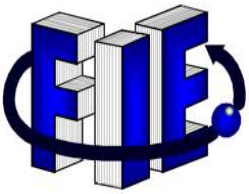


Universidad Michoacana de San Nicolás de Hidalgo

Facultad de Ingeniería Eléctrica

CLASIFICACIÓN DE OBJETOS POR MEDIO DE APRENDIZAJE PROFUNDO

TESIS

Que para obtener el grado de
INGENIERO EN COMPUTACIÓN

Presenta
Angel Torres Martín

Asesor de Tesis
M.I. Moisés García Villanueva

Morelia, Michoacán, Agosto 2019

Agradecimientos

Agradezco a la Facultad de Ingeniería Eléctrica de la UMSNH por permitirme formarme en ella, a mi asesor M.I. Moisés García Villanueva por tener la paciencia para guiarme en el desarrollo de este trabajo, así como a mis padres José Albino Abelardo Torres Rojas y Agustina Martín Gomez por su apoyo incondicional a lo largo de mi vida y los proyectos que he emprendido, también agradezco a mi hermano y hermana, amigos y personas que estuvieron a mi lado durante esta etapa de mi vida, por sus ánimos y recomendaciones para concluir este trabajo.

Dedicatoria

Dedico esta tesis a mis padres por darme la vida, cuidarme y permitirme llegar hasta aquí con su apoyo y consejos. A mis amigos quienes me motivaron a continuar y concluir este trabajo. A todos mis maestros quienes me dieron las pautas para mi formación profesional.

Resumen

El propósito de esta tesis es la implementación de un sistema de clasificación de objetos por medio de aprendizaje profundo y visión computacional; se presenta mediante evidencia empírica el proceso de diseño de una arquitectura de una red neuronal profunda, basada en capas de redes neuronales convolucionales. Adicionalmente se creó un conjunto de datos con 4 clases de imágenes que contiene un total de 8423 elementos. Utilizando la biblioteca Keras para la creación de la red neuronal, Flickr para la obtención de imágenes, junto con OpenCV biblioteca utilizada para el manejo de imágenes. Se muestran resultados de las pruebas del entrenamiento de un modelo de aprendizaje profundo y su aplicación, logrando una eficiencia del 95 % en los datos de validación. Finalmente el presente trabajo es una evidencia de un desarrollo en aprendizaje profundo con un conjunto de datos pequeño que puede aplicarse en sistemas más complejos.

Palabras clave: Visión computacional, Aprendizaje Profundo, Aprendizaje Automático, Identificación de objetos, Deep Learning, Redes Neuronales Convolucionales.

Abstract

The purpose of this thesis is the implementation of an object classification system through deep learning and computer vision; The process of designing an architecture of a deep neural network, based on layers of convolutional neural networks, is presented by empirical evidence. Additionally, a data set was created with 4 class of images containing a total of 8423 elements. Using the Keras library for the creation of the neural network, Flickr for obtaining images, together with OpenCV library used for image management. Results of the training tests of a deep learning model and its application are shown, achieving an efficiency of 95 % in the validation data. Finally, this work is evidence of a development in deep learning with a small data set that can be applied in more complex systems.

Keywords: Computer Vision, Deep Learning, Machine Learning, Object Identification, Deep Learning, Convolutional Neural Networks.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Símbolos

CUDA	**C**ompute **U**nified **D**evice **A**rchitecture (Arquitectura de cálculo paralelo)
CV	**C**omputer **V**ision (Visión computacional)
IA	**I**nteligencia **A**rtificial
CNN	**C**onvolutional **N**eural **N**etwork (Red neuronal convolucional)
LSTM	**L**ong **S**hort **T**erm **M**emory (Memoria a largo plazo)
RGB	**R**ed **G**reen **B**lue (Rojo,Verde, Azul)
CNTK	(The Microsoft Cognitive Toolkit)
CPU	**C**entral **P**rocessing **U**nit (Unidad de procesamiento central)
GPU	**G**raphics **P**rocessing **U**nit (Unidad de procesamiento gráfico)
API	**A**pplication **P**rogramming **I**nterface (Interfaz de Programación de Aplicaciones)

Capítulo 1

Introducción

Una forma para distinguir nuestro entorno es clasificando los objetos que se encuentran en este. Nuestro cerebro a cada instante está clasificando los objetos a nuestro alrededor y con ello tomamos decisiones, por ejemplo, si deseamos caminar, primero verificamos las condiciones existentes para poder hacerlo, si hay un pasillo lo podremos cruzar, si hay una puerta debemos primero realizar la acción de abrir dicha puerta y entonces caminar, en el caso de observar un objeto como una mesa habrá que determinar la dirección en que deberemos caminar. Una gran cantidad de aplicaciones pueden desarrollarse cuando se cuenta con mecanismos de clasificación eficientes y recientemente en el área de Inteligencia Artificial en el subcampo de Aprendizaje de Máquina han surgido nuevas técnicas y paradigmas que abordan el problema de clasificación de objetos.

El crecimiento en el desarrollo del poder computacional y procesamiento de datos ha sido de manera exponencial en la última década, esto ha ayudado a que las técnicas del área conocida como aprendizaje profundo (del inglés, Deep Learning) se popularicen en mayor medida para la solución de problemas que los métodos convencionales de Aprendizaje Automático, los cuales son usados para problemas mas específicos [Wang17].

Gracias a los grandes volúmenes de datos (textos e imágenes) que segundo a segundo se generan y pueden estar disponibles para todos los usuarios en el internet, las técnicas con redes neuronales han tomado relevancia nuevamente, dando lugar a nuevas arquitecturas que logran un mejor procesamiento de la información [Chollet18].

El aprendizaje automático, aprendizaje automatizado o aprendizaje de máquinas (del inglés, Machine Learning) es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos [Russell04].

Necesitamos aprender en los casos en que no podemos escribir directamente un programa de computadora para resolver un problema dado, pero necesitamos datos de ejemplo o experiencia [Alpaydin09].

El aprendizaje profundo es un subcampo específico del aprendizaje automático, una nueva forma de representación de aprendizaje por medio de los datos que ponen un énfasis en el aprendizaje de capas sucesivas. El aprendizaje profundo es la idea de representación por capas sucesivas [Chollet18].

1.1. Antecedentes

El antecedente más importante en relación al desarrollo de poder de cómputo, se dio gracias a los grandes avances que exigían las tarjetas gráficas en los videojuegos, el aspecto de más impacto es el relacionado al renderizado y su rendimiento, lo que ocasionó que los fabricantes de tarjetas gráficas, tales como Nvidia, mejoraran sus tarjetas de vídeo, pero esto no solo ayudó a la industria del videojuego dado que varios procesos computacionales requieren de muchas operaciones de matrices, creando librerías especializadas, una de ellas es CUDA *Compute Unified Device Architecture*, la cual permite a las tarjetas gráficas realizar cálculos computacionales científicos en las GPU's (Graphics Processing Unit). La GPU es el nuevo motor de la computación [Kirk07].

Alrededor de 2009-2010 llegaron varios cambios simples pero importantes para los algoritmos de redes neuronales, mejores funciones de activación para capas neuronales, mejores esquemas de inicialización de peso, comenzando con el entrenamiento previo de capas, y mejores esquemas de optimización [Chollet18].

1.2. Justificación

La automatización de procesos requiere de mecanismos eficientes de clasificación. Para ello es importante contar con aplicaciones que suministren conocimiento a los mecanismos de control del proceso automatizado [Huval15].

La inteligencia artificial requiere de grandes volúmenes de información con la finalidad de construir la memoria, para ello es deseable que la información contenga información de clasificación, junto con las características propias de dicha información. Al unir estos se podrá generar una aplicación de alto nivel que pueda clasificar un objeto en una escena [Chollet18].

Explorar y aplicar una nueva herramienta, como lo es el Aprendizaje Profundo, para buscar mejores soluciones a problemas en los cuales el aprendizaje de máquina utilizado hasta ahora, no ha logrado los mejores resultados en forma eficiente. Al usar el aprendizaje profundo con visión computacional, se espera tener una gran cantidad de aplicaciones con un mejor desempeño en la identificación de los objetos en una escena, ya que con estos modelos de aprendizaje es más sencillo implementar un identificador de objetos dentro de una imagen.

Existe una gran cantidad de tareas de aplicación en el mundo real que requieren conocer o identificar ciertos elementos en una imagen. Algunos ejemplos son:

1. En una tarea simple, en donde con una cámara de vigilancia se desea saber si la puerta que supervisa está abierta o cerrada.

2. En tareas más complejas, en donde se desea identificar la existencia de varios tipos de objetos.

3. En la identificación de rostros para tomar asistencia en un trabajo.

4. En tareas donde se debe identificar cuantos objetos del mismo tipo se encuentran en una escena y su ubicación en esta.

5. Saber los tipos de objetos existentes para tomar una decisión.

1.2.1. Motivaciones de la Tesis

Establecer una base para el uso del aprendizaje profundo junto con visión computacional, generando así módulos de conocimiento que puedan ser unidos para obtener un programa mas complejo en el reconocimiento de objetos a través de una entrada de vídeo o información almacenada en algún dispositivo.

1.3. Objetivo de la Tesis

1.3.1. Objetivo general

Implementar y probar un sistema de identificación de objetos mediante aprendizaje profundo , para lograr identificar mesas, sillas, escaleras y puertas dentro de una imagen, realizando pruebas para verificar la eficiencia de la identificación, la imagen a analizar podrá ser obtenida mediante una cámara web, vídeo ó imagen previamente guardados.

1.3.2. Objetivos particulares

- Generar la base de conocimiento para la identificación de objetos.
- Obtener evidencia empírica de un modelo de Aprendizaje Profundo respecto a su eficiencia en la clasificación de múltiples objetos.
- Usar los modelos generados para la identificación de objetos en un sistema de visión computacional que indicará si en una escena o imagen se encuentra uno de los objetos entrenados en el sistema.

1.4. Descripción de los capítulos

- En el capítulo uno se da una breve introducción a esta Tesis. Se mencionan antecedentes, se plantea el problema y se establecen los objetivos principales y particulares.
- El Capítulo Dos describe el marco teórico. Da una descripción de los términos a utilizar en este documento y se abordan los temas de redes neuronales, aprendizaje profundo y visión computacional.

- El Capítulo Tres describe que son y como usar las Redes Neuronales Convolucionales.
- El Capítulo Cuatro se muestra la arquitectura propuesta y el proceso de implementación realizado para entrenar una red neuronal en la clasificación de objetos.
- El Capítulo Cinco describe los experimentos realizados y los resultados obtenidos, al probar la identificación de los objetos propuestos (escaleras, mesas, puertas y sillas).
- El Capítulo Seis muestra las conclusiones, recomendaciones y propuestas para trabajos futuros.

Capítulo 2

Aprendizaje Profundo

En este capítulo se presenta un panorama general del Aprendizaje Profundo, conceptos y nomenclaturas que se estarán utilizando a lo largo del documento. Además de introducir el área de Visión Computacional que al usarse con Aprendizaje Profundo nos permitirá la identificación de objetos en una escena, ya sea una sola imagen o en un vídeo.

2.1. Inteligencia Artificial

La Inteligencia Artificial (IA) es una rama especializada de la informática que investiga y produce razonamiento por medio de máquinas autómatas y que pretende fabricar artefactos dotados de la capacidad de pensar [Munárriz94]. Una definición concreta sería la siguiente: es el esfuerzo por automatizar las tareas intelectuales normalmente realizadas por humanos [Chollet18].

Se dice que muchas actividades mentales humanas, como escribir programas de computadora, hacer matemáticas, participar en el razonamiento de sentido común, comprender el lenguaje e incluso conducir un automóvil, demandan "inteligencia". En las últimas décadas, se han construido varios sistemas informáticos que pueden realizar tareas como estas. Específicamente, hay sistemas informáticos que pueden diagnosticar enfermedades, planificar la síntesis de compuestos químicos orgánicos complejos, resolver ecuaciones diferenciales en forma simbólica, analizar circuitos electrónicos, comprender cantidades limitadas de texto en lenguaje humano y lenguaje natural, o escribir pequeños programas

de computadora para cumplir con las normas formales. Podríamos decir que tales sistemas poseen algún grado de inteligencia artificial [Nilsson14].

Para implementar un sistema de IA, este debe ser capaz de analizar datos en grandes cantidades(big data), identificar patrones y tendencias y, por lo tanto, formular predicciones de forma automática, con rapidez y precisión. Al integrar análisis predictivo en un sistema de IA, como la clasificación de objetos y otras técnicas, se logran desarrollar aplicaciones con un grado de inteligencia, lográndose gracias a la existencia de grandes cantidades de datos e información para el entrenamiento de dicho sistema.

2.2. Aprendizaje de máquina

También llamado aprendizaje automático, como su nombre lo indica es una rama de la inteligencia artificial la cual permite a las computadoras "aprender". Para el caso de la visión computacional, se trata de un proceso en el cuál se "enseña" a las computadoras la forma y las características del objeto a analizar, para posteriormente ser clasificado, dichas características pueden incluir valores como temperatura, intensidad de color, distancias, bordes, entre otros.

De acuerdo al libro del año 2008 de los autores Gary Bradsky y Adrian Kaehler [Bradski08], la meta de las técnicas de aprendizaje de máquina es convertir datos en información discriminante. Después de aprender de una colección de datos, se quiere que la máquina sea capaz de resolver preguntas tales como "¿Qué otros datos son similares a éste?", "¿Hay un carro en la imagen?". El aprendizaje de máquina convertirá los datos en información extrayendo reglas o patrones de esos datos, muchas de las ocasiones identificadas como características.

Para resolver un problema en una computadora, nosotros necesitamos un algoritmo. Un algoritmo es una secuencia de instrucciones que nos llevarán a transformar una entrada en una salida. Un ejemplo puede ser un algoritmo para ordenar, cuya entrada de datos sería un arreglo de números, y la salida sería el mismo arreglo pero ordenado. Para algunas tareas no se cuenta con un algoritmo completamente efectivo, por ejemplo, distinguir los emails spam de emails no spam, en donde los datos de entrada en este caso es

un simple archivo con caracteres, la salida debería ser un si/no indicando si el mensaje es spam o no, se busca entonces como transformar la entrada en la salida para indicar lo que consideramos spam, que en muchas ocasiones cambia para cada individuo, dependiendo del contexto. Lo que nosotros tenemos es conocimiento, que creamos desde los datos, nosotros podemos compilar toneladas de mensajes ejemplo, para saber cuales son spam y cuales no, y eso es a lo que llamamos "aprender". En otras palabras , nosotros haremos que la computadora extraiga un modelo de los datos mediante un algoritmo para intentar resolver en forma efectiva esta tarea. Esta forma de trabajar sirve para problemas en los cuales no existe un algoritmo [Alpaydin09].

2.3. Neurona Artificial

Desde 1943, cuando Warren Mcculloch y Walter Pitts presentaron el primer modelo de neuronas artificiales o perceptrón, se hicieron propuestas nuevas y más sofisticadas de una década a otra. El análisis matemático ha resuelto algunos de los misterios planteados por los nuevos modelos, pero ha dejado muchas preguntas abiertas para futuras investigaciones [Rojas13]. Específicamente, el modelo de la neurona calcula una suma ponderada de sus entradas de otras unidades y genera un "uno" o un "cero" según si se encuentra por encima o debajo del umbral [Hertz18]. En la Figura 2.1 muestra el diagrama de una neurona donde la unidad se dispara si la suma de los pesos $\sum_j = w_{ij} n_j$ de las entradas alcanza o excede el umbral u_i [McCulloch43].

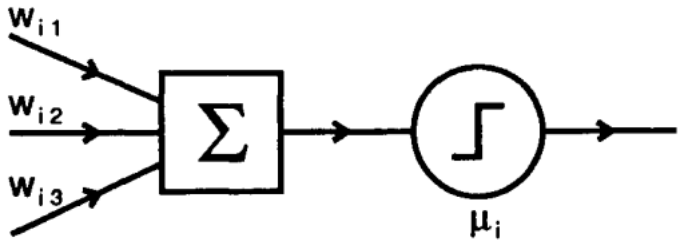


Figura 2.1: Diagrama esquemático de una neurona McCulloch-Pitts [McCulloch43].

2.4. Red neuronal

La representación de las capas en aprendizaje profundo es por medio de las redes neuronales de forma estructurada una encima de otra [Chollet18].

Una red neuronal es un conjunto interconectado de elementos (véase la Figura 2.2), unidades o nodos de procesamiento simples, cuya funcionalidad está basada en las neuronas animales. La capacidad de procesamiento de la red se almacena en las fortalezas de conexión entre unidades, o pesos, obtenidos por un proceso de adaptación o aprendizaje de un conjunto de patrones de entrenamiento [Gurney14].

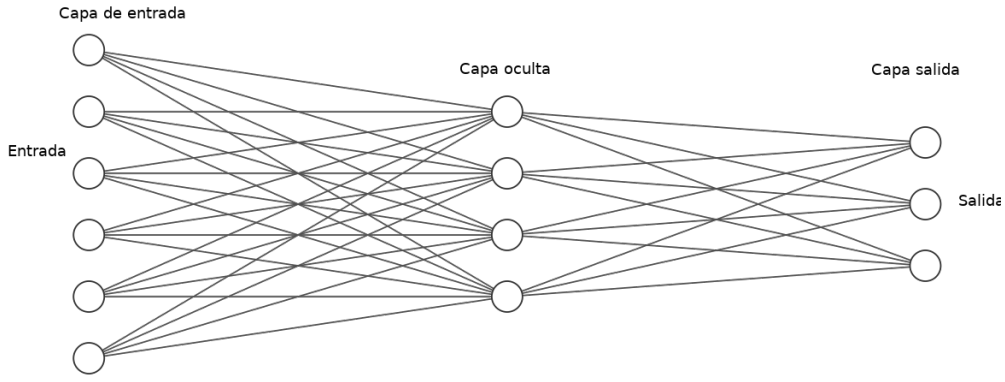


Figura 2.2: Ejemplo de una red neuronal simple.

Las redes neuronales se componen principalmente de los elementos que se definen en forma sencilla en la Tabla 2.1 [Chollet18]:

Elemento	Definición
Capas	Capas que se combinan en una red o modelo
Datos de entrada	Los datos de entrada y objetivos correspondientes
Función de pérdida	Define la señal de retroalimentación para el aprendizaje
Optimizador	Determina cómo procede el aprendizaje

Tabla 2.1: Princpales elementos que componen una red neuronal y su definición más simple.

2.4.1. Función de activación de las redes neuronales

La función de activación decide qué neurona será activada o no (véase la ecuación 2.1, en donde W son los pesos y b se conoce cómo el sesgo del perceptrón; $b \equiv$ umbral de la función de activación [Nielsen15]). Esto hará que la información recibida por la neurona se catalogue como relevante o que sea ignorada en el caso que así lo especifique la función de activación.

$$Y = \sum(W * entrada) + b \tag{2.1}$$

La función de activación es una transformación no lineal que nosotros hacemos sobre la señal de entrada. Esta salida se envía a la siguiente capa de neuronas como entrada.

Cuando no tenemos la función de activación, los pesos y sesgos simplemente harían una transformación lineal. Una ecuación lineal es simple de resolver pero está limitada en su capacidad para resolver problemas complejos. Una red neuronal sin una función de activación es esencialmente un modelo de regresión lineal. La función de activación realiza la transformación no lineal de la entrada, lo que la capacita para aprender y realizar tareas más complejas. Nos gustaría que nuestras redes neuronales trabajen en tareas complicadas como traducciones de idiomas y clasificaciones de imágenes. Las transformaciones lineales nunca podrían realizar tales tareas.

Las funciones de activación hacen posible la propagación hacia atrás ya que los gradientes se suministran junto con el error para actualizar los pesos y sesgos. Sin la función no lineal diferenciable, esto no sería posible. La literatura señala que el sesgo de un perceptrón es una medida de que tan fácil es lograr que la salida se active, es decir, tenga un valor de 1 [Nielsen15].

2.4.2. Función de paso binario

El primer intento cuando tenemos una función de activación sería un clasificador basado en el umbral, es decir, si la neurona debe activarse o no [Gupta17]. Si el valor Y está por encima de un valor de umbral dado, active la neurona; de lo contrario, déjelo

desactivado, descrito por la ecuación 2.2.

$$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \tag{2.2}$$

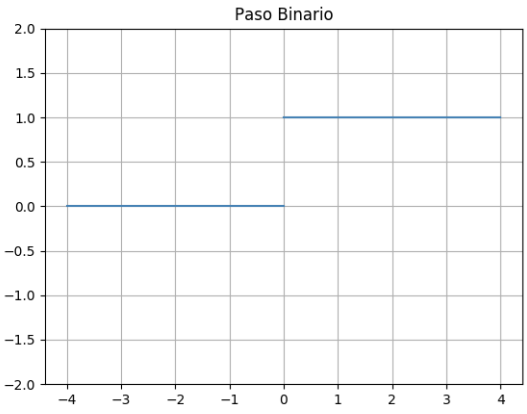


Figura 2.3: Función de paso binario

La función binaria es extremadamente simple. Se puede utilizar al crear un clasificador binario. Cuando simplemente necesitamos decir sí o no para una sola clase, la función de paso sería la mejor opción, ya que activaría la neurona o la dejaría en cero.

Por otra parte, el gradiente de la función de paso es cero (véase la Figura 2.4). Esto hace que la función de paso no sea tan útil, ya que durante la propagación hacia atrás, cuando los gradientes de las funciones de activación se envían para cálculos de error donde se mejora y optimizan los resultados. El gradiente de la función de escalón lo reduce todo a cero y la mejora de los modelos realmente no sucede [Gupta17].

$$f'(x) = \begin{cases} 0 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \tag{2.3}$$

La función binaria no es utilizada en el proceso de aprendizaje, por lo expuesto anteriormente, pero es un primer ejemplo que introduce al entendimiento de las funciones de activación, señalando la necesidad de utilizar funciones con un gradiente definido y la posibilidad de aplicarla en un clasificador binario.

Figura 2.4: Gradiente de la función de paso binario

2.4.3. Función lineal

Se vio que el problema con la función de paso binario, es que su gradiente es cero, por lo que es imposible actualizar el gradiente durante la propagación hacia atrás. En lugar de una función de paso simple, se puede intentar usar una función lineal [Gupta17]. La ecuación 2.4 define la función lineal, para cada valor de x la función nos regresa el producto con el valor constante a.

$$f(x) = ax \tag{2.4}$$

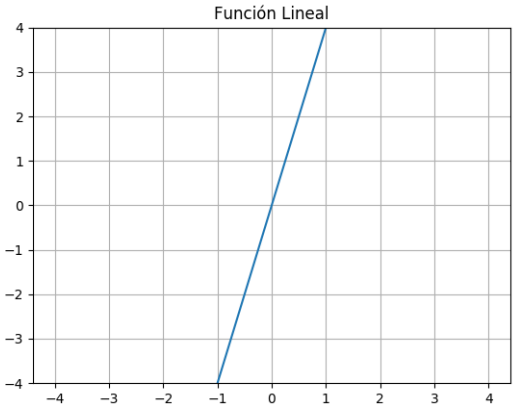


Figura 2.5: Función Lineal, en donde el valor de $a = 4$

En la Figura 2.5, se ha considerado el valor 4 para la constante a. Aquí la activación es proporcional a la entrada. La entrada x, se transformará en ax. Esto se puede aplicar a varias neuronas y se pueden activar múltiples neuronas al mismo tiempo. Ahora, cuando tenemos varias clases en una capa de salida, podemos elegir la que tiene el valor máximo. Pero todavía tenemos un problema aquí, al observar la derivada de esta función en la Figura 2.6. La ecuación 2.5 es el gradiente de la función lineal, un valor constante [Gupta17].

$$f'(x) = a \tag{2.5}$$

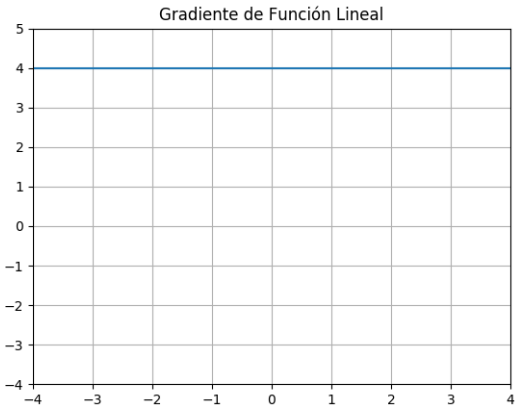


Figura 2.6: Gradiente de la función Lineal

La derivada de una función lineal es constante, es decir, no depende del valor de entrada x. Esto significa que cada vez que hacemos una propagación hacia atrás, el gradiente sería el mismo. Este es un gran problema en la retroalimentación de las neuronas, debido a que realmente no se mejora el error, valor que estima el máximo o mínimo que se está buscando, observándose que el gradiente es prácticamente el mismo. Otra consecuencia que se observa es cuando estamos intentando realizar una tarea complicada para la cual necesitamos múltiples capas en nuestra red. Ahora, si cada capa tiene una transformación lineal, no importa cuántas capas tengamos, la salida final no es más que una transformación lineal de la entrada. Por lo tanto, la función lineal podría ser ideal para tareas simples donde la interpretabilidad es altamente deseada [Gupta17].

2.4.4. Función Sigmoide

La Sigmoide es una función de activación ampliamente utilizada. Está definida por la ecuación 2.6.

$$f(x) = sigmoid(x) = \frac{1}{(1 + e^{-x})} \tag{2.6}$$

La gráfica de la función se muestra en la Figura 2.7.

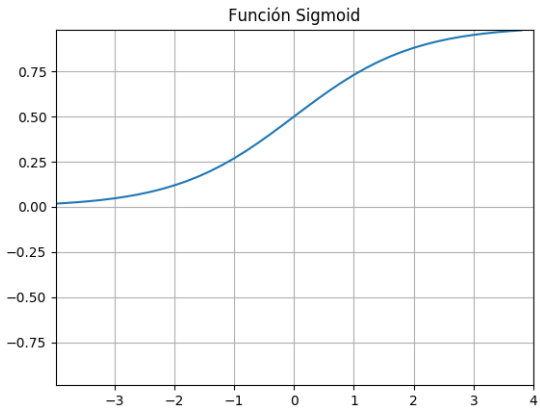


Figura 2.7: Función Sigmoide

Esta es una función suave y es continuamente diferenciable. La mayor ventaja que tiene sobre las funciones descritas anteriormente es que no es lineal. Esta es una característica increíblemente positiva de la función sigmoide y otras funciones. Básicamente, esto significa que cuando se tienen múltiples neuronas que tienen una función sigmoide como su función de activación, la salida tampoco es lineal. La función varía de 0 a 1 con forma de 'S'. Obsérvese la forma de la curva del gradiente de la función en la Figura 2.8. El gradiente es muy alto entre los valores de -3 y 3, pero se vuelve mucho más plano en las regiones fuera de este rango. La utilidad de esto significa que para este rango, pequeños cambios en x también traerían grandes cambios en el valor de la salida Y. Por lo tanto, la función esencialmente intenta empujar los valores de Y hacia los extremos. Esta es una calidad muy deseable cuando estamos tratando de clasificar los valores de una clase en particular [Gupta17].

$$f'(x) = g(x) = sigmoid(x) * (1 - sigmoid(x)) \tag{2.7}$$

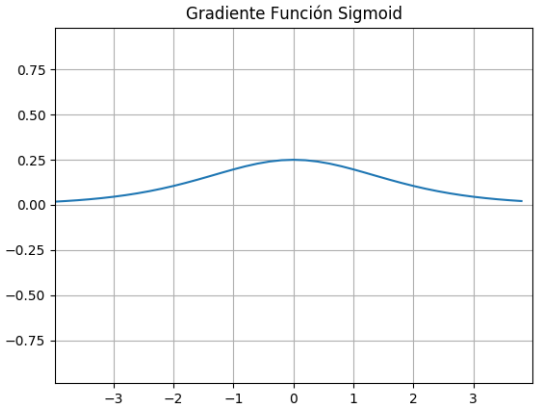


Figura 2.8: Gradiente de la función Sigmoide

La pendiente es suave y depende de x. Esto significa que durante la propagación hacia atrás podemos usar esta función fácilmente. El error se puede propagar a la inversa y los pesos se pueden actualizar en consecuencia.

Las funciones sigmoide son muy utilizadas incluso hoy en día, pero aún existen problemas que se deben resolver. Como vimos anteriormente, la función es bastante plana más allá de las regiones +3 y -3. Esto significa que una vez que la función cae en esa región, los gradientes se vuelven muy pequeños ocasionando que el gradiente se esté acercando a cero y la red no está realmente aprendiendo [Gupta17].

Otro problema que sufre la función sigmoide es que los valores solo varían de 0 a 1. Esto significa que la función sigmoide no es simétrica alrededor del origen y que los valores recibidos son todos positivos. Entonces, no siempre se desea que los valores que van a la siguiente neurona sean todos del mismo signo. Esto se puede resolver escalando la función sigmoide [Gupta17].

2.4.5. Función Tangente Hiperbólica (tanh)

La función tanh es muy similar a la función sigmoide (véase la Figura 2.9). En realidad es solo una versión a escala de la función sigmoide, funciona similar a la función sigmoide pero con la simetría en el origen y los rangos de -1 a 1, la ecuación 2.8 define la función tanh [Gupta17].

$$tanh(x) = \frac{2}{(1 + e^{-2x}) - 1} \tag{2.8}$$

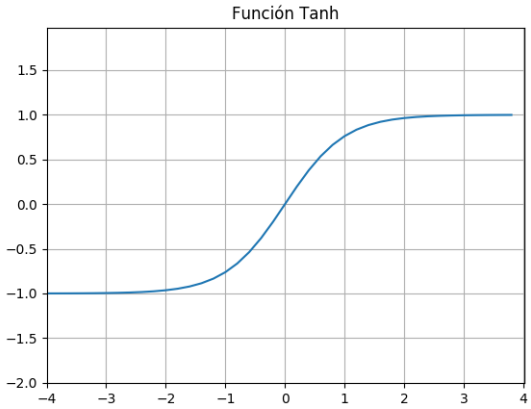


Figura 2.9: Función Tanh,a diferencia de la función sigmoide su salida va en el rango -1 a 1.

Básicamente resuelve el problema de que los valores son todos del mismo signo. Todas las demás propiedades son las mismas que las de la función sigmoide. Es continua y diferenciable en todos los puntos. Se puede ver que la función no es lineal, por lo que permite fácilmente repartir los errores [Gupta17].

La Figura 2.10 nos muestra la gráfica de la pendiente o gradiente de la función tanh.

$$f'(x) = g(x) = 1 - tanh^2(x) \tag{2.9}$$

El gradiente de la función tanh es más pronunciado en comparación con la función sigmoide. Nuestra elección de usar sigmoide o tanh dependería básicamente del requisito

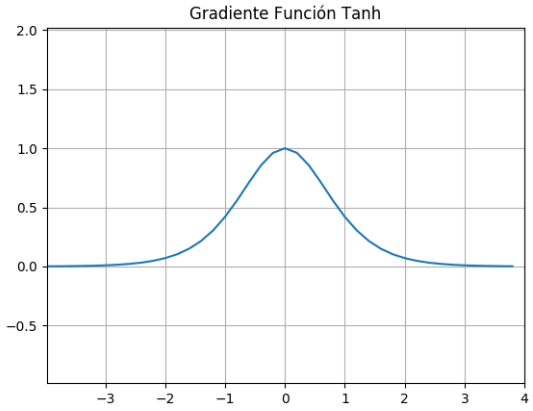


Figura 2.10: Gráfica de la pendiente de la función Tanh.

de gradiente en la declaración del problema. Pero similar a la función sigmoide todavía tenemos el problema del gradiente de fuga, es decir, en los valores extremos de la función el gradiente tiende a cero. En otras palabras, la gráfica de la función tanh es plana en los extremos y los gradientes son valores muy bajos [Gupta17].

2.4.6. Función ReLU

Es la función de activación más utilizada en los modelos neuronales. Se define por la ecuación 2.10 [Gupta17].

$$f(x) = max(0, x) \tag{2.10}$$

Se puede representar gráficamente como se ilustra en la Figura 2.11.

ReLU es la función de activación más utilizada al diseñar redes en la actualidad. Esta función no es lineal, lo que significa que podemos fácilmente propagar los errores y tener múltiples capas de neuronas activadas por la función ReLU.

La principal ventaja de usar la función ReLU sobre otras funciones de activación es que no activa todas las neuronas al mismo tiempo. Si se observa la función ReLU, para valores de entrada negativos, los convertirá a cero y la neurona no se activará. Esto significa que, en un momento dado, solo unas pocas neuronas se activan, lo que hace que la red sea escasa, y por o tanto la hace eficiente y fácil de calcular, algo muy deseable en el proceso

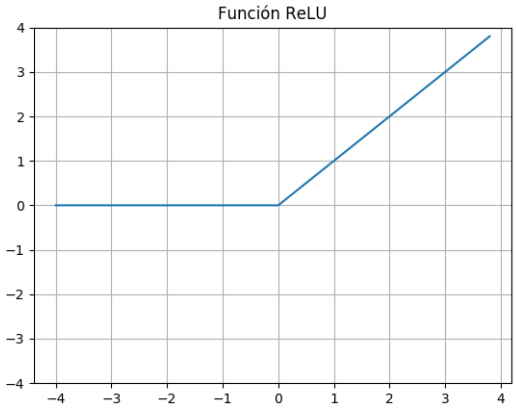


Figura 2.11: Función ReLU

de aprendizaje [Gupta17].

Se puede observar el gradiente de la función ReLU en la Figura 2.12.

$$f'(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \tag{2.11}$$

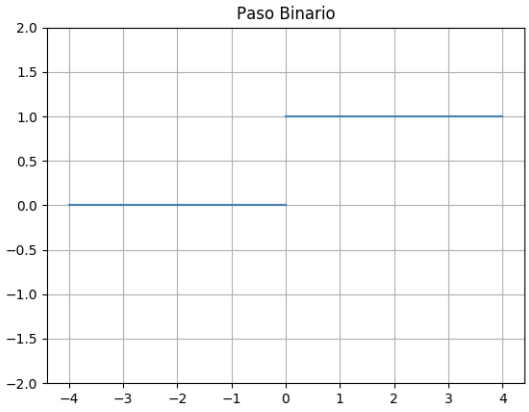


Figura 2.12: Gradiente de la función ReLU

Pero la función ReLU también presenta el mismo problema de los gradientes, que se mueven hacia cero. Si se observa el lado negativo del gráfico en la Figura 2.12, el gradiente

es cero, lo que significa que para las activaciones en esa región, el gradiente es cero y los pesos no se actualizan durante la propagación hacia atrás. Esto puede crear neuronas muertas que nunca se activan. Para superar esta problemática se ha propuesto la función de activación Leaky ReLU, descrita en la Sección 2.4.7 [Gupta17].

2.4.7. Función Leaky ReLU

La función Leaky ReLU no es más que una versión mejorada de la función ReLU. Como se vio para la función ReLU, el gradiente es 0 para $x < 0$, lo que hace que las neuronas mueran por activaciones en esa región. Leaky ReLU se define para abordar este problema. En lugar de definir la función ReLU como 0 para x menor que 0, la definimos como un pequeño componente lineal de x. La ecuación 2.12 define la función Leaky ReLU, en donde a es un valor constante [Gupta17].

$$f(x) = \begin{cases} ax & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \tag{2.12}$$

Lo que hemos hecho aquí es que simplemente se ha reemplazado la línea horizontal con una línea no cero, no horizontal. Aquí a es un valor pequeño como 0.01 o menos [Gupta17]. Un ejemplo de la función Leaky ReLU se representa en la gráfica 2.13.

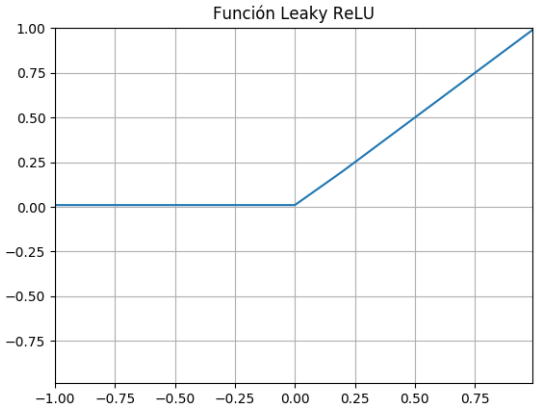


Figura 2.13: Función Leaky ReLU

La principal ventaja de reemplazar la línea horizontal es eliminar el gradiente cero. Entonces, en este caso, el gradiente del lado izquierdo del gráfico no es cero, por lo que ya no encontraremos neuronas muertas en esa región [Gupta17]. El gradiente de la función Leaky ReLU se muestra gráficamente en la Figura 2.14 .

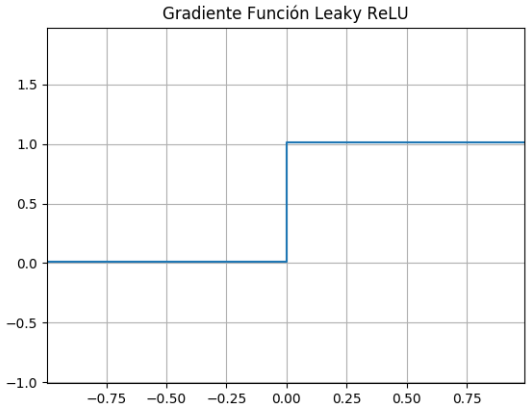


Figura 2.14: Gradiente de la función Leaky ReLU

Sin embargo, en el caso de una función ReLU parametrizada, a también es un parámetro entrenable. La red también aprende el valor de "a" para una convergencia más rápida y óptima. La función ReLU parametrizada se usa cuando la función Leaky ReLU aún no puede resolver el problema de las neuronas muertas y la información relevante no se pasa con éxito a la siguiente capa [Gupta17].

2.4.8. Función Softmax

La función softmax también es un tipo de función sigmoidea pero es útil cuando tratamos de manejar problemas de clasificación. La función sigmoidea como vimos anteriormente fue capaz de manejar solo dos clases. La función softmax extraerá las salidas para cada clase entre 0 y 1 y también se dividiría por la suma de las salidas. Esto da esencialmente la probabilidad de que la entrada esté en una clase particular [Gupta17]. Se puede definir como:

$$f(x) = softmax(x)_i = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1, ..., K \tag{2.13}$$

Por ejemplo tenemos las salidas [1.2, 0.9, 0.75], cuando aplicamos la función softmax obtendríamos [0.42, 0.31, 0.27]. Así que ahora podemos usarlos como probabilidades para que el valor esté en cada clase. La función softmax se usa idealmente en la capa de salida del clasificador donde realmente estamos tratando de alcanzar las probabilidades para definir la clase de cada entrada.

2.4.9. Propagación hacia atrás

Para descubrir los pesos óptimos para las neuronas, realizamos un pase hacia atrás, retrocediendo desde la predicción de la red a las neuronas que generaron esa predicción. Esto se llama propagación hacia atrás. La propagación hacia atrás rastrea las derivadas de las funciones de activación en cada neurona sucesiva, para encontrar ponderaciones que reduzcan al mínimo la función de pérdida, lo que generará la mejor predicción. Comienza con el valor de pérdida final y funciona hacia atrás desde las capas superiores hasta la parte inferior, aplicando la regla de la cadena para calcular la contribución que cada parámetro tuvo en el valor de pérdida. Este proceso matemático también es llamado "Descenso de Gradiente" [Chollet18].

2.5. Optimización del gradiente

Los "pesos" (parámetros entrenables) contienen la información aprendida por la red al exponerse contra los datos de entrada, inicialmente estos pesos son rellenados con valores aleatorios, por esta razón la primera aproximación sera errónea o sin sentido, pero dará un punto de partida. Para después gradualmente ajustar estos pesos, basados en la señal de retroalimentación. Este ajuste gradual, también es llamado "entrenamiento", esto es básicamente el "aprendizaje" en el aprendizaje profundo. Esto sucede en el "entrenamiento cíclico" que funciona de la siguiente manera:

- Generar los datos de entrenamiento y sus objetivos

- Correr la red para obtener la predicción
- Calcular la "pérdida" con respecto a los objetivos
- Actualizar los "pesos" de la red en forma que reduzca la "pérdida"

Eventualmente terminaremos con la red que tiene muy poca "pérdida" con los datos de entrada. Por lo tanto la red habrá "aprendido" al asignar sus entradas a los objetivos correctos [Chollet18].

Un enfoque para actualizar los pesos de una manera eficiente es tomar ventaja del hecho de que todas las operaciones utilizadas en la red son diferenciables, y computan el gradiente de la pérdida con respecto a los coeficientes de la red. Luego puede mover los coeficientes en la dirección opuesta al gradiente, lo que disminuye la pérdida.

En general los sistemas de aprendizaje de máquina usan tensores como su estructura de dato base. Como su base, un tensor es un contenedor para los datos. Si se está familiarizado con matrices, estas son tensores 2D, un tensor es una generalización de matrices de un numero arbitrario de dimensiones. Note que para un tensor una dimensión es un eje [Chollet18].

Por lo tanto, gradiente es la derivada de la operación del tensor. Es la generalización del concepto de derivada a las funciones de entradas multidimensionales, es decir, a funciones que toman los tensores como entradas [Chollet18].

2.6. Aprendizaje profundo

El aprendizaje profundo es un subcampo específico del aprendizaje automático, se refiere a una nueva forma en la representación del aprendizaje de datos, que pone énfasis en el aprendizaje a través de capas sucesivas en un modelo neuronal que obtiene representaciones significativas de los datos. El aprendizaje profundo no es una referencia a ningún tipo de comprensión más profunda lograda por el enfoque, más bien, representa la idea de éxito de capas sucesivas de representaciones de los datos. Por otra parte, las capas que contribuyen al modelo de los datos es llamado "la profundidad del modelo". Para nuestro propósito, el

aprendizaje profundo es un marco de trabajo matemático para aprender representaciones desde los datos [Chollet18].

En la programación clásica, en el paradigma de la Inteligencia Artificial simbólica, los humanos introducen las reglas (un programa) y los datos, y así se obtienen las respuestas. Con el aprendizaje profundo, los humanos introducen los datos así como las respuestas esperadas de los datos, y así obtienen las reglas. Estas reglas pueden ser aplicadas con nuevos datos para producir una respuesta original [Chollet18].

Las raíces del aprendizaje profundo que se definen en [Chollet18] son:

- Reconocimiento de patrones estadísticos, teoría de control adaptativo.
- IA, descubrir reglas usando la decisión de arboles, programación de lógica inductiva.
- Modelos cerebrales, como las redes neuronales.
- Modelos psicológicos.
- Estadística.

El aprendizaje profundo está catalogado como un subcampo del aprendizaje de máquina, este último perteneciente al campo de la IA, véase la Figura 2.15.

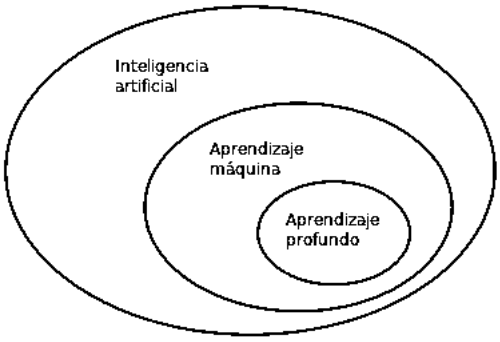


Figura 2.15: Se muestra en donde se encuentra ubicado el subcampo de aprendizaje profundo [Chollet18].

2.6.1. Cómo trabaja el aprendizaje profundo

Los pasos a seguir para realizar el aprendizaje profundo son los siguiente [Salcedo18]:

- Recolección de datos - En esta etapa se generan u obtienen los datos a ser clasificados.
- Preparación de los datos - En esta etapa se separan y desechan los datos erróneos.
- Elegir el modelo a usar - Se selecciona el modelo con el cual se trabajara.
- Entrenar la máquina - Se procede a entrenar la máquina .
- Evaluar los datos obtenidos - Se verifica que el error no sea alto.
- Configurar parámetros si los resultados no son los esperados.
- Realizar la predicción a base del modelo generado.

En la Figura 2.16 se muestra como trabaja el aprendizaje profundo en donde tras una entrada de datos estos son procesados en las capas para dar un resultado el cual será puesto a prueba contra uno verdadero, después del análisis la función de pérdida arrojará una optimización al peso ingresado a las capas para obtener un mejor resultado.

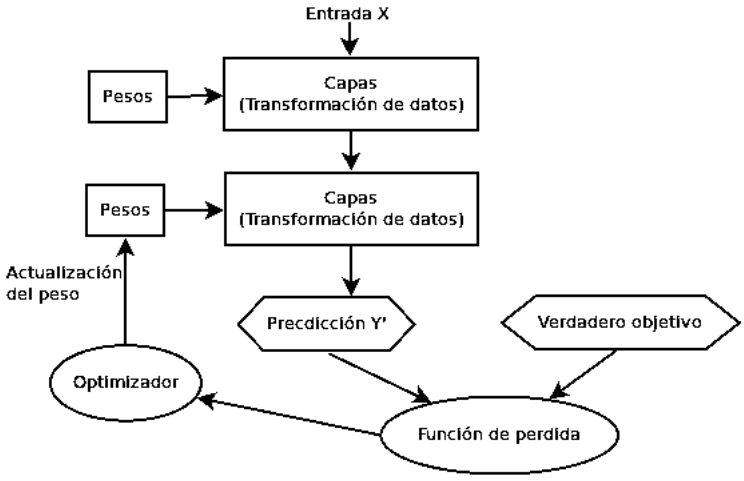


Figura 2.16: Se muestra como trabaja el aprendizaje profundo [Chollet18].

2.7. Visión Computacional

La visión por computadora es la transformación de datos de una cámara fotográfica o vídeo en una decisión o una nueva representación. Todas estas transformaciones se

realizan para lograr algunos objetivos particulares. Los datos de entrada pueden incluir alguna información contextual como "la cámara se encuentra en un automóvil" o "el rango del láser indica que el objetivo se encuentra a 1 metro de distancia". La decisión podría ser "hay una persona en esta escena" o "hay 14 células tumorales en esta diapositiva". Una nueva representación podría significar convertir una imagen de color en escala de grises o eliminar el movimiento de la cámara de una secuencia de imágenes [Bradski08].

Un sistema de visión computacional se compone principalmente de las siguientes etapas:

1. Captura de información (imágenes)
2. Pre-procesamiento (mejora de la imagen)
3. Segmentación (identificación de regiones de interés)
4. Clasificación (Aprendizaje de Máquina)

El aprendizaje profundo descubre una estructura compleja de la información en grandes conjuntos de datos mediante el uso del algoritmo de retropropagación para indicar cómo una máquina debe cambiar sus parámetros internos que se utilizan para calcular la representación en cada capa a partir de la representación en la capa anterior. Las redes convolucionales profundas han logrado avances en el procesamiento de imágenes, vídeo, voz y audio [LeCun15]. Una de las principales aplicaciones de las redes convolucionales ha sido en visión computacional, esencialmente porque han ayudado a encontrar en forma automática las características que permiten discriminar los objetos en una escena, es decir, apoyar en la tarea de clasificación en un sistema de visión computacional.

Capítulo 3

Redes Neuronales Convolucionales (CNN)

3.1. Arquitecturas para el aprendizaje profundo

En el aprendizaje profundo existen varias arquitecturas, principalmente dos de ellas son las que se observan en la naturaleza: redes neuronales convolucionales(CNN) para el modelado de imágenes y redes de memoria a corto plazo (LSTM Long Short-Term Memory) (redes recurrentes) utilizadas en el modelado secuencial [Patterson17].

3.1.1. Redes neuronales convolucionales (Convolutional Neural Networks CNN)

En redes neuronales, la red neuronal convolucional (ConvNets o CNN) es una de las categorías principales que se utilizan ampliamente en áreas tales como: el reconocimiento de imágenes, clasificación de imágenes, la detección de objetos, el reconocimiento de rostros, entre otras. La funcionalidad en que están basadas las CNN's consisten a grandes rasgos en filtrar una imagen usando máscaras, diferentes máscaras generarán diferentes resultados.

El objetivo de una CNN es aprender características de orden superior en los datos a través de convoluciones. Se adaptan bien al reconocimiento de objetos con imágenes. Pueden identificar rostros, individuos, letreros de calles, ornitorrincos y muchos otros aspectos de los

datos visuales. Las CNN se superponen con el análisis de texto mediante el reconocimiento óptico de caracteres, pero también son útiles cuando se analizan las palabras como unidades de texto discretas, además han resultado ser buenas analizando el sonido [Patterson17].

La eficacia de las CNN en el reconocimiento de imágenes es una de las razones principales por las que el mundo reconoce el poder del aprendizaje profundo. Como se puede observar en la Figura 3.1, se ilustra como una imagen pasa a través de varias capas que son parte del proceso de las CNN's las cuales son buenas para la construcción y obtención de características invariantes a la rotación y posición en las imágenes a partir de datos en bruto, una aplicación en visión computacional de las redes CNN.

Figura 3.1: CNN's y visión computacional. Imagen obtenida de [Patterson17]

Comúnmente una imagen puede fácilmente tener 300 píxeles de ancho por 300 píxeles de altura con 3 canales de información para el color RGB (de sus siglas en Inglés Red, Green and Blue). Esto crearía 270,000 pesos de conexión por neurona oculta. Esto muestra la rapidez con la que una red multicapa completamente conectada crea una gran cantidad de conexiones al modelar datos a partir de una imagen. La estructura de los datos de una imagen nos permite cambiar la arquitectura de una red neuronal de manera que podamos aprovechar esta estructura. Con las CNN, podemos organizar las neuronas en una estructura tridimensional para la cual tenemos anchura, altura y profundidad los cuales son

atributos que coinciden con la estructura de una imagen que serían ancho de la imagen en píxeles, altura de la imagen en píxeles y el canal RGB según la profundidad.

Podemos considerar que esta estructura es un volumen tridimensional de neuronas. Un aspecto significativo de cómo las CNN evolucionaron a partir de variantes en las arquitecturas de redes neuronales con retroalimentación y cómo se ha logrado la eficiencia computacional con los nuevos tipos de capas en 3 dimensiones. Técnicamente, los modelos CNN de aprendizaje profundo en su etapa de aprendizaje y prueba, requieren que cada imagen de entrada pase a través de una serie de capas de convolución con filtros (kernels), capas de agrupación, capas totalmente conectadas (Full Connected) y finalmente pasar por una función para clasificar un objeto, como la función Softmax que proporciona valores probabilísticos entre 0 y 1 para identificar la clase. La Figura 3.2 nos muestra un flujo completo de la estructura de una CNN, señalando tipos de capas en donde se realiza la etapa de aprendizaje de características de alto nivel y la etapa de clasificación. De forma general las capas en una CNN se dividen en las siguientes:

- Capa de entrada (recibe la imagen).
- Capas de extracción de características (“Aprendizaje”)
- Capas de clasificación (totalmente conectadas, softmax)

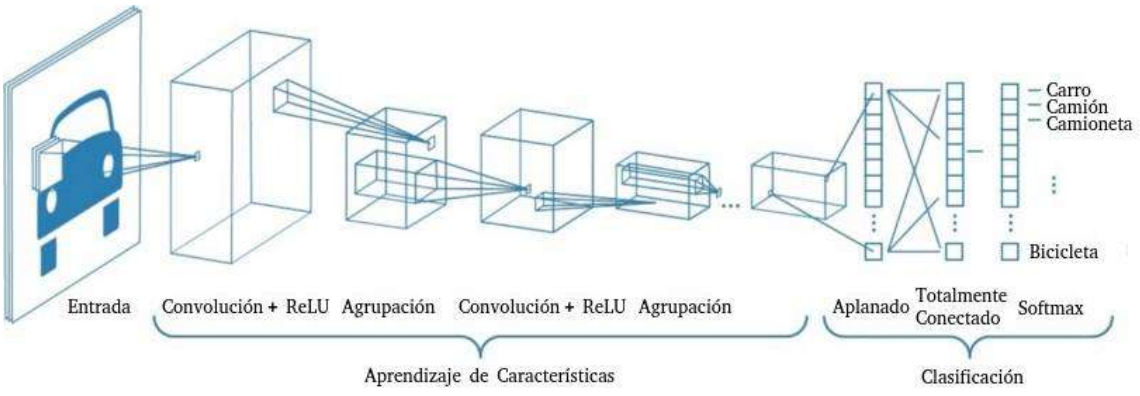


Figura 3.2: Ejemplo de una red con muchas capas convolucionales. Los filtros se aplican a cada imagen de entrenamiento con diferentes resoluciones, y la salida de cada imagen convolucionada se usa como entrada a la siguiente capa, imagen obtenida de [mat19].

Existen muchas variaciones de arquitecturas CNN, pero se basan en el patrón de capas secuenciales que se señalo anteriormente y como se observa en la Figura 3.2.

La capa de entrada - Acepta una entrada tridimensional generalmente en la forma espacial del tamaño (ancho x alto) de la imagen y tiene una profundidad que representa los canales de color (generalmente tres para los canales de color RGB). Es donde cargamos y almacenamos los datos de entrada sin ningún procesamiento de la imagen, para pasarlos a las capas de aprendizaje de la red.

Mapa de características (mapa de activación) - La activación es un resultado numérico si una neurona decide dejar pasar la información. Esta es una función de las entradas a la función de activación, los pesos en las conexiones (para las entradas y el tipo de función de activación en sí). Cuando decimos que el filtro se "activa", queremos decir que el filtro permite que la información pase a través del volumen de entrada al volumen de salida. Deslizamos cada filtro a través de las dimensiones espaciales (ancho, alto) del volumen de entrada durante el paso de información hacia adelante a través de la CNN. Esto produce una salida bidimensional llamada mapa de activación para ese filtro específico. La Figura 3.3 muestra cómo este mapa de activación se relaciona con el concepto que define una característica compleja o de alto nivel. La característica convolucionada es el resultado de aplicar alguno de los filtros, que mapeado en un escala de grises nos resulta el mapa de activación, el cual puede ser utilizado en capas subsecuentes para determinar el estado de las siguientes neuronas, en alguna literatura relacionada lo identifica o se refieren a él como mapa de características.

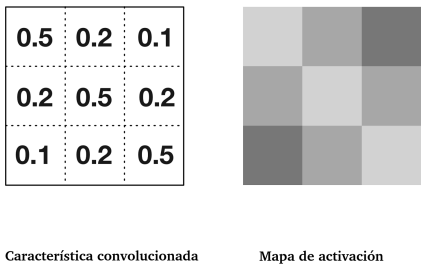


Figura 3.3: Convoluciones y mapa de activación [Patterson17]

Las capas de extracción de características - Tienen un patrón general que se basa

en la siguiente secuencia:

- **Capa de convolución.** Son consideradas los bloques de construcción centrales de las arquitecturas CNN. Como lo ilustra la Figura 3.4 , las capas convolucionales transforman los datos de entrada utilizando un bloque de neuronas conectadas localmente desde la capa anterior. La capa calculará un producto punto entre la región de las neuronas en la capa de entrada y los pesos a los que están conectadas localmente en la capa de salida. La salida resultante generalmente tiene las mismas dimensiones espaciales (o dimensiones espaciales más pequeñas) pero a veces aumenta el número de elementos en la tercera dimensión de la salida (dimensión de profundidad) [Patterson17].

- **Capa de agrupación.** La sección de capas de agrupación reduciría la cantidad de parámetros cuando las imágenes son demasiado grandes. Realiza la agrupación espacial de los datos, también se llama submuestreo o pooling, lo que reduce la dimensionalidad del mapa pero conserva la información importante. La agrupación espacial puede ser de diferentes tipos:

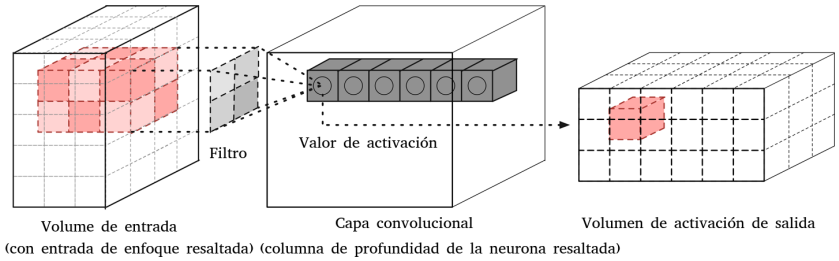


Figura 3.4: Capa de convolución con volúmenes de entrada y salida tomado del libro [Patterson17]

  - Max Pooling (agrupación máxima): Toma el elemento más grande del mapa de características. [Nielsen15].
  - Average Pooling (Promedio): Toma el promedio del mapa de características [Nielsen15].
  - Sum pooling (Suma): Suma de todos los elementos en el mapa de características [Nielsen15].

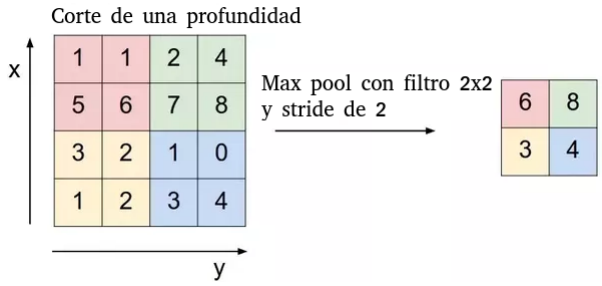


Figura 3.5: Operación Max Pooling en una capa donde se toma el maximo de cada sección [Moroshko18].

Estas capas encuentran una serie de características en las imágenes y construyen progresivamente características de orden superior. Esto corresponde directamente a la habilidad del aprendizaje profundo, mediante el cual las características se aprenden automáticamente en lugar de extraerlas manualmente como lo realizan las técnicas tradicionales.

Capas de clasificación. Es común que en estas capas se tengan una o más capas totalmente conectadas para tomar las características de orden superior y producir probabilidades o puntajes de clase. Estas capas están completamente conectadas a todas las neuronas en la capa anterior, como su nombre lo indica. La salida de estas capas produce típicamente una salida bidimensional de las dimensiones $[b \times N]$, donde b es el número de ejemplos en el mini-lote y N es el número de clases que estamos interesados en clasificar.

Transformamos nuestra matriz de las capas de extracción de características en un vector y la introducimos en una capa completamente conectada como una red neuronal.

En el Figura 3.6, la matriz del mapa de características se convertirá como vector $(x_1, x_2, x_3, ...)$. Con las capas totalmente conectadas, combinamos estas características para crear un modelo. Finalmente, tenemos una función de activación como softmax o sigmoide para clasificar las salidas (y_1, y_2, y_3) como gato, perro, carro, camión, etcétera.

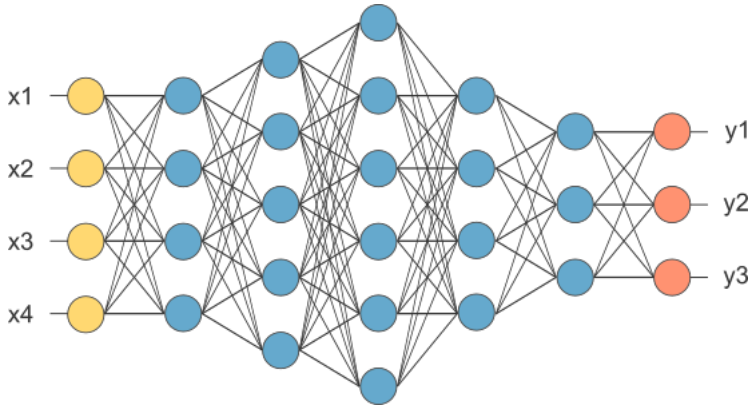


Figura 3.6: Transformación de nuestro mapa de características a un vector en la red (capa completamente conectada), para generar la etapa de clasificación [und18].

3.2. Operaciones de las capas convolucionales

3.2.1. Convolución

La convolución se define como una operación matemática que describe una regla sobre cómo combinar dos conjuntos de información. Es importante tanto en física como en matemáticas y define un puente entre el dominio espacio / tiempo y el dominio de la frecuencia mediante el uso de transformadas de Fourier. Toma su entrada, aplica un kernel de convolución y nos da un mapa de características como salida.

La operación de convolución, que se muestra en la Figura 3.7, se conoce como el detector de características de una CNN. La entrada a una convolución puede ser datos sin procesar o una salida del mapa de características de otra convolución. A menudo se interpreta como un filtro en el que el kernel filtra los datos de entrada para ciertos tipos de información; por ejemplo, un kernel de borde permite pasar solo información de bordes en una imagen.

La Figura 3.7 ilustra cómo se desliza el kernel a través de los datos de entrada para producir los datos de la característica (salida) entrelazados. En cada paso, el kernel se multiplica por los valores de los datos de entrada dentro de sus límites, creando una sola entrada en el mapa de características de salida. En la práctica, la salida es grande si la característica que estamos buscando se detecta en la entrada.

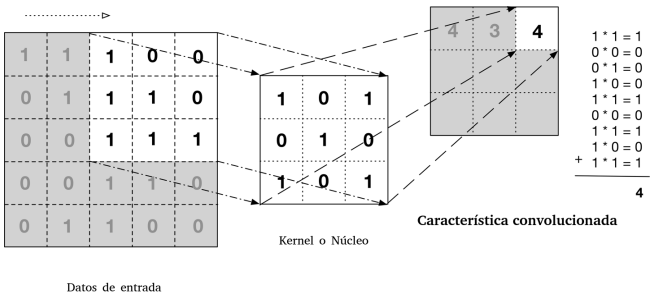


Figura 3.7: Operación de convolución. Imagen obtenida de [Patterson17]

Por lo general, nos referimos a los conjuntos de pesos en una capa convolucional como un filtro (o núcleo). Este filtro está convolucionado con la entrada y el resultado es un mapa de características (o mapa de activación). Las capas convolucionales realizan transformaciones en el volumen de datos de entrada que son una función de las activaciones en el volumen de entrada y los parámetros (pesos y sesgos de las neuronas). El mapa de activación para cada filtro se apila a lo largo de la dimensión de profundidad para construir el volumen de salida 3D.

3.2.2. Paso (Strides)

Esta operación configura cuánto se moverá nuestra ventana de filtro deslizante para aplicar la función de filtro en la operación de convolución. Es el número de píxeles desplazados sobre la matriz de entrada. Cuando el desplazamiento es de 1, movemos los filtros a 1 píxel a la vez. Cuando se configura a 2, se mueven los filtros a 2 píxeles a la vez y así sucesivamente como se muestra en la Figura 3.8.

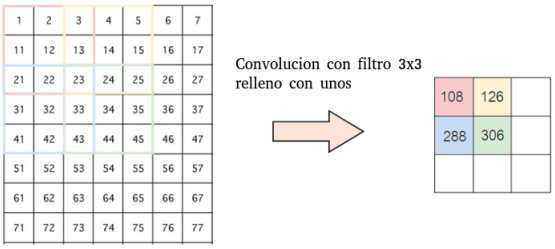


Figura 3.8: Función de convolución con un paso de 2, stride=2 [und18].

3.2.3. Relleno de ceros (Zero-padding)

Cuando el filtro no se ajusta a la imagen de entrada se procede a controlar el tamaño espacial de los volúmenes de salida. Rellena la imagen en los espacios que no ajusta la matriz de entrada con ceros (cero relleno) con la finalidad de que se ajuste, ver la Figura 3.9. El ajuste se refiere a mantener las dimensiones adecuadas o congruentes para la siguiente capa en la arquitectura neuronal profunda.

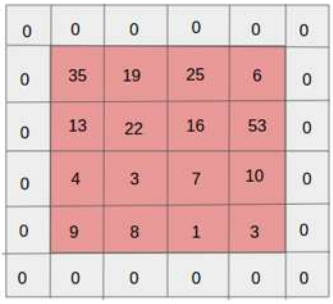


Figura 3.9: Operación Zero-padding [Patterson17].

3.2.4. Activación (función ReLU)

Es la Unidad lineal rectificada para una operación no lineal. La salida es $f(x) = max(0, x)$. El propósito de ReLU es introducir la no linealidad en nuestra red convolucional, veáse en la Sección 2.4.6, en donde se proporcionó una descripción más detallada del significado de la misma. La Figura 3.10 muestra el efecto en una capa de una CNN, en donde a la matriz de entrada se le aplica la función ReLU y cómo los valores negativos son ajustados a 0 en la matriz resultante.

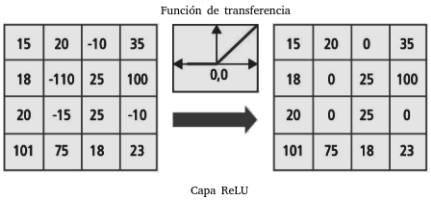


Figura 3.10: Operación ReLU [und18].

Capítulo 4

Arquitectura CNN y Descripción del Conjunto de Datos

4.1. Arquitectura CNN

Las redes CNN nos sirven para generar sistemas de clasificación de imágenes, que se utilizan como modelos de predicción en sistemas del área de visión computacional. La entrada a un modelo es una imagen y esta debe ser clasificada por el modelo CNN. En el Capítulo 3 se describieron las CNN's, los elementos que las componen y como logran el aprendizaje profundo. La arquitectura propuesta de la CNN usada en el presente trabajo se muestra en la Figura 4.1. Consiste de una capa de entrada, que corresponde a una imagen en 3 dimensiones, el ancho, alto y los tres canales de color (RGB) correspondientes a dichas dimensiones. Además de 4 capas de convolución con 32 filtros de 3 x 3 cada una y un Max-Pooling de 2x2 por cada capa de convolución. La salida de las capas convolucionales llegan a una capa densa o totalmente conectada de dimensión 1x64 y la capa final que proporciona la clasificación corresponde a una capa de categorización de dimensión 1x4 con la función softmax, que permite identificar cuatro clases de objetos o tipos de imágenes que se han propuesto en esta Tesis. La implementación de nuestra arquitectura, se realizó con el software keras [Chollet15], cuyas características se describen en la Sección 4.2. Este sofware nos permite obtener una descripción de la cantidad de parámetros y las dimensiones que

cada una de las capas adquiere después de la operación Max-Pooling y los filtros utilizados, para nuestra arquitectura se muestra el resumen que proporciona el software utilizado en la creación de una red neuronal, ver el Listado 4.1

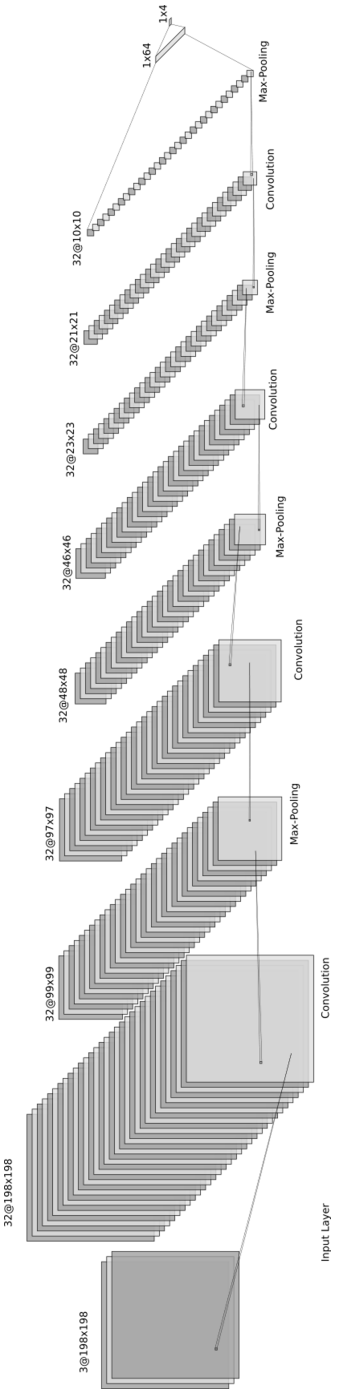


Figura 4.1: Arquitectura CNN propuesta para la clasificación de 4 objetos

La estrategia que se siguió para determinar la arquitectura propuesta fue de ir incrementando capas de convolución. La primer arquitectura utilizada fue de una sola capa de convolución y después se probó el desempeño de la arquitectura, los resultados se muestran en el Capítulo 5. Se observó que al tener una arquitectura con 5 capas de convolución se obtuvo un desempeño inferior a la arquitectura con 3 y 4 capas de convolución, por lo que se decidió mantener la arquitectura mostrada en la Figura 4.1.

Listado 4.1: Resumen de la implementación de una red neuronal en el software keras

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 198, 198, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 198, 198, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 99, 99, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 97, 97, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 97, 97, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 46, 46, 32)        9248
_________________________________________________________________
activation_3 (Activation)    (None, 46, 46, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 21, 21, 32)        9248
_________________________________________________________________
activation_4 (Activation)    (None, 21, 21, 32)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                204864
_________________________________________________________________
activation_5 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 260
_________________________________________________________________
activation_6 (Activation)    (None, 4)                 0
=================================================================
Total params: 233,764
Trainable params: 233,764
Non-trainable params: 0
```

4.2. Keras

Keras es una API (Interfaz de programación de aplicaciones) de redes neuronales de alto nivel, escrita en Python y capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollado con un enfoque que permite la experimentación rápida. Poder pasar de la idea al resultado con el menor tiempo posible es clave para hacer buena investigación [Chollet15].

Keras es una biblioteca de aprendizaje profundo que:

- Permite realizar prototipos de forma fácil y rápida (a través de la facilidad de uso, modularidad y expansibilidad).
- Admite redes convolucionales y redes recurrentes, así como combinaciones de las dos.
- Funciona a la perfección con un CPU ó con un GPU.

Facilidad de uso: Keras sigue las mejores prácticas para reducir la carga cognitiva: ofrece una API consistente y simple, minimiza el número de acciones de usuario requeridas para los casos de uso comunes y proporciona comentarios claros y procesables en caso de error del usuario.

Modularidad: Un modelo se entiende como una secuencia o un gráfico de módulos independientes y totalmente configurables que se pueden conectar con la menor cantidad de restricciones posible. En particular, las capas neuronales, las funciones de costo, los optimizadores, los esquemas de inicialización, las funciones de activación y los esquemas de regularización son módulos independientes que pueden cambiar para crear nuevos modelos.

Fácil extensibilidad: Los nuevos módulos son fáciles de agregar (como clases o funciones), los módulos existentes brindan amplios ejemplos. Puede crear fácilmente nuevos módulos y permite una expresividad total, lo que hace que Keras sea adecuado para la investigación avanzada.

Trabaja con Python: No hay archivos de configuración de modelos separados en un formato declarativo. Los modelos se describen en el código python, que es compacto, más fácil de depurar y permite la extensibilidad.

4.3. Conjunto de datos

Al igual que en otras técnicas de aprendizaje de máquina, en donde se requiere para entrenar los algoritmos de aprendizaje, primero es necesario obtener una base de conocimientos o conjunto de datos previamente etiquetado, es decir, que cada objeto en el conjunto de datos debe tener asignada la clase a la que pertenece, de igual forma en el aprendizaje profundo se requiere disponer de conocimiento previo. En el presente trabajo se ha construido un conjunto de imágenes que serán utilizadas para el entrenamiento de una red neuronal profunda. La literatura señala que es necesario contar con varias decenas de miles de imágenes, sin embargo algunas arquitecturas se han desempeñado en forma aceptable con cantidades menores a las 2000 imágenes por clase. Las clases que se usarán en el presente trabajo son:

- Escaleras
- Mesas
- Sillas
- Puertas

La idea es aprender a diferenciar estas 4 clases que son muy comunes en ambientes del hogar u oficinas y con ello abordar aplicaciones en un futuro en este ambiente.

4.3.1. Flickr

Flickr (pronunciado flicker) es un sitio web que permite almacenar, ordenar, buscar, vender y compartir fotografías o vídeos en línea, a través de Internet. Cuenta con una comunidad de usuarios que comparten fotografías y vídeos creados por ellos mismos. Esta comunidad se rige por normas de comportamiento y condiciones de uso que favorecen la buena gestión de los contenidos. La popularidad de Flickr se debe fundamentalmente a la capacidad para administrar imágenes mediante herramientas que permiten a los autores: etiquetar sus fotografías, explorar y comentar las imágenes de otros usuarios. Flickr es una herramienta que se puede usar para potenciar las clases de fotografía, cuenta con un API

que permite con algunos lenguajes de programación bajar grandes cantidades de imágenes en base a una consulta dada [Butterfield18]

Con la herramienta Flickr API (Apéndice A) para python se realizó un programa de descarga masiva, ver el código fuente en el Apéndice B, para descargar las imágenes en base a una palabra o frase que se utilizó como consulta en el API. Las palabras utilizadas y la cantidad de imágenes descargadas para cada una de las clases en base a la consulta realizada se muestran en la Tabla 4.1.

Clase	**Consulta**	**Cantidad de imágenes**
Escaleras	stair	58880
	armstair	497
mesas	table	17461
	dinner table	627
puertas	door	35627
	entryway	2249
sillas	chair	26018
	arm chair	497

Tabla 4.1: Palabras de consulta en Flickr API y cantidad de imágenes obtenidas

En el Apéndice A se muestra como realizar la ejecución del programa para la obtención de las imágenes. De la Figura 4.2(a) a la Figura 4.2(d) se ilustran algunos ejemplos de las imágenes que fueron descargadas con el Flickr API para las consultas que se indicaron en la tabla 4.1. La depuración de la información que se realizó consistió en eliminar las imágenes duplicadas y también quitando imágenes que no correspondían al conjunto al objeto a clasificar, quedando finalmente el conjunto con las cantidades indicadas en la Tabla 4.2. El software que se utilizó para depurar los datos fue `fdupes`, en el Apéndice B se indica la forma en como se utilizó para eliminar las imágenes duplicadas.

Clase	Cantidad de imágenes obtenidas
Escalera	3491
Mesas	1286
Puertas	1750
Sillas	1896

Tabla 4.2: Palabras de consulta en Flickr API y depuradas

Se dividió el conjunto de datos en dos grandes grupos: entrenamiento y validación. Se estableció el uso de las cantidades se indican en la Tabla 4.3 para ser usadas en las primeras pruebas. Usando en las pruebas finales un particionado de las imágenes utilizando 80 % para el conjunto de entrenamiento y un 20 % para el conjunto de validación.

Clase	Entrenamiento	Validación
Escalera	1000	200
Mesas	1000	200
Puertas	1000	200
Sillas	1000	200

Tabla 4.3: Cantidades usadas para entrenamiento y validación

(a) ***Ejemplo de Escalera*** **(b)** ***Ejemplo de Mesa***

(c) ***Ejemplo de Puerta*** **(d)** ***Ejemplo de Silla***

Figura 4.2: Ejemplo de imágenes obtenidas por medio de flickr

4.4. Generar imágenes

Dado que para el proceso de entrenamiento de una red neuronal profunda se necesita la obtención de muchos datos, para satisfacer esta necesidad se recurre a la generación de datos aleatorios, en donde podemos agregar mayor iluminación, rotación, traslación, etcétera de las imágenes, así el modelo aprenderá en diferentes situaciones la clasificación de objetos que deseamos. La generación de imágenes se realiza por software, en keras dentro del mismo código B de entrenamiento se permite la generación aleatoria de imágenes, con esto se ahorra la generación real de las mismas, de tal forma que estás se pasarán al entrenamiento.

Es común crear nuevos datos a partir de los ya existentes, a partir de las imágenes obtenidas por medio de flickr se pueden generar más, esto modificando la imagen original por medio de la función `ImageDataGenerator`, a continuación se lista las posibles operaciones de modificación para las imagen que se someten a dicha función.

- Desplazamiento Horizontal (height_shift_range)
- Desplazamiento Vertical (width_shift_range)
- Giro Vertical (vertical_flip)
- Giro Horizontal (horizontal_flip)
- Rotación aleatoria (rotation_range)
- Brillo aleatorio (brightness_range)
- Zoom aleatorio (zoom_range)
- Corte aleatorio (shear_range)

Ejemplos de posibles operaciones de afectación a las imágenes se muestran de la Figura 4.3(e) a la Figura 4.3(f).

Para los datos de entrenamiento se utilizaron solamente las siguientes operaciones:

- Reescalamiento (rescale)

(a) ***Ejemplo de imagen original*** (b) ***Ejemplo de Desplazamiento vertical***

(c) ***Ejemplo de Giro Horizontal*** (d) ***Ejemplo de Rotación***

(e) ***Ejemplo de Desplazamiento horizontal*** (f) ***Ejemplo de Zoom***

Figura 4.3: Ejemplo de imágenes creadas a partir de ImageDataGenerator

- Corte aleatorio (shear_range)
- Zoom (zoom_range)
- Voltear aleatoriamente de forma horizontal (horizontal_flip)

Capítulo 5

Pruebas y Resultados

La resolución de problemas de clasificación de objetos en visión computacional ha mejorado en gran medida por el uso de las redes neuronales convolucionales [Jarrett09, Cireşan12, Chollet18]. Al conjuntar las bases de conocimiento y las arquitecturas propuestas nos sirven para desarrollar sistemas más complejos, esto requiere crear evidencias que prueben los modelos propuestos y que además permitan adquirir experiencia al identificar aquellas pruebas que impactan en los resultados.

Existe una gran cantidad de parámetros y posibilidades en el manejo de las arquitecturas de aprendizaje profundo que impactan al desempeño o aprendizaje de la información, algunos de ellos son:

1. Cantidad de capas en la arquitectura.
2. Dimensión de las imágenes de entrada.
3. Ciclos de entrenamiento o cantidad de épocas.
4. Inicialización de los pesos con modelos previamente entrenados.
5. Cantidad de imágenes en los datos de entrenamiento y validación.
6. Tipos de capas en la arquitectura.
7. Función de activación a utilizar en cada capa.

8. Cantidad de filtros y dimensiones utilizados.

9. Operaciones en las capas (Max-Pooling, stride, dropout).

Teniendo como objetivo la creación de un sistema para la búsqueda e identificación de objetos a través del aprendizaje profundo, se hace necesario la implementación de diferentes pruebas que nos permita proponer un modelo o arquitectura de red neuronal profunda, con un desempeño aceptable. Los objetos que se plantea identificar con el sistema son puertas, sillas, mesas y escaleras. Mediante las pruebas correspondientes se podrá obtener el modelo con el mejor desempeño en la identificación.

Se realizaron las primeras 5 de las 9 posibles pruebas sobre los parámetros que se han enumerado anteriormente, observando el resultado de precisión en los datos de validación. El tipo de capas en la arquitectura no se modificó, manteniendo siempre la función de activación ReLU, la cantidad de filtros en las capas CNN de 32 y dimensiones 3x3 y respecto a las operaciones en las capas, siempre se aplicó Max-Pooling cuya dimensión es 2x2. Las pruebas se realizaron sobre un equipo de cómputo cuyas características se describen en el Apéndice A.1.

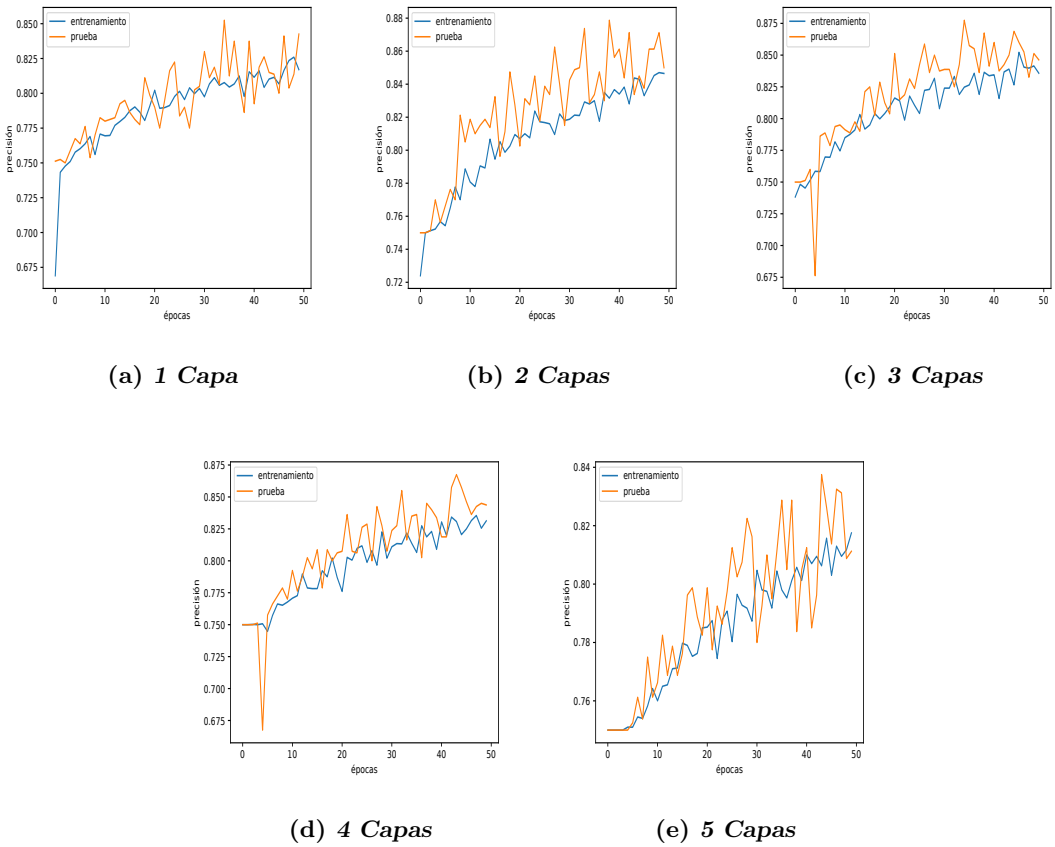
5.1. Prueba 1: Cantidad de capas en la arquitectura

El objetivo de esta prueba es determinar el efecto que tiene en el desempeño del modelo al ir agregando capas de convolución a la arquitectura y con ello determinar la mejor cantidad. Esta primer prueba nos permite verificar mediante evidencia empírica si se mejora la precisión en el resultado al incrementar las capas de convolución al modelo y determinar la arquitectura final. En esta prueba se utilizó la misma cantidad de imágenes en las cuatro clases, para el entrenamiento se emplearon 1000 imágenes en cada clase y para la validación 200 imágenes por clase. El entrenamiento se realizó para 50 épocas en cada arquitectura y con las dimensiones de imágenes de entrada de 120 x 120 píxeles (ancho x alto de cada imagen). La Tabla 5.1 muestra la cantidad de capas de convolución que contiene el modelo y su correspondiente valor de precisión obtenido.

Resultado: al ver los resultados en la Tabla 5.1 se observa que con dos y tres capas de convolución en la arquitectura se obtiene el mejor valor de precisión, lo que nos indica

Capas de Convolución	Tiempo por época	Precisión
1	19 seg	0.85250
2	19 seg	0.87875
3	19 seg	0.87750
4	19 seg	0.86750
5	19 seg	0.83750

Tabla 5.1: Valores de precisión obtenidos a diferentes cantidades de capas de convolución en la arquitectura



(a) ***1 Capa*** **(b)** ***2 Capas*** **(c)** ***3 Capas***

(d) ***4 Capas*** **(e)** ***5 Capas***

Figura 5.1: Gráficas que indican el comportamiento de la exactitud en los datos de entrenamiento y prueba al incrementar de 1 a 5 capas de convolución en la arquitectura

que por el momento no se agregarán más capas para obtener un mejor resultado y es posible elegir entre dos y tres capas de convolución para nuestro modelo. No fue posible superar las 5 capas de convolución pues después de 5 capas de convolución las operaciones de max-pooling

aplicadas, produce que el tamaño se reduzca al menor requerido (32), después de la tercera capa de convolución se observa que el rendimiento va disminuyendo; el tiempo requerido en el entrenamiento en cada época es el mismo independientemente de la cantidad de capas en el modelo, no viéndose afectado el tiempo de ejecución por este factor. En conclusión se usarán tres capas de convolución para proceder con las siguientes pruebas. De la Figura 5.1 se observa que el comportamiento de los modelos es incremental hasta lograr el mejor resultado de precisión y posterior a ello se decrementa su desempeño en los datos de prueba. La tendencia de los resultados con los datos de entrenamiento en las gráficas de la Figura 5.1 nos indican que es posible seguir entrenando más los modelos, consideración que se aplicó en otras pruebas subsecuentes. Lo deseable de las gráficas es que se mantenga paralelos los valores de exactitud en ambos conjuntos de datos.

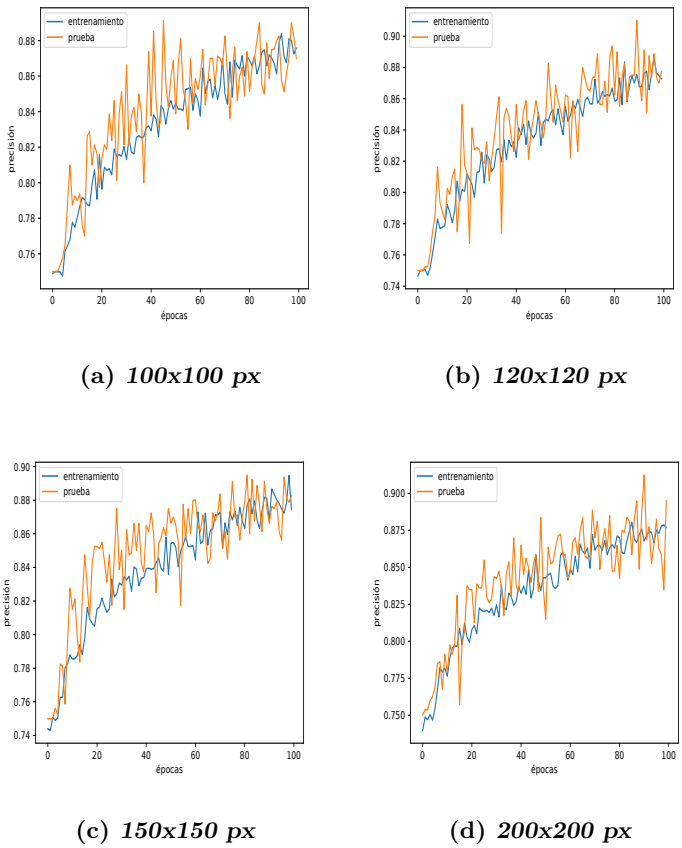
5.2. Prueba 2: Dimensión de las imágenes de entrada

El objetivo de esta prueba es determinar si la cantidad de píxeles (dimensiones de las imágenes) tomados inicialmente afectan al desempeño de nuestro modelo. Para esta prueba se tomaron en cuenta los resultados anteriores usando como base 3 capas de convolución y un total de 100 épocas para tener una estimación mejor. La prueba consiste en ir variando la cantidad de píxeles al incrementar las dimensiones de la imagen de entrada a la red neuronal.

Px	Tiempo por época	Precisión
100x100	18 seg	0.89125
120x120	18 seg	0.91000
150x150	19 seg	0.89500
200x200	23 seg	0.91250

Tabla 5.2: Valores de precisión obtenidos a diferente cantidades de píxeles en la entrada (dimensión de la imagen de entrada)

Resultado: como se puede observar en la Tabla 5.2 que muestra los resultados de precisión y la cantidad de píxeles utilizados en el entrenamiento, se observó que con las dimensiones de 120x120 y 200x200 se encuentra el mejor valor de precisión cuyo valor es de 91 % y 91.25 %, por lo tanto se concluye que es posible utilizar cualquiera de las



(a) ***100x100 px*** **(b)** ***120x120 px***

(c) ***150x150 px*** **(d)** ***200x200 px***

Figura 5.2: Resultados obtenidos en el entrenamiento para diferentes dimensiones de la capa de entrada en la arquitectura (dimensiones de las imágenes)

dos dimensiones de entrada, sin embargo se puede optar elegir la que requiere una menor cantidad de tiempo de entrenamiento por época, en este caso la dimensión 120x120. De las gráficas que se muestran en la Figura 5.2 los mejores comportamientos se observan en las figuras (b) y (d), debido a que se alinean mejor los resultados tanto de los datos de entrenamiento como de prueba. Aún cuando el mejor resultado obtenido fue con las dimensiones 200x200, la gráfica de las dimensiones de 120x120 presenta un comportamiento de mejores valores de precisión en lo general y más alineados a los datos de prueba respecto a los datos de entrenamiento. Se tomó el criterio del mejor resultado y menor tiempo para las pruebas subsecuentes.

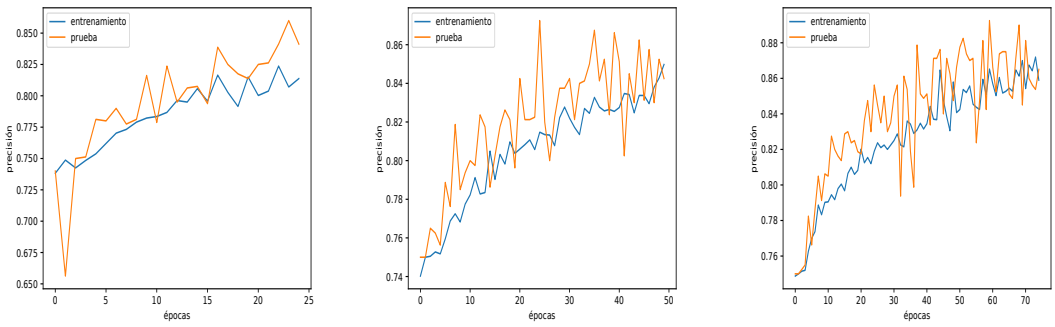
5.3. Prueba 3: Tiempo de entrenamiento o cantidad de épocas

El objetivo de esta prueba es determinar cuántas épocas son necesarias para la obtención de un mejor resultado, además de observar a partir de que época ya no se obtiene una mejora, para detener el proceso de entrenamiento. Para esta prueba se tomó en cuenta el resultado de las pruebas anteriores, donde se selecciono la arquitectura con 3 capas de convolución por el resultado obtenido en la prueba anterior, la entrada de las imágenes con dimensión de 120x120 px. La prueba consiste en ir incrementando el numero de épocas en el proceso de entrenamiento y observar el comportamiento de la precisión y función de pérdidas en esta arquitectura.

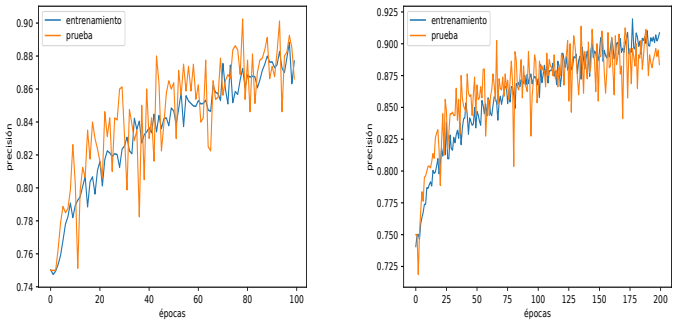
Cantidad de épocas	Tiempo por época	Precisión
25	19 seg	0.86000
50	19 seg	0.87250
75	19 seg	0.89250
100	19 seg	0.90250
200	19 seg	0.91375
300	19 seg	0.93375
500	19 seg	0.93375

Tabla 5.3: A mayor cantidad de épocas el valor de precisión que se obtiene es principalmente incremental.

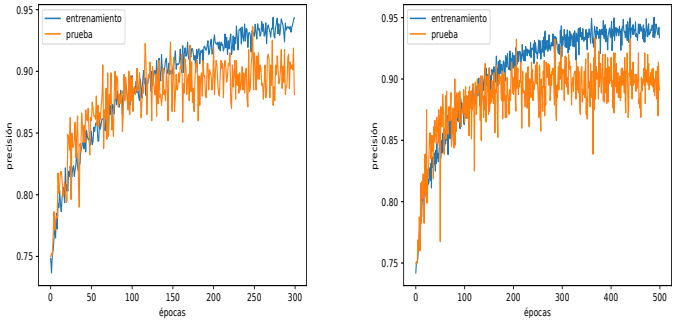
Resultado: la tabla 5.3 muestra la cantidad de épocas utilizadas en el entrenamiento y su correspondiente mejor valor de precisión obtenido. A mayor cantidad de épocas se obtiene un mejor resultado. Como se puede observar en las gráficas de la Figura 5.3, con 200, 300 y 500 épocas el comportamiento de los resultados en los datos de prueba tiende a estabilizarse, a diferencia de los resultados de los datos de entrenamiento que mantienen un comportamiento incremental en la cantidad de épocas anteriores a 200; por lo observado se concluye que es necesario al menos 200 épocas o más para llegar a la mejor aproximación o desempeño del modelo. Con este resultado se retomara la primer prueba para 200 épocas, debido a que el porcentaje de precisión obtenido al variar la cantidad de capas de convolución tiene una diferencia mínima entre 2, 3 y 4 capas, concluyendo que con una cantidad de épocas mayor en el entrenamiento y lo observado en esta prueba pueden ser mejorados los



(a) *25 Épocas* **(b) *50 Épocas*** **(c) *75 Épocas***



(d) *100 Épocas* **(e) *200 Épocas***



(f) *300 Épocas* **(g) *500 Épocas***

Figura 5.3: Resultados obtenidos al variar de 25 épocas a 500 épocas el entrenamiento de la arquitectura

resultados. Se justifica no considerar la prueba de 300 y 500 épocas por el comportamiento al final de las gráficas, que indican un decremento en los resultados y ello lleva a inferir un sobre entrenamiento de los datos.

5.4. Revaloración de la prueba No. 1

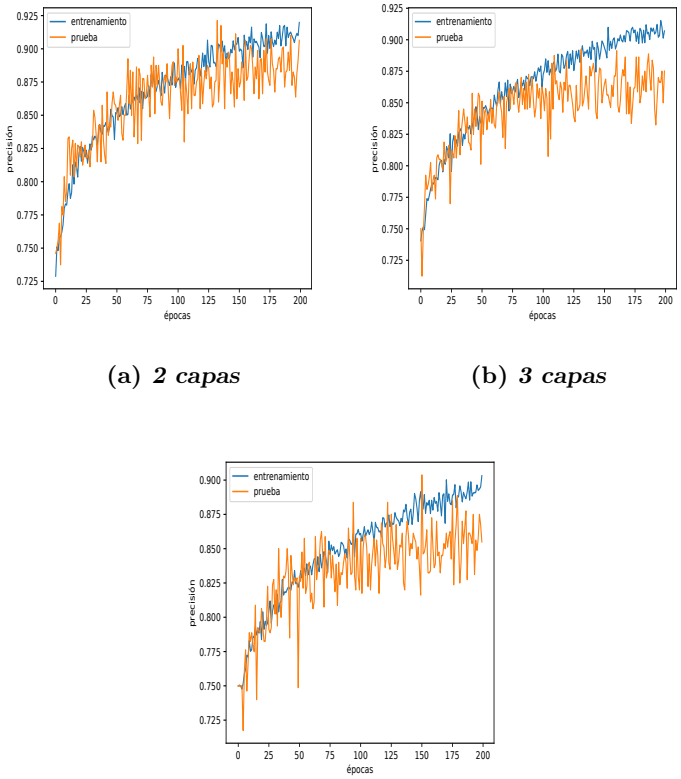
Por lo observado en la prueba anterior (5.3), no se logra definir si con 2, 3 o 4 capas de convolución se obtiene el mejor resultado, pues la diferencia de los resultados de precisión es muy pequeño y no significativo. Se propone repetir la prueba No. 1 con 200 épocas para las cantidades 2, 3 y 4 capas de convolución y entonces determinar cual es la mejor opción u opciones que proporcionen un mejor comportamiento en los resultados.

La prueba consiste en cambiar el número de capas de convolución a utilizar (de 2 a 4 capas), manteniendo fijo el tamaño de entrada de las imágenes, considerando 120x120 píxeles de ancho y alto respectivamente que fue concluido por las pruebas previas.

Cantidad de capas	Tiempo por época	Precisión
2	21 seg	0.92125
3	21 seg	0.92250
4	21 seg	0.91750

Tabla 5.4: Resultados de precisión para diferentes cantidades de capas de convolución considerando 200 épocas y tamaño de imágenes de entrada de 120x120 píxeles.

Resultado: como se observa en la Tabla 5.4 y en las gráficas de la Figura 5.4, el utilizar 3 capas de convolución en la arquitectura, se obtiene un mejor resultado, la diferencia entre 2 y 3 capas de convolución es de aproximadamente el 0.1 %, mientras que con 4 capas es de alrededor del 1 % con respecto a las demás. El comportamiento de los resultados en las gráficas que se observan con la tendencia incremental tanto para los datos de entrenamiento como los de validación es cuando se emplean 2 y 4 capas en la arquitectura. Por lo que se determina utilizar la arquitectura con 3 capas de convolución por obtener el mejor resultado. Y por el comportamiento incremental presentado por la arquitectura con 4 capas, el cual indica una mayor tendencia en los datos de entrenamiento, se decide incluirlo en las pruebas. La arquitectura con 2 capas se considera incluida dentro del resultado obtenido con 3 capas.



(a) ***2 capas*** **(b)** ***3 capas***

(c) ***4 capas***

Figura 5.4: Gráficas con los resultados de precisión en el entrenamiento de la arquitectura con 2, 3, y 4 capas de convolución en 200 épocas.

5.5. Prueba 4.-Inicialización de los pesos con modelos previamente entrenados

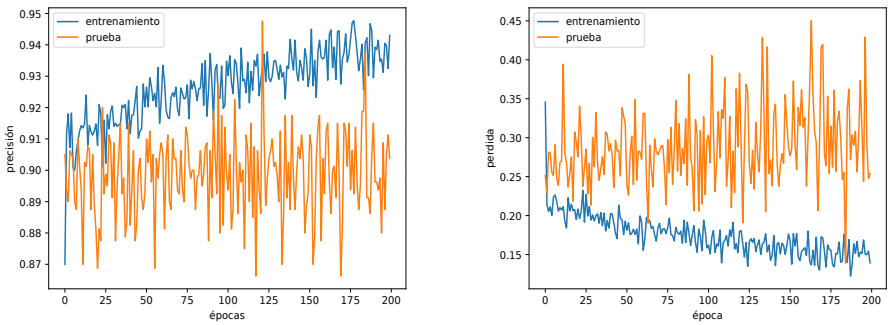
A partir de los resultados anteriores se tomó la decisión de usar 3 capas convoluciones, tomando 120x120 px para la capa de entrada, y el uso de 200 épocas, durante el proceso de entrenamiento se va almacenando los pesos del modelo que mejor resultado ha obtenido, con la finalidad de ser reutilizado en un segundo o tercer entrenamiento del modelo, partiendo con dichos pesos iniciales. La inicialización del modelo se realiza por defecto en forma aleatoria por la API Keras, sin embargo es posible inicializar el modelo para un nuevo entrenamiento y observar el impacto en los resultados de la clasificación.

El objetivo de esta prueba es verificar si inicializando el modelo planteado con los mejores resultados de las pruebas anteriores se mejora el desempeño en la tarea de clasificación.

5.5.1. Pesos inicializados con el mejor modelo previamente entrenado

Esta prueba consiste en inicializar los pesos del modelo que obtuvo el mejor valor de precisión y que fueron almacenados en el entrenamiento seleccionado. El objetivo de esta prueba es determinar si se obtiene un mejor resultado en la precisión de los datos de validación en el modelo durante el proceso de entrenamiento a partir de inicializar los pesos de las capas en el modelo propuesto. Al igual que en entrenamiento anterior se guardarán los pesos que indiquen el mejor valor de precisión. En la Figura 5.5(a) se muestra la tendencia de la precisión a través de las épocas, en donde se observa que la tendencia de precisión en ambos conjuntos de datos no están alineados, de donde en la época 122 se tuvo un pico con valor de precisión de 94.75 superando así la precisión de los pesos de inicialización. Los resultados de los datos de validación, durante el proceso de entrenamiento, muestran un comportamiento general con tendencia en los mismos rangos, excepto por 3 o 4 picos máximos. La Figura 5.5(b) muestra la tendencia de la función de pérdidas a través de las épocas, se observa que el comportamiento de las pérdidas del conjunto de validación no decrementa significativamente y no presenta un comportamiento similar al obtenido por los datos de entrenamiento, por lo tanto se espera que el rendimiento del modelo sea menor al esperado durante el entrenamiento.

Resultado : En esta prueba se puede observar que el mejor resultado guardado es de 94.75 % como se nota en la Figura 5.5 oscilando entre 87 % y 95 % la precisión, indicando que la mejora obtenida es uno de los picos obtenidos; el tiempo utilizado es en total de 19 segundos por época; en conclusión no es necesario realizar el entrenamiento nuevamente del modelo con la inicialización de los mejores pesos almacenados previamente, debido a que no garantiza en su mayoría obtener mejores resultados durante el segundo entrenamiento (200 épocas) y se puede establecer que los mejores resultados se obtienen durante las primeras 200 épocas de entrenamiento.



(a) ***Precisión*** **(b)** ***Pérdida***

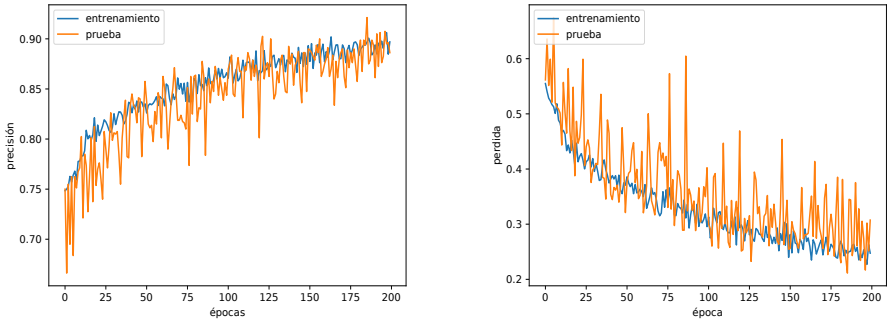
Figura 5.5: Comportamiento del resultado de precisión y pérdida con pesos iniciales

5.6. Diferente cantidad de imágenes en las clases para el entrenamiento

El objetivo de esta prueba es no mantener las mismas imágenes en la etapa de entrenamiento, debido a que en pruebas anteriores se fijo a 1000 imágenes, permitiendo tomar en forma aleatoria diferentes imágenes iniciales en el proceso de aprendizaje; con esta prueba se corrobora que para diferentes conjuntos de datos de entrenamiento el resultado de precisión se mantiene cercano al obtenido por mantener el conjunto de imágenes fijo. En esta prueba se propone considerar la cantidad total de imágenes obtenidas menos 200 que serán usadas para validación, y del total resultante seleccionar en forma aleatoria 1000. En la Tabla 5.5 se indican las cantidades usadas para cada clase en el conjunto de datos de entrenamiento.

Clase	Imágenes de entrenamiento
Escalera	3291
Mesas	1086
Puertas	1550
Sillas	1696

Tabla 5.5: Cantidad de imágenes del conjunto de entrenamiento



(a) ***Precisión*** **(b)** ***Pérdida***

Figura 5.6: Comportamiento del resultado de precisión y pérdida con un peso inicial de 92 %

Resultado: la mejor precisión alcanzada por este método fue de 0.92125, cada época tomó un total de 18 segundos en ejecutarse, siendo un poco inferior a la prueba de 200 épocas y 3 convoluciones en la cual se usaron 1000 imágenes de entrenamiento fijas y 200 de prueba. Como se muestra en la Figura 5.6 no se ha logrado superar la prueba pero se ha alcanzado un resultado similar por lo tanto se concluye que no afecta para el modelo seleccionar en forma aleatoria las imágenes y se observa un comportamiento alineado tanto en la precisión como en las pérdidas entre los datos de validación y entrenamiento.

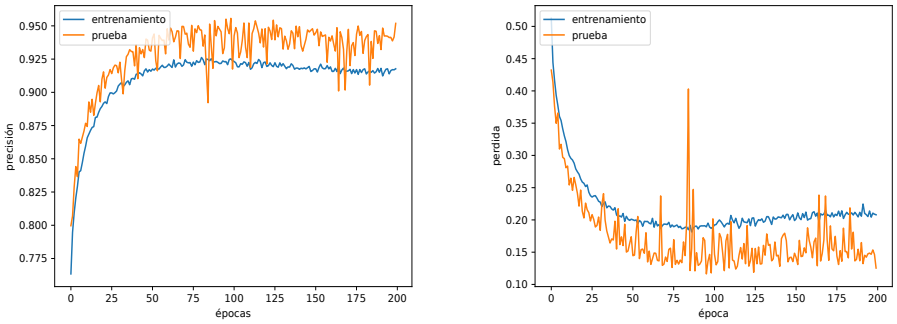
5.7. Prueba 80-20 del total de imágenes

El objetivo de esta prueba es verificar la respuesta del modelo con respecto al uso del total de las imágenes con que se cuenta para el entrenamiento, particionando en un porcentaje los datos, el porcentaje seleccionado es 80 % para entrenamiento y de 20 % para validación. En esta prueba se utilizaron los mejores parámetros obtenidos durante las pruebas anteriores, es decir, 200 épocas, una entrada de la imagen inicial de 120x120 px y por los resultados obtenidos se usaran 3 y 4 capas de convolución en el modelo. La Tabla 5.6 indica la cantidad total de imágenes con que se cuenta para cada clase y las cantidades en que se dividieron los datos manteniendo una relación 80-20.

Resultado: como se muestra en las gráficas de las Figuras 5.7 y 5.8 en las épocas 99

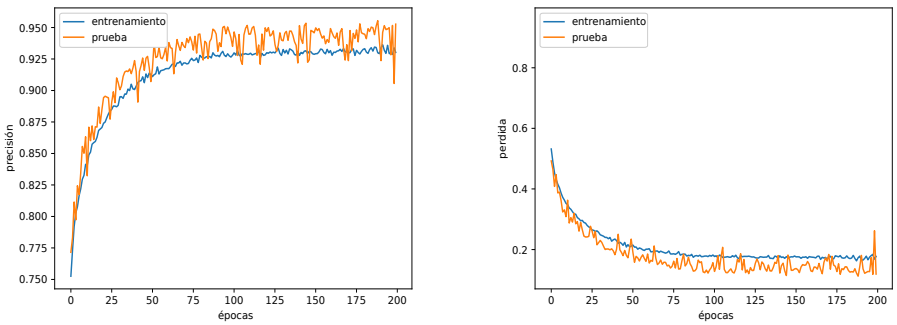
Clase	Total de imágenes	80 %	20 %
Escaleras	3491	2793	698
Mesas	1286	1029	257
Puertas	1750	1400	350
Sillas	1896	1517	379

Tabla 5.6: Cantidad de imágenes del conjunto de entrenamiento en la relación 80-20.



(a) *Precisión* **(b) *Pérdida***

Figura 5.7: Comportamiento de precisión y pérdida con una relación 80-20 y 3 capas de convolución



(a) *Precisión* **(b) *Pérdida***

Figura 5.8: Comportamiento de precisión y pérdida con una relación 80-20 y 4 capas de convolución

y 189 se obtiene el mejor valor de precisión para los modelos de 3 y 4 capas de convolución que es de 95.542 % y 95.563 respectivamente, los cuales se encuentran bastante cercanos. El tiempo por época en promedio es de 260 segundos, esto nos ayuda a entender que mientras más datos de entrenamiento se tengan, aumenta más el tiempo en ejecutar una época y aumenta más la precisión del modelo. Respecto al comportamiento de las gráficas de las Figuras 5.7(a) y 5.8(a), se observa que en la primera existe un sobre entrenamiento de los datos, al presentar una tendencia de los valores de precisión decreciente en aproximadamente las últimas 120 épocas del proceso de entrenamiento; mientras que la función de pérdidas en la Figura 5.7(b) presenta un comportamiento incremental y los datos de entrenamiento y prueba no se encuentran alineados o en paralelo para garantizar un comportamiento del modelo similar al del entrenamiento. En la Figura 5.8(a) se observa un mejor comportamiento de los resultados obtenidos, al mostrar una tendencia ascendente en las 200 épocas que se fijaron para el entrenamiento del modelo; se observa además que se encuentran alineados los resultados tanto de los datos de prueba y validación en ambas gráficas de la prueba. La Figura 5.8(b) que representa el comportamiento de la función de pérdidas indica un comportamiento similar a los datos de prueba y validación, con una tendencia a decrecer, algo muy deseable en los sistemas de aprendizaje profundo, este comportamiento garantiza un desempeño del modelo ante objetos a clasificar nunca vistos, es decir, el desempeño será similar al presentado durante la etapa de entrenamiento. Con este resultado se deberá elegir la arquitectura con 4 capas de convolución para algún sistema que se pretenda implementar y en donde se aplique la clasificación de los objetos o clases planteadas en el presente trabajo.

5.8. Prueba del modelo obtenido con otra fuente de datos

El objetivo de la siguiente prueba es probar los mejores modelos obtenidos en la tarea de clasificación de imágenes con los objetos a identificar en tiempo real con una cámara web o en imágenes individuales obtenidas de otra fuente de datos y que no se han utilizado para obtener alguno de los modelos. Las nuevas imágenes fueron descargadas de los repositorios de Google, al realizar búsquedas para las clases ya establecidas, obteniendo

25 imágenes para cada clase. La prueba se realiza mediante el programa de predicción en imagen que se encuentra en el Apéndice B. En las Tablas 5.8 y 5.8 se muestran los resultados de predicción de las clases y la cantidad de imágenes evaluadas, utilizando los modelos de 3 y 4 capas de convolución reportados en la prueba 5.7 cuya precisión durante el proceso de entrenamiento alcanzaron eficiencias del 95.542 % y 95.563 respectivamente sobre los datos de validación.

Clase	Cantidad imágenes	Predicción correcta	Precisión (%)
escaleras	25	22	88
mesas	25	14	56
puertas	25	22	88
sillas	25	22	88

Tabla 5.7: Resultados empleando el modelo con 3 capas de convolución.

Clase	Cantidad imágenes	Predicción correcta	Precisión (%)
escaleras	25	23	92
mesas	25	12	48
puertas	25	25	100
sillas	25	21	84

Tabla 5.8: Resultados empleando el modelo con 4 capas de convolución.

De la Tabla 5.8 se observa que la clase puertas es la que mejor se clasifica, teniendo un 100 % en la predicción; en segundo termino se tiene la clase escaleras con un 92 %. La clase más difícil de clasificar es la de mesas, en las predicciones obtenidas en su mayoría fue por la clase sillas, que al momento de observar las imágenes utilizadas en la prueba, los objetos encontrados en la escena son ambos.

En esta prueba se corroboran los resultados de los modelos entrenados respecto a la apreciación de las gráficas que representan el comportamiento de la eficiencia y función de pérdidas. El mejor desempeño es para el modelo con 4 capas de convolución. El resultado de la clase mesa aún para un humano es difícil asignar la clase correcta por el tipo de imágenes en las que se encuentran dos o más objetos de una clase.

En la Tabla 5.9 se muestra la predicción realizada por los modelos de las imágenes mostradas en la Figura 5.9, en el cual se hace notar que si en la escena se encuentra más

(a) ***Puerta y escalera*** **(b)** ***Mesa y sillas***

(c) ***Sillas y mesa*** **(d)** ***Escalera y puerta***

Figura 5.9: Ejemplo de imágenes con diferentes objetos en la escena

de uno de los objetos clasificados puede responder diferente a lo esperado.

Imagen	**Etiqueta asignada**	**Predicción del Modelo**	
		3 capas conv.	**4 capas conv.**
(a)	Puerta	Escalera	Puerta
(b)	Mesa	Mesa	Silla
(c)	Mesa	Mesa	Silla
(d)	Escalera	Puerta	Escalera

Tabla 5.9: Predicción de las imágenes de la Figura 5.9 por los modelos de aprendizaje profundo.

En esta ultima prueba se pudo observar que el modelo propuesto da un resultado aceptable de precisión, en la prueba con cada uno de los modelos, las clases mesas y sillas tienen el mayor problema de identificación, se observa que las imágenes utilizadas en todas las etapas (entrenamiento, validación, y prueba) contienen en muchas ocasiones ambos objetos (mesas y sillas), es por ello que la predicción en mesas y sillas son valores inferiores a los esperados por modelos entrenados. Ver la Figura 5.9 que muestra ejemplos de imágenes utilizadas uno o más objetos que fueron clasificados.

Capítulo 6

Conclusiones y Trabajos Futuros

6.1. Conclusiones

Como conclusiones de la presente Tesis se señalan las siguientes:

- Se ha presentado una arquitectura de red neuronal profunda, basada en CNN, propuesta para la clasificación de imágenes de cuatro clases (escaleras, mesas, puertas y sillas), logrando un desempeño de 95.5 % de precisión en los datos de validación.
- Se han descrito las principales operaciones en las CNN's y algunos otros elementos que son parte esencial del funcionamiento de las mismas y que nos permiten entender la extracción automática de características que se obtienen.
- Se concluye que no es de impacto el hecho de tener cantidades diferentes de imágenes en las clases, y que mientras más sean estas imágenes mejor sera entrenado el modelo.
- No impacta en los resultados de precisión, realizar un entrenamiento cuando ya se superó las 200 épocas inicializando los pesos del modelo con los mejores obtenidos en el entrenamiento previo para el modelo propuesto.
- El software libre ha proporcionado una forma de modelar arquitecturas de redes neuronales de una manera muy sencilla, gracias a lo cual se permite desarrollar y probar sistemas que integran varias tecnologías de una manera rápida. La forma modular de

representar las capas de los modelos de redes neuronales profundas permite concentrarse en el diseño de las arquitecturas.

- El tiempo utilizado para el entrenamiento del modelo se vio afectado por la cantidad de imágenes en los conjuntos de datos, mientras que para una mayor cantidad de capas de convolución el tiempo es afectado en una gran forma.
- El uso de imágenes incorrectas para entrenar el modelo puede generar un sobre-entrenamiento o un sub-entrenamiento, lo cual ocasiona que se identifique mal los objetos, recomendando que se debe tener mucho cuidado con la creación del conjunto de imágenes que se emplean para el entrenamiento.
- Se presenta evidencia empírica para modelar una arquitectura de aprendizaje profundo, realizando pruebas solamente para ciertos parámetros de la arquitectura, mostrando los comportamientos que nos permitan elegir o decidir el mejor modelo.

6.2. Trabajos futuros

En trabajos futuros se plantean los siguientes:

- Crear una nueva clase que identifique imágenes que contienen mesas y sillas, depurando los conjuntos de datos de entrenamiento y validación.
- Conjuntar el clasificador en un sistema de Visión Computacional que nos lleve a construir sistemas más complejos y eficientes, dando la oportunidad de explorar otras tareas como el conteo e identificación de objetos en una escena.
- Identificar en la imagen analizada cuantos objetos hay de cada clase.
- Identificar en la imagen analizada donde se encuentran los objetos de cada clase.
- Localizar la sección donde se encontró la clase y segmentarla.
- Usar el modelo generado para ser entrenado e identificar los sentimientos de una persona atreves de su lenguaje corporal.

Apéndice A

Instalación de bibliotecas y programas

A continuación se presenta como se realizó la instalación para el uso y ejecución de las librerías y programas usados durante el presente trabajo. Primero es necesario cumplir con los requerimientos básicos de instalación, como se indica en la sección A.1 de este apéndice. Posteriormente se señala que la instalación se puede realizar de dos formas, la primera es instalando uno por uno las librerías y programas necesarios, la segunda es usando la instalación por archivo para el programa `pip`.

La instalación se llevó a cabo sobre una computadora con las siguientes características: Sistema operativo: Ubuntu 18.04, Tarjeta gráfica: Nvidia Gforce Gtx 980, Procesador: i5 4440, Memoria Ram: 16Gb.

A.1. Instalación Básica

En la instalación básica, se consideran los programas base, que son necesarios para efectuar alguna de las instalaciones de las bibliotecas utilizadas, tales como: keras, tensorflow y opencv principalmente. La lista y forma de ejecución en la línea de comandos de Linux de los programas básicos es la siguiente:

- Tkinter

```
# apt install python3-tk
```

- Virtualenv

```
# apt install python-virtualenv
```

- Python 3 en un entorno virtual

```
$ virtualenv -p python3 env
```

- Fdupes

```
# apt install fdupes
```

Nota: El prompt que se señala con el símbolo "#" representa que se cuenta con los privilegios de superusuario (`root`), mientras que el prompt que se señala con el símbolo "$" representa que lo puede ejecutar cualquier usuario en el sistema operativo.

A.2. Instalación Manual

Una de las formas de instalar el conjunto de bibliotecas que nos ayudan a implementar la integración del aprendizaje profundo en una aplicación de visión computacional, se describe a continuación. El primer paso es activar el entorno virtual, el cual se realiza con la siguiente instrucción:

```
$ source env/bin/activate
```

Una vez en el prompt del entorno virtual, señalado por `(env)$` se procede a ejecutar la siguiente secuencia de instrucciones para lograr la instalación de librerías.

- Tensorflow

```
(env)$ pip install tensorflow
```

- Tensorflow para GPU (solo tarjetas gráficas Nvidia compatibles ver A.4)

```
(env)$ pip install tensorflow-gpu
```

- Keras

```
(env)$ pip install keras
```

- Pillow

```
(env)$ pip install pillow
```

- Flickr

```
(env)$ pip install flickrapi
```

- OpenCV

```
(env)$ pip install opencv-python
```

- Matplotlib

```
(env)$ pip install matplotlib
```

A.3. Instalación por archivo

La segunda forma de instalación de librerías se refiere a escribir en un archivo de texto plano los nombres de dichas librerías, este archivo se obtuvo por medio del comando `(env)$ pip freeze >requeriments.txt` después de la instalación manual. Una vez que se ha activado el entorno virtual se procede a realizar la instalación con el archivo, que de preferencia tiene por nombre `requirements.txt`. El contenido del archivo deberá ser el siguiente (en caso de no contar con un tarjeta gráfica Nvidia cambiar tensorflow-gpu por tensorflow):

```
absl-py==0.7.1
astor==0.7.1
backcall==0.1.0
certifi==2019.3.9
chardet==3.0.4
cycler==0.10.0
decorator==4.4.0
```

```
flickrapi==2.4.0
gast==0.2.2
google-pasta==0.1.5
grpcio==1.20.0
h5py==2.9.0
idna==2.8
ipython==7.4.0
ipython-genutils==0.2.0
jedi==0.13.3
Keras==2.2.4
Keras-Applications==1.0.7
Keras-Preprocessing==1.0.9
kiwisolver==1.0.1
Markdown==3.1
matplotlib==3.0.3
mock==2.0.0
numpy==1.16.2
oauthlib==3.0.1
opencv-python==4.1.0.25
parso==0.4.0
pbr==5.1.3
pexpect==4.7.0
pickleshare==0.7.5
Pillow==6.0.0
pkg-resources==0.0.0
prompt-toolkit==2.0.9
protobuf==3.7.1
ptyprocess==0.6.0
Pygments==2.3.1
pyparsing==2.4.0
python-dateutil==2.8.0
PyYAML==5.1
requests==2.21.0
```

```
requests-oauthlib==1.2.0
requests-toolbelt==0.9.1
scipy==1.2.1
six==1.12.0
tb-nightly==1.14.0a20190301
tensorboard==1.13.1
tensorflow-estimator==1.13.0
tensorflow-gpu==1.13.1
termcolor==1.1.0
tf-estimator-nightly==1.14.0.dev2019030115
traitlets==4.3.2
urllib3==1.24.2
wcwidth==0.1.7
Werkzeug==0.15.2
```

La instalación se realizará de la siguiente forma en el prompt:

```
(env)$ pip install -r requeriments.txt
```

A.4. Instalación para uso de GPU's Nvidia

Tensorflow-gpu es una librería alternativa a la librería Tensorflow con la cual será posible utilizar una tarjeta gráfica que nos permita el manejo de los GPU's para lograr mayor capacidad de procesamiento o calculo matemático. El fabricante Nvidia proporciona librerías para utilizar sus GPU's en el campo de la inteligencia artificial. Activar esta característica requiere descargar los drivers oficiales en la siguiente url: `https://www.nvidia.com/Download/index.aspx?lang=en-us` e instalarlos de la siguiente forma:

```
sh NVIDIA-Linux-x86_64-418.56.run
```

Una vez instalado el driver Nvidia se procede a descargar e instalar la librería NVIDIA CUDA Deep Neural Network library (cuDNN) de la url: (`https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html`), en la que se señala proceder de la siguiente forma:

Descomprimir el siguiente archivo:

```
$ tar -xzvf cuda-repo-ubuntu1804-10-0-local-10.0.130-410.48_1.0-1_amd64.deb
```

Copiar los archivos a las rutas indicadas:

```
# cp cuda/include/cudnn.h /usr/local/cuda/include/
# cp cuda/lib64/libcudnn* /usr/local/cuda/lib64/
# chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```

Instalar las librerías con los siguientes comandos en el prompt:

```
# dpkg -i libcudnn7\_7.5.0.56-1+cuda10.1\_amd64.deb
# dpkg -i libcudnn7-dev\_7.5.0.56-1+cuda10.1\_amd64.deb
# dpkg -i libcudnn7-doc\_7.5.0.56-1+cuda10.1\_amd64.deb
```

Finalmente instalar la librería CUDA (Para el momento de realizar este documento la librería 10.0 es la más compatible con tensorflow-gpu):

```
# dpkg -i cuda-repo-ubuntu1804-10-0-local-10.0.130-410.48\_1.0-1\_amd64.deb
# apt-key add /var/cuda-repo-<version>/7fa2af80.pub
# apt update
# apt install cuda
```

Apéndice B

Códigos Fuente

B.1. Código fuente para obtención masivo de imágenes

Código fuente en lenguaje de programación python, que permite a través de flickrapi la obtención masiva de imágenes

```
import flickrapi
import urllib
from PIL import Image
import sys

flickr=flickrapi.FlickrAPI('c6a2c45591d4973ff525042472446ca2',
                           '202ffe6f387ce29b', cache=True)

keyword = sys.argv[1]

photos = flickr.walk(
  text=keyword,
  tag_mode='any',
  tags=keyword,
  extras='url_c',
  per_page=500,
  sort='relevance',
```

```
  accuracy=1
)

try:
  for i, photo in enumerate(photos):
    if photo.get('url_c') is not None:
      url = photo.get('url_c')
      try:
        urllib.request.urlretrieve(url,
                                   "{nombre}{numero}.jpg".format(nombre=sys.argv[2],
                                                                 numero=str(i).zfill(5))
                                   )
      except Exception as e:
        print(e)
    if i > 100000:
      break
except Exception as e:
  print(e)
```

B.2. Comando para borrado de imágenes duplicadas

Busca la ruta dada para archivos duplicados. Dichos archivos se comparan por tamaños de archivo y firmas MD5, seguidos de una comparación byte por byte.

```
fdupes -d -N ruta/
```

B.3. Código fuente para entrenamiento del modelo con porcentaje 80-20

El siguiente código consta de la generación del modelo con 4 convoluciones, con sus respectivos maxpooling, usando la función de activación relu, los datos de entrenamiento y validación son aplicados gracias a la función ImageDataGenerator, guardando con ello el modelo, y los mejores pesos encontrados durante las épocas generadas.

```
'''Esta es la estructura de los directorios:
data
|---- train
      |---- escaleras
      |---- mesas
      |---- puertas
      |---- sillas
|---- validation
      |---- escaleras
      |---- mesas
      |---- puertas
      |---- sillas
'''

from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
import sys

filepath = None

#dimensiones de la imagen de entrada
img_width, img_height = 120, 120

train_data_dir = 'data/train'
validation_data_dir = 'data/validation'
nb_train_samples = 1200
nb_validation_samples = 200
epochs = 200
```

```
batch_size = 100 # tamanio divisible entre el total de ejemplos

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(4))
model.add(Activation('softmax'))

if len(sys.argv) == 2:
  model.load_weights(sys.argv[1])
```

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

#este sera la configuracion de aumento para el entrenamiento
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

#este sera la configuracion de aumento para la validacion
#test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)

validation_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')


filepath="weights-improvement-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
```

```
                                save_best_only=True, mode='max')
callbacks_list = [checkpoint]

history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    callbacks=callbacks_list,
)

# Graficar el historial de aprendizaje
#print(history.history.keys())
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('exactitud del modelo')
plt.ylabel('exactitud')
plt.xlabel('epocas')
plt.legend(['entrenamiento', 'prueba'], loc='upper left')
plt.savefig('accuracy.pdf',bbox_incles='tight')
plt.close()
#graficar historial de perdida
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('perdida del modelo')
plt.ylabel('perdida')
plt.xlabel('epoca')
plt.legend(['entrenamiento', 'prueba'], loc='upper left')
plt.savefig('loss.pdf',bbox_incles='tight')

model.save('modelo.hdf5')
model.save_weights('pesos.h5')
```

B.4. Código fuente para búsqueda de objetos dentro de una imagen

El siguiente código consta de la lectura del modelo generado, para tomar imágenes y determinar que clase contiene cada una de ellas.

```
import sys, cv2, numpy as np
from keras.preprocessing.image import img_to_array, load_img
from keras.models import load_model
from keras.applications.imagenet_utils import preprocess_input, decode_predictio
from keras.preprocessing import image
import sys

def identifica(lista):
  encontro = []
  clases = ['escaleras','mesas','puertas','sillas']
  maxi = np.amax(lista)
  print(maxi)
  indice = np.where(lista == maxi)
  print(indice[0][0])

  return clases[indice[0][0]]

if len(sys.argv) == 3:
  loaded_model=load_model(sys.argv[1])
else:
  print("Favor de indcar la ruta del modelo o imagen a usar ...")
  sys.exit(0)

width, height = 120, 120

img = cv2.imread(sys.argv[2],1)
array = img_to_array(img)
arrayresized = cv2.resize(array, (width,height))
```

```
arrayresized=np.multiply(arrayresized,[1.0/255])

inputarray = arrayresized[np.newaxis,...] # dimension added to fit input size
preds = loaded_model.predict(inputarray)

for item in preds:
  resp = identifica(item)
  print("la imagen contiene:",str(resp))
```

Referencias

[Alpaydin09] Alpaydin, E. *Introduction to machine learning.* MIT press, 2009.

[Bradski08] Bradski, G. y Kaehler, A. *Learning OpenCV: Computer vision with the OpenCV library.* Reilly Media, Inc.", 2008.

[Butterfield18] Butterfield, S. y Fake, C. Flickr. `https://www.flickr.com/`, 2018. Accessed: 2019-06-28.

[Chollet15] Chollet, F. et al. Keras. `https://keras.io`, 2015.

[Chollet18] Chollet, F. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek.* MITP-Verlags GmbH & Co. KG, 2018.

[Cireşan12] Cireşan, D., Meier, U., y Schmidhuber, J. Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*, 2012.

[Gupta17] Gupta, D. Activation functions and when to use them? `https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/`, 2017. Accessed: 2019-07-05.

[Gurney14] Gurney, K. *An introduction to neural networks.* CRC press, 2014.

[Hertz18] Hertz, J. A. *Introduction to the theory of neural computation.* CRC Press, 2018.

[Huval15] Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R., et al. An

empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[Jarrett09] Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. What is the best multi-stage architecture for object recognition? *En 2009 IEEE 12th international conference on computer vision*, págs. 2146–2153. IEEE, 2009.

[Kirk07] Kirk, D. et al. Nvidia cuda software and gpu parallel computing architecture. *En ISMM*, tomo 7, págs. 103–104. 2007.

[LeCun15] LeCun, Y., Bengio, Y., y Hinton, G. Deep learning. *nature*, 521(7553):436, 2015.

[mat19] Convolutional neural network. `https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html`, 2019. Accessed: 2019-08-08.

[McCulloch43] McCulloch, W. S. y Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[Moroshko18] Moroshko, M. Cnn for visual recognition. `http://cs231n.github.io/convolutional-networks/`, 2018. Accessed: 2019-08-10.

[Munárriz94] Munárriz, L. Á. *Fundamentos de inteligencia artificial*, tomo 1. Editum, 1994.

[Nielsen15] Nielsen, M. A. *Neural networks and deep learning*, tomo 25. Determination press San Francisco, CA, USA:, 2015.

[Nilsson14] Nilsson, N. J. *Principles of artificial intelligence*. Morgan Kaufmann, 2014.

[Patterson17] Patterson, J. y Gibson, A. *Deep learning: A practitioner's approach*. Reilly Media, Inc.", 2017.

[Rojas13] Rojas, R. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[Russell04] Russell, S. J. y Norvig, P. *Inteligencia Artificial: un enfoque moderno*. 04; Q335, R8y 2004. 2004.

[Salcedo18] Salcedo, L. Pasos para construir un modelo de machine learning. `http://www.pythondiario.com/2018/01/introduccion-al-machine-learning-6.html`, 2018. Accessed: 2019-06-28.

[und18] Understanding of convolutional neural network (cnn). `https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760` 2018. Accessed: 2019-08-10.

[Wang17] Wang, H. y Raj, B. On the origin of deep learning. *arXiv preprint arXiv:1702.07800*, 2017.