas de la ciudad de Surrey, BC, Canada.	95.80 %	
	Implementación propia	**96.00 % ± 1.1 %**	**95.58 % ± 1.03 %**

Tab. 7.5: Comparación y resultados finales.

hay mayores limitaciones a la hora de comparar resultados, ya que es un set que ha sido publicado recientemente, obtuvimos resultados también superadores, alcanzando valores dentro del margen de variación de los esperados.

Es necesario mencionar que en el caso planteado en el artículo [102] sólo llevaron a cabo diez corridas en donde fueron seleccionadas estocásticamente las muestras de los subsets de train y test. En nuestro caso, el modelo fue entrenado 50 veces, 5 veces más que el estudio comparado. Esto nos dejaba en una posición mucho más vulnerable en donde teníamos muchas más oportunidades para introducir numerosos errores de clasificación. Es satisfactorio saber que el modelo se comportó de manera estable.

Capítulo 8

Conclusiones y Trabajos Futuros

En este trabajo final de carrera se ha llevado a cabo un estudio sobre la problemática y el estado del arte de la clasificación de escenarios urbanos resultantes del sensado remoto a partir de imágenes HSR y su abordaje con técnicas de aprendizaje profundo, con la intención de reproducir resultados de la literatura.

Luego propusimos implementaciones de modelos basados en la red convolucional AlexNet pre-entrenada, para el problema de clasificación de escenarios de imágenes de alta resolución espacial. Los experimentos fueron llevados a cabo haciendo uso de las bases de datos estandarizadas UC Merced Land Use y WHU-RS, las cuales son ampliamente aceptadas como punto de referencia para las tareas de clasificación. Estos conjuntos de datos presentan varios desafíos entre los que se encuentran: tienen pocas muestras para el proceso de entrenamiento, las clases comparten objetos similares ademas de texturas y patrones, y en el caso del conjunto WHU-RS la variación de iluminación, escala, resolución, la apariencia dependiente del punto de vista, y el tamaño más grande de las imágenes.

Al desarrollar estas implementaciones nos enfrentamos a la cantidad distinta de puntos de vista en todos los temas, en muchas ocasiones contradictorios sobre ciertos aspectos. Por esta razón tratamos de seguir los lineamientos expuestos en los principales trabajos de referencia [8][102][10], sabiendo que habían logrado buenos resultados.

A lo largo de lo experimentado, el elemento fundamental para alcanzar los resultados obtenidos y mantener a los modelos estables durante los entrenamientos realizados, fue el uso del módulo de Pooling Piramidal Espacial. Esto permitió abarcar toda la información de las señales de entrada, sin vernos forzados a dejar datos afuera para así obtener modelos que demuestran la exactitud y precisión buscada en la clasificación de las imágenes utilizadas. Adicionalmente, esta nueva capa incorporada posibilita el entrenamiento de modelos pre-entrenados con imágenes de diferentes tamaños a diferencia del tamaño fijo de entrada para la arquitectura tradicional de AlexNet.

El uso de la técnica de Transfer Learning junto con otras estrategias como data augmentation y una buena metodología para estimar el rendimiento del modelo de aprendizaje, nos permitió lograr resultados satisfactorios que mejoran algunos de los enfoques de la literatura basados en AlexNet. Para la UC Merced, el modelo propuesto alcanzó un *accuracy* de $96.00 \pm 1.10\,\%$ y para el caso de la WHU-RS una tasa de reconocimiento de $95.58 \pm 1.03\,\%$ fue obtenida.

El mundo del Deep Learning es un universo de posibilidades, considerando los infinitos caminos que se pueden seguir para entrenar a un modelo. En la literatura hay un sinnúmero de posibilidades y variaciones a las técnicas ya conocidas y ampliamente utilizadas.

A futuro sería un buen punto de partida para mejorar los valores obtenidos tratar de modificar la rama de Transfer Learning adoptada, para comparar resultados. Otra posibilidad seria estudiar que otras ventajas nos puede presentar el uso del módulo de Pooling Piramidal Espacial (SPP). El uso de otros modelos de CNN también podría ser considerado.

Bibliografía

[1] G. Cheng, J. Han, and X. Lu, "Remote sensing image scene classification: Benchmark and state of the art," *Proceedings of the IEEE*, vol. 105, no. 10, pp. 1865–1883, 2017.

[2] G. Cheng, J. Han, L. Guo, Z. Liu, S. Bu, and J. Ren, "Effective and efficient midlevel visual elements-oriented land-use classification using vhr remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 8, pp. 4238–4249, 2015.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[5] R. Pires de Lima and Marfurt, "Convolutional neural network for remote-sensing scene classification: Transfer learning analysis," *Remote Sensing*, vol. 12, p. 86, 12 2019.

[6] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1717–1724, 2014.

[7] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[8] F. Hu, G.-S. Xia, J. Hu, and L. Zhang, "Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery," *Remote Sensing*, vol. 7, pp. 14680–14707, 11 2015.

[9] Y. Yang and S. Newsam, "Bag-of-visual-words and spatial extensions for land-use classification," in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, (New York, NY, USA), p. 270–279, Association for Computing Machinery, 2010.

[10] M. A. Shafaey, M. A.-M. Salem, H. M. Ebied, M. N. Al-Berry, and M. F. Tolba, *Deep Learning for Satellite Image Classification*, pp. 383–391. 2019.

[11] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva, "Land use classification in remote sensing images by convolutional neural networks," 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014.

[13] S. Haykin, *Neural networks and learning machines*. New York: Prentice Hall/Pearson, 2009.

[14] F. Chollet, *Deep learning with Python*. Shelter Island, NY: Manning Publications Co, 2018.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[16] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, p. 1237–1242, AAAI Press, 2011.

[17] D. Hubel and T. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, 1962.

[18] F. Huang and Y. LeCun, "Large-scale learning with svm and convolutional nets for generic object categorization," in *Proceedings - 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2006*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 284–291, Dec. 2006. 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2006 ; Conference date: 17-06-2006 Through 22-06-2006.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," 2014.

[20] F.-F. Li, J. Johnson, and S. Yeung, "Lecture 5 — convolutional neural networks." `https://www.youtube.com/watch?v=bNb2fEVKeEo`, 2017. Accedido 27-07-2020.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[22] L. Fei-Fei, J. Johnson, and S. Yeung, "Lecture 8 deep learning software." `https://www.youtube.com/watch?v=6SlgtELqOWc`, 2017.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

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

[25] M. Claesen and B. D. Moor, "Hyperparameter search in machine learning," 2015.

[26] W.-Y. Lee, S.-M. Park, and K.-B. Sim, "Optimal hyperparameter tuning of convolutional neural networks based on the parameter-setting-free harmony search algorithm," *Optik*, vol. 172, pp. 359 – 367, 2018.

[27] N. Mohd Aszemi and D. D. D. Panneer Selvam, "Hyperparameter optimization in convolutional neural network using genetic algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 10, pp. 269 – 278, 06 2019.

[28] "Entrenamiento (perceptrón multicapa)." `https://www.ibm.com/support/knowledgecenter/es/SSLVMB_sub/statistics_mainhelp_ddita/spss/neural_network/idh_idd_mlp_training.html`. Accedido 23-04-2020.

[29] "Ajuste de los hiperparámetros de un modelo mediante azure machine learning." `https://docs.microsoft.com/es-es/azure/machine-learning/how-to-tune-hyperparameters`. Accedido 16-04-2020.

[30] P. Probst, B. Bischl, and A.-L. Boulesteix, "Tunability: Importance of hyperparameters of machine learning algorithms," 2018.

[31] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012.

[32] M. de J. Pérez Jiménez and F. Caparrini, *Máquinas moleculares basadas en ADN*. Colección Divulgación Científica, Universidad de Sevilla, 2003.

[33] G. Beni and J. Wang, "Swarm intelligence in cellular robotic systems," in *Robots and Biological Systems: Towards a New Bionics?* (P. Dario, G. Sandini, and P. Aebischer, eds.), (Berlin, Heidelberg), pp. 703–712, Springer Berlin Heidelberg, 1993.

[34] R. Mosquera, "Diseño de un algoritmo de enjambre para el trabajo colaborativo de mini robots para recoger y clasificar piezas de diferentes formas y colores. propuesta de semillero de investigación syna," *Memorias De Congresos UTP*, pp. 218–223, 2017.

[35] J. Kennedy, *Swarm intelligence*. San Francisco: Morgan Kaufmann Publishers, 2001.

[36] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation.," in *Encyclopedia of Database Systems* (L. Liu and M. T. Özsu, eds.), pp. 532–538, Springer US, 2009.

[37] B. Ghojogh and M. Crowley, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial," 2019.

[38] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statist. Surv.*, vol. 4, pp. 40–79, 2010.

[39] S. Raschka, "Model evaluation, model selection, and algorithm selection in machine learning," *CoRR*, vol. abs/1811.12808, 2018.

[40] M. Kuhn and K. Johnson, *Applied predictive modeling*. New York: Springer, 2013.

[41] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning : with applications in R*. New York, NY: Springer, 2013.

[42] B. Efron, "How biased is the apparent error rate of a prediction rule?," *Journal of the American Statistical Association*, vol. 81, no. 394, pp. 461–470, 1986.

[43] P. Zhang, "Model selection via multifold cross validation," *Ann. Statist.*, vol. 21, pp. 299–313, 03 1993.

[44] "K-fold and montecarlo cross-validation vs bootstrap: a primer." `https://nirpyresearch.com/kfold-montecarlo-cross-validation-bootstrap-primer/`. Accedido 26-08-2020.

[45] R. Fonseca-Delgado and P. Gómez-Gil, "An assessment of ten-fold and monte carlo cross validations for time series forecasting," in *2013 10th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pp. 215–220, 2013.

[46] R. Kohavi and F. Provost, "Glossary of terms," *Machine Learning*, vol. 2, pp. 271–274, 01 1998.

[47] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *in COMPSTAT*, 2010.

[48] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011.

[49] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[50] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[51] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4148–4158, Curran Associates, Inc., 2017.

[52] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[53] X. Jiang, B. Hu, S. Satapathy, S. Wang, and Y.-D. Zhang, "Fingerspelling identification for chinese sign language via alexnet-based transfer learning and adam optimizer," *Scientific Programming*, vol. 2020, pp. 1–13, 05 2020.

[54] C. Danelon de Almeida, L. Goliatt, T. Thales, L. Moisés, C. Gulliver, and C. Lecino, "Automatic steel microstructural quantification by convolutional neural networks," *Mecánica Computacional Vol XXXVII*, pp. 2023–2032, 11 2019.

[55] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011.

[56] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[57] S. Schuchmann, *Analyzing the Prospect of an Approaching AI Winter*. PhD thesis, 05 2019.

[58] K. Kadupitige, "Intersection of hpc and machine learning," 2017.

[59] V. Sze, Y. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–8, 2017.

[60] NVIDIA, "On-premises deep learning solutions." `https://www.nvidia.com/en-us/deep-learning-ai/solutions/on-premises/`. Accedido 16-11-2020.

[61] G. D. Maayan, "Supercomputers and machine learning: A perfect match." `https://insidebigdata.com/2019/11/27/supercomputers-and-machine-learning-a-perfect-match/`. Accedido 27-11-2020.

[62] H. El-Rewini and M. Abd-El-Barr, *Advanced Computer Architecture and Parallel Processing (Wiley Series on Parallel and Distributed Computing)*. USA: Wiley-Interscience, 2005.

[63] W. Stallings, *Computer organization and architecture : designing for performance*. Boston: Pearson-Prentice Hall, 2016.

[64] A. Ali and K. S. Syed, "Chapter 3 - an outlook of high performance computing infrastructures for scientific computing," in *Advances in Computers* (A. Memon, ed.), vol. 91 of *Advances in Computers*, pp. 87 – 118, Elsevier, 2013.

[65] S. W. Kim and R. Eigenmann, "The structure of a compiler for explicit and implicit parallelism," in *Proceedings of the 14th International Conference on Languages and Compilers for Parallel Computing*, LCPC'01, (Berlin, Heidelberg), p. 336–351, Springer-Verlag, 2001.

[66] "Symmetric multi-processor." `https://www.sciencedirect.com/topics/computer-science/symmetric-multi-processor`. Accedido 16-11-2020.

[67] A. Tanenbaum, *Sistemas distribuidos : principios y paradigmas*. México: Pearson Educación, 2008.

[68] "Intel® core™ i9-7960x x-series processor product specifications." `https://ark.intel.com/content/www/us/en/ark/products/126697/intel-core-i9-7960x-x-series-processor-22m-cache-up-to-4-20-ghz.html`. Accedido 15-11-2020.

[69] D. Mackay, "Hybrid parallelism: Parallel distributed memory and shared memory." `https://software.intel.com/content/www/us/en/develop/articles/hybrid-parallelism-parallel-distributed-memory-and-shared-memory-computing.html`. Accedido 15-11-2020.

[70] "Debu.gs: Inferno part 2: Let's make a cluster!." `http://debu.gs/entries/inferno-part-2-let-s-make-a-cluster`. Accedido 03-11-2020.

[71] A. Hadoop, "The apache software foundation." `https://hadoop.apache.org/releases.html`. Accedido 16-11-2020.

[72] S. M. Hashemi and A. K. Bardsiri, "Cloud computing vs. grid computing," 2012.

[73] I. T. L. C. S. Division, "Cloud computing: Csrc." `http://csrc.nist.gov/groups/SNS/cloud-computing/`. Accedido 15-11-2020.

[74] A. Gray, "Gpu architecture." `https://www.archer.ac.uk/training/course-material/2017/11/gpu-daresbury/slides/GPU_Architecture.pdf`. Accedido 15-11-2020.

[75] C. Yang, C. Huang, and C. Lin, "Hybrid cuda, openmp, and mpi parallel programming on multicore gpu clusters," *Comput. Phys. Commun.*, vol. 182, pp. 266–269, 2011.

[76] E. Montes de Oca, L. C. De Giusti, F. Chichizola, A. E. De Giusti, and M. Naiouf, "Utilización de cluster de gpu en hpc," in *XX Congreso Argentino de Ciencias de la Computación (Buenos Aires, 2014)*, 2014.

[77] N. Hagoort, "Exploring the gpu architecture." `https://nielshagoort.com/2019/03/12/exploring-the-gpu-architecture/`. Accedido 16-11-2020.

[78] A. Priyadarshana, "Cuda - gpu memory architecture." `https://medium.com/@ashanpriyadarshana/cuda-gpu-memory-architecture-8c3ac644bd64`. Accedido 16-11-2020.

[79] C. McClanahan, "History and evolution of gpu architecture a paper survey," 2011.

[80] K. Nogueira, O. Penatti, and J. dos Santos, "Towards better exploiting convolutional neural networks for remote sensing scene classification," *Pattern Recognition*, vol. 61, 02 2016.

[81] B. Barney, "Introduction to parallel computing." `https://computing.llnl.gov/tutorials/parallel_comp/`. Accedido 12-11-2020.

[82] B. Barney, "Introduction and objectives." `https://www.top500.org/project/introduction/`. Accedido 24-10-2020.

[83] T. . T. List, "November 2020." `https://www.top500.org/lists/top500/2020/11/`. Accedido 04-12-2020.

[84] A. NANJAPPA, *CAFFE2 QUICK START GUIDE : modular and scalable deep learning made easy*. S.l: PACKT PUBLISHING LIMITED, 2019.

[85] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang, "Tensorflow: A system for large-scale machine learning," *CoRR*, vol. abs/1605.08695, 2016.

[86] "Repositorio caffe github." `https://pytorch.org/`. Accedido 20-04-2020.

[87] "Documentación tensorflow core v2.1.0 tf.graph." `https://www.tensorflow.org/api_docs/python/tf/Graph`. Accedido 24-04-2020.

[88] "Tensorflow 2.0 is now available!." `https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html`. Accedido 16-04-2020.

[89] "Colaboratory: Preguntas frecuentes." `https://research.google.com/colaboratory/faq.html`. Accedido 16-04-2020.

[90] "Imágenes de máquina virtual de aprendizaje profundo." `https://cloud.google.com/deep-learning-vm`. Accedido 16-04-2020.

[91] "Amazon sagemaker." `https://aws.amazon.com/es/sagemaker`. Accedido 16-04-2020.

[92] "Azure machine learning." `https://azure.microsoft.com/es-es/services/machine-learning/`. Accedido 16-04-2020.

[93] "Tutorial: Fundamentals of remote sensing," February 29, 2016 2016.

[94] "How to Acquire Large Satellite Image Datasets for Machine Learning Projects." `https://appsilon.com/how-to-acquire-large-satellite-image-datasets-for-machine-learning-projects/`, Feb. 2020. Accedido 27-10-2020.

[95] R. C. Gonzalez and R. E. Woods, *Digital image processing*. New York, NY: Pearson, 2018.

[96] E. Salinero, *Fundamentos de teledeteccion espacial*. Madrid: Ediciones Rialp, 1996.

[97] "Landsat 8 overview." `https://landsat.gsfc.nasa.gov/landsat-8/landsat-8-overview`. Accedido 17-11-2020.

[98] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning : data mining, inference, and prediction*. New York: Springer, 2009.

[99] Y. Xu and R. Goodacre, "On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning," *Journal of Analysis and Testing*, vol. 2, 10 2018.

[100] "National geospatial program." `https://www.usgs.gov/core-science-systems/national-geospatial-program/imagery`. Accedido 26-08-2020.

[101] B. van Driel, "Roof type classification in aerial imagery using convolutional neural networks: exploring the exploitation of convolutional features using feature coding and random forests," Master's thesis, Eindhoven University of Technology, april 2019.

[102] X. Han, Y. Zhong, C. Liqin, and L. Zhang, "Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification," *Remote Sensing*, vol. 9, p. 848, 08 2017.

[103] "Cs231n convolutional neural networks for visual recognition course website." `https://cs231n.github.io/convolutional-networks/`. Accedido 26-08-2020.

[104] S. L. Smith, P. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *CoRR*, vol. abs/1711.00489, 2017.

[105] L. Prechelt, "Early stopping-but when?," in *Neural Networks: Tricks of the Trade* (G. B. Orr and K.-R. Müller, eds.), vol. 1524 of *Lecture Notes in Computer Science*, pp. 55–69, Springer, 1996.

[106] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Early stopping is nonparametric variational inference," 2015.

[107] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig, "Early stopping without a validation set," *CoRR*, vol. abs/1703.09580, 2017.

[108] K. Men, P. Boimel, J. Janopaul-Naylor, H. Zhong, M. Huang, H. Geng, C. Cheng, Y. Fan, J. Plastaras, E. Ben-Josef, and Y. Xiao, "Cascaded atrous convolution and spatial pyramid pooling for more accurate tumor target segmentation for rectal cancer radiotherapy," *Physics in Medicine and Biology*, vol. 63, 08 2018.

[109] J. Chen, C. Wang, and Y. Tong, "Aticnet: semantic segmentation with atrous spatial pyramid pooling in image cascade network," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, 12 2019.

[110] S. Jiang, H. Zhao, W. Wu, and Q. Tan, "A novel framework for remote sensing image scene classification," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-3, pp. 657–663, 2018.

[111] C. Cao, S. Dragicevic, and S. Li, "Land-use change detection with convolutional neural network methods," *Environments*, vol. 6, p. 25, 02 2019.