UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICA
MAESTRÍA EN MODELOS ALEATORIOS

ESTUDIO DE LOS MÉTODOS DE APRENDIZAJE ESTADÍSTICO PROVISTOS POR AMAZON WEB SERVICES E IMPLEMENTACIÓN DE REDES NEURONALES CONVOLUCIONALES PARA EL RECONOCIMIENTO DE PATRONES VISUALES CON AMAZON DEEPLENS

Trabajo de Grado de Maestría presentado ante la ilustre Universidad Central de Venezuela por el **Ing. Alexander A. Ramírez M.** para optar al título de **Magister Scientiarum Mención Modelos Aleatorios**

Tutor: Dr. Ricardo Ríos.

Caracas, Venezuela

Octubre - 2019

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
COMISIÓN DE ESTUDIOS DE POSTGRADO

Comisión de Estudios de Postgrado

VEREDICTO

Quienes suscriben, miembros del jurado designado por el Consejo de la Facultad de Ciencias de la Universidad Central de Venezuela, para examinar el **Trabajo de Grado** presentado por: **Alexander A. Ramírez Morales,Cédula de identidad** 10.488.134, bajo el título "**ESTUDIO DE LOS MÉTODOS DE APRENDIZAJE ESTADÍSTICO PROVISTOS POR AMAZON WEB SERVICES E IMPLEMENTACIÓN DE REDES NEURONALES CONVOLUCIONALES PARA EL RECONOCIMIENTO DE PATRONES VISUALES CON AMAZON DEEPLENS**", a fin de cumplir con el requisito legal para optar al grado académico de **MAGÍSTER SCIENTIARUM, MENCIÓN MODELOS ALEATORIOS**, dejan constancia de lo siguiente:

1.- Leído como fue dicho trabajo por cada uno de los miembros del jurado, se fijó el día 11 de octubre de 2019 a las 11:30 am, para que el autor lo defendiera en forma pública, lo que éste hizo en la **Sala de Seminario de Matemática**, mediante un resumen oral de su contenido, luego de lo cual respondió satisfactoriamente a las preguntas que le fueron formuladas por el jurado, todo ello conforme con lo dispuesto en el Reglamento de Estudios de Postgrado.

2.- Finalizada la defensa del trabajo, el jurado decidió **APROBARLO con la calificación de EXCELENTE** por considerar, sin hacerse solidario con las ideas expuestas por el autor, que se ajusta a lo dispuesto y exigido en el Reglamento de Estudios de Postgrado.

Para dar este veredicto, el jurado estimó que el trabajo proporciona información sobre el uso de redes neuronales convolucionales para el reconocimiento de patrones visuales en imágenes, usando las plataformas de Amazon destinadas al uso intensivo de la inteligencia artificial y su implementación computacional apta para el uso extensivo de usuarios de formación e interés multidisciplinario.

En fe de lo cual se levanta la presente ACTA, a los 11 días del mes de octubre del año 2019, conforme a lo dispuesto en el Reglamento de Estudios de Postgrado, actuó como Coordinador del jurado **Ricardo Ríos.**

Mairene Colina / C.I.12.761.954
Institución (UCV)

Elvia Flores / C.I. 4.128.598
Institución (UCV)

Ricardo Ríos / C.I. 3.949.476
Institución (UCV)
Tutor

Resumen

En este trabajo se estudiarán los servicios de Inteligencia Artificial y Aprendizaje estadísticos provistos por Amazon Web Services y se muestra una metodología de trabajo para aplicar dichos métodos utilizando grandes cantidades de datos.

En particular se estudiarán y se entrenará una Red Neuronal Convolucional para identificar patrones visuales y se implementará en el dispositivo Amazon Deeplens.

Palabras clave: Aprendizaje estadístico, Aprendizaje Profundo, Redes Neuronales, Redes Neuronales Convolucionales, Amazon, AWS, Deeplens.

Dedicatoria

A mi esposa Zulay Duarte por estar ahí en todo momento y apoyarme en todo lo que me propongo, por la paciencia y por la comprensión. Soy afortunado de que seas mi compañera de vida.

A mis hijos que me inspiran a ser mejor y por darme tantas razones todos los días de sentirme orgulloso. Para alcanzar algo sólo deben quererlo realmente.

A mis hermanos, mis compañeros de toda la vida. Los que realmente me conocen y me quieren como soy. A mis sobrinos que me llenan de cariño. Sigan sus sueños.

A mis amigos, los hermanos que la vida me dió. A los que están cerca y a los que no.

En especial a mi mamá Estrella Morales, has sembrado en mí tantas cosas que hoy sin darme cuenta están ahí gracias a tí. Soy el resultado de todos tus sacrificios.

Agradecimiento

A Dios, por ser la fuerza que guía lo que haga todo los días.

A la Facultad de Ciencias de la UCV y en especial a la Escuela de Matemática por contar con profesores que a pesar de las circunstancias siguen incólumes en mantenerla funcionando haciendo grandes sacrificios.

A mi tutor, el Doctor Ricardo Ríos, por todo su apoyo, por compartir tantos conocimientos y guíar la elaboración de este trabajo.

Al equipo de Synergy Vision que ha tenido la paciencia y me ha permitido seguir creciendo profesionalmente.

A mis socios José Ricardo Rivera y Alfredo Pereyra por su comprensión y paciencia en todas mis aventuras.

Índice general

Índice de figuras

Índice de tablas

Introducción

El trabajo de grado consiste en la investigación documental y práctica del estado del arte en la aplicación de algoritmos de Aprendizaje estadístico provistos por Amazon para desarrollar una práctica que permita implementar soluciones a problemas que plantea la industria, así como conocer los límites de dichos servicios y plantearnos nuevas líneas de investigación que complementen la oferta de servicios actual.

Este trabajo pretende estudiar los modelos ofrecidos a través de las herramientas provistas por Amazon, así como desarrollar metodologías y prácticas para implementar servicios a empresas que aplican dichos modelos y algoritmos de aprendizaje estadístico, a través de los servicios e infraestructura en la nube provistos por Amazon.

La aplicación de métodos para reconocer patrones, inferir y clasificar partiendo de los datos, se están popularizando en todas las industrías. En este sentido las grandes empresas proveedoras de tecnología están ofreciendo opciones para implementar algunos de estos métodos y algoritmos en la nube, accesibles a través de internet y sin necesidad de implementar capacidad de procesamiento y almacenamiento propios, mediante granjas de servidores.

Estos servicios ofrecidos en la nube permiten, a las empresas proveedoras de soluciones, implementar aplicaciones que reciben los datos, los preprocesan, los consumen, generan modelos y se ajustan, para luego predicir, reconocer patrones y clasificar. Estos servicios estandarizan la forma de aplicar modelos de aprendizaje basado en los datos y permiten su aplicación de forma inmediata.

Amazon es una empresa que inicia en el año 1997 como un portal de comercio electrónico, es una de las pocas que ha sobrevivido luego de la crisis de las empresas punto com entre 1999 y el año 2000. Lo curioso de esta empresa es que para sostener un negocio de comercio electrónico global desarrolló grandes motores de aprendizaje estadístico para ofrecer recomendaciones a sus usuarios de forma oportuna. Estas aplicaciones requieren gran capacidad de cómputo y almacenamiento, Amazon las desarrolla para su propio uso y

luego empieza a ofrecerlas como servicios a través de su plataforma Amazon Web Services (AWS).

Ahora es posible implementar un servidor o una base de datos sin necesidad de comprarlo y sin necesidad de implementar toda la infraestructura necesaria para instalar el servidor como: espacio acondicionado y climatizado, acceso a internet, gabinetes, electrícidad, UPS, entre otros. Amazon a través de AWS ofrece infraestructura que se paga en la medida de la capacidad que se consume, también conocido como Software como Servicio *Software as a Service - SaaS* y Plataforma como Servicio *Platform as a Service - PaaS*.

Este modelo de negocio permite a los usuarios la oferta de servicios de forma flexible y elástica, ya que la infraestructura en la nube crece de acuerdo a las necesidades y se adapta a la estacionalidad. Cuando se implementa la infraestructura de servidores propia, esta se debe dimensionar para soportar la mayor carga posible durante el año y es usual que el resto del año dicha capacidad esté ociosa, generando costos de inversión altos. Una de las ventajas de los servicios de AWS es que son elásticos y se pagan en la medida en que se utilizan.

Sobre esta plataforma de servicios en la nube se implementan muchos servicios útiles para los desarrolladores de aplicaciones, entre ellos los servicios de Aprendizaje Automático que consisten en la implementación de modelos y algoritmos de aprendizaje estadístico.

AWS - Aprendizaje Automático. Entre los servicios que ofrece AWS relacionado con las técnicas de aprendizaje estadístico se encuentran: Apache MXNet en AWS, Amazon Comprehend, AMI de aprendizaje profundo de AWS, AWS DeepLens, Amazon Lex, Amazon Machine Learning, Amazon Polly, Amazon Rekognition, Amazon SageMaker, Amazon Transcribe y Amazon Translate.

Otros proveedores como Google, Microsoft e IBM ofrecen sus propias soluciones.

Google.

- AI en la Nube
- Motor de Aprendizaje Automatizado en la Nube

Microsoft Azure.

- Azure AI
- AI + Aprendizaje Automatizado
- Servicios de Aprendizaje Automatizado

IBM Bluemix, Analítica y Watson.

- Bluemix Watson
- Ciencia de los Datos y Aprendizaje Automatizado
- Aprendizaje Automatizado con Watson

Adicionalmente a los proveedores de soluciones se encuentran las herramientas que implementan los modelos, entre ellas: Tensorflow, Keras, Caffee, Pytorch, Spark MLlib, scikit learn, xgboost, SPSS, R y la comunidad que aporta paquetes y librerías.

Debido a la gran diversidad de problemas, existe una variedad de opciones que bien valen la pena ser estudiadas y al mismo tiempo permiten desarrollar soluciones con los últimos avances logrados en la academia y en la industria.

En Junio del 2018, Amazon lanza "La primera cámara de vídeo del mundo con aprendizaje profundo para desarrolladores". Es el kit para desarrolladores con una cámara de alta definición denominado AWS DeepLens.

> AWS DeepLens es un conjunto de herramientas integradas (kit) para desarrolladores con cámara de alta definición conectada con un conjunto de proyectos de muestra para ayudar a los desarrolladores a aprender conceptos de aprendizaje automático mediante casos de uso de visión artificial prácticos. AWS DeepLens está preconfigurado para funcionar con varios servicios de AWS con el objetivo de ofrecer un marco de trabajo de aprendizaje profundo que esté optimizado para AWS DeepLens, por lo que es sencillo crear aplicaciones de visión artificial. Si bien AWS DeepLens tiene capacidad suficiente para los expertos, también está diseñado para ayudar a todos los desarrolladores a empezar a trabajar rápidamente con poca o nula experiencia en aprendizaje profundo.

En este sentido queremos explorar las capacidades ofrecidas por este kit con un enfoque doble: en primer lugar entender la oferta de AWS, en el entendido que ofrecen opciones de gran avance tecnológico que vale la pena estudiar y en segundo término, partiendo de la oferta de AWS implementar soluciones y sobre ellas realizar aportes y mejoras. En este sentido el desarrollo académico puede partir del avance que ya otros han logrado y sobre estos desarrollar nuevos modelos.

Capítulo 1

Antecedentes y Motivación

1. Antecedentes

Nuestra racionalidad es fascinante y enigmática y ha sido objeto de estudio por diversas disciplinas, tanto las médicas como las tecnológicas. Nuestra forma de razonar, el pensamiento, los procesos mentales y nuestros comportamientos generan interés desde el punto de vista de la tecnología y el gran objetivo es automatizar actividades imitando estos procesos. El objetivo, además de entender la inteligencia, es adicionalmente construir entidades inteligentes [1].

La Inteligencia Artificial como campo de estudio está compuesto por una gran variedad de disciplinas que van desde las generales como el aprendizaje y la percepción hasta las más específicas como jugar ajedrez o Go y manejar un carro de forma autónoma [1].

Dentro de las capacidades que deben manifestarse para mostrar inteligencia podemos contar con:

- Procesamiento del Lenguage Natural (NLP): Consiste en comunicarse en un idioma.
- Representación del Conocimiento: Almacenar lo que se conoce o se escucha.
- Razonamiento automatizado: Responder preguntas e inferir basado en el conocimiento adquirido.
- Aprendizaje automatizado: Adaptarse a nuevos datos, detectar patrones e inferir.
- Visión automatizada: Percibir objetos e identificarlos
- Robótica: Manipular objetos.

Estas capacidades surgen de la prueba definida por Alan Turing (1950) donde se realizan una serie de preguntas y basado en las respuestas, el que interroga no puede decidir si las respuestas provienen de una persona o una computadora.

De todas estas capacidades vamos a enfocarnos en el aprendizaje automatizado, también conocido como Machine Learning. Esta capacidad nos genera gran interés ya que a partir de ella, y sus métodos, se pueden desarrollar otras capacidades.

El aprendizaje automatizado consiste en una serie de modelos, técnicas y algoritmos que parten de los datos y sobre estos se identifican características y patrones que permiten inferir. El problema fundamental es encontrar el balance adecuado entre el sesgo y el error.

El estudio de los métodos de aprendizaje estadístico en la academia se centra en su entendimiento teórico y los aspectos técnicos mediante herramientas didácticas. Este trabajo se realiza con la idea de abonar en el terreno de la construcción de aplicaciones con datos reales a través de las plataformas y servicios de uno de los grandes proveedores de tecnología. La idea es que los métodos explorados, estudiados y aplicados queden documentados para que otros puedan utilizar en aplicaciones y soluciones para la industria, sobre la infraestructura de Amazon. Partiendo de estos modelos, su uso y aplicación, podemos entender los límites y al mismo tiempo las líneas de investigación que se pueden seguir para avanzar en este terreno.

En nuestra investigación documental hemos hallado cuatro grandes corrientes de desarrollo de la investigación sobre los métodos de aprendizaje estadístico, tanto supervisados como no supervisados, entre ellos:

- Métodos de aprendizaje estadístico supervisados basados en regresión, clasificación, máquinas de soporte vectorial y no supervisados como agrupamiento, componentes principales, ranking, bosques aleatorios, entre otros.
- Redes Neuronales
- Aprendizaje reforzado (Reinforcement Learning)
- Aprendizaje bayesiano

Este trabajo se enfocará en los dos primeros. En primer término en los métodos de aprendizaje estadístico y sus metodologías como introducción al segundo modelo que aunque inspirado en el cerebro humano, sus métodos siguen los patrones de ajuste de parámetros y optimización desarrollados en los primeros. Se prestará especial énfasis a las Redes Neuronales y sus métodos en detalle hasta el desarrollo de las Redes Neuronales Convolucionales.

1.1. Redes Neuronales. Las Redes Neuronales son un modelo computacional y matemático de las neuronas biológicas. Este modelo se inspira en el funcionamiento del cerebro humano y consiste en construir arquitecturas de neuronas conectadas que se entrenan con datos y resultados conocidos, para identificar patrones.

En [2] McCulloch y Pitts en 1943 inician el estudio de las Redes Neuronales donde presentan un modelo computacional de cómo las neuronas pueden trabajar en conjunto para realizar cálculos complejos utilizando lógica proposicional. Esta fué la primera arquitectura.

En 1957 Frank Rosemblatt en [3], desarrolla el perceptrón simple que consiste en una red de una sola neurona. Una neurona recibe el estímulo de otras neuronas que pueden activar o no a la neurona receptora que integra todos los estímulos o impulsos recibidos de otras neuronas. Si los impulsos integrados generan un valor mayor a cierto umbral entonces la neurona se activa.

El modelo original recibe una entrada binaria y la integración de las entradas se realiza mediante la suma ponderada de las entradas. Este modelo sirvió como referencia para resolver problemas de clasificación lineal. La principal limitación del modelo es su incapacidad para resolver problemas de regresión con datos que no son linealmente separables. Este modelo es la base de los modelos de Redes Neuronales actuales.

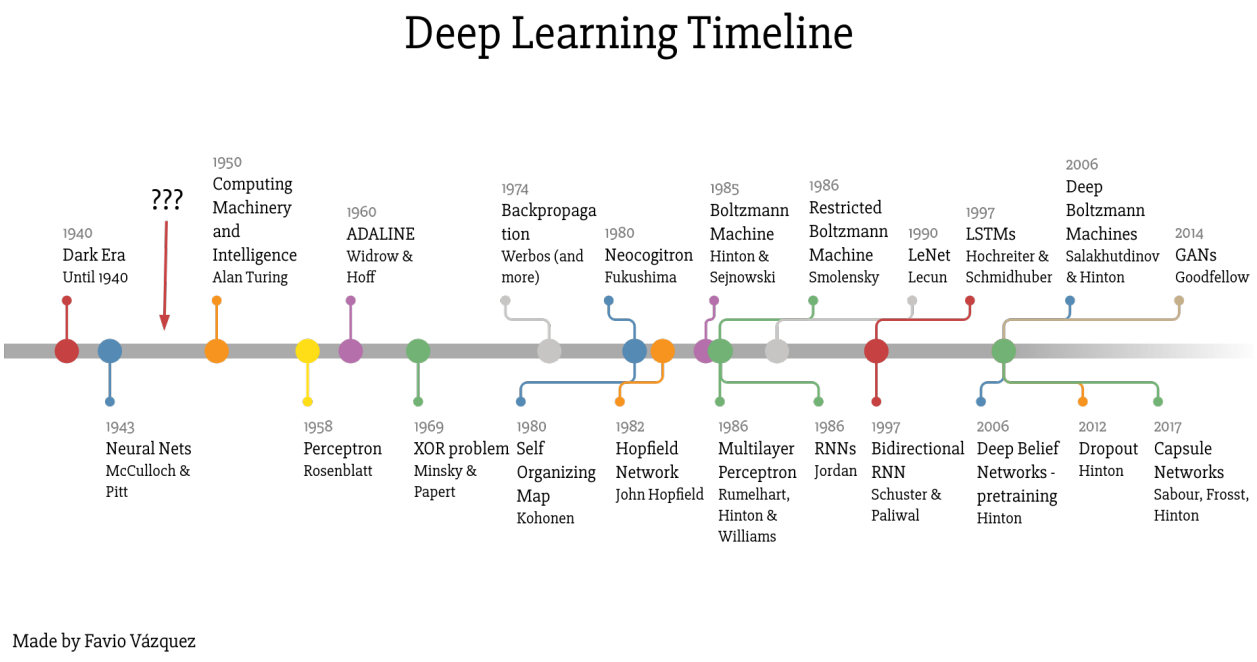


FIGURA 1.1. Progresión de avances en Redes Neuronales

El estudio de las Redes Neuronales no es nuevo, son muchos los aportes de diversos investigadores como Turing, Werbos, Fukushima, Hinton, LeCun, Bengio, Goodfellow entre

otros. En la imagen 1.1 se puede ver el progreso gradual que ha tenido el área durante los últimos 80 años.

El descubrimiento de nuevas arquitecturas y avances en los métodos y definición de las Redes Neuronales así como la creciente capacidad de cómputo han generado un grán avance en los últimos 7 años. Entre ellos la popularización de las Redes Neuronales Convolucionales.

1.2. Redes Neuronales Convolucionales. En [4] LeCun y Bengio indican que una pregunta esencial cuando se diseñan arquitecturas de aprendizaje es cómo representar la invarianza. Las propiedades de la invarianza son cruciales para cualquier método de aprendizaje, en particular en el reconocimiento de patrones visuales.

En el mismo artículo definen las Redes Neuronales Convolucionales como arquitecturas multicapa en las cuales las capas sucesivas están diseñadas para extraer, aprender progresivamente características más complejas, hasta alcanzar la última capa que representa las categorías. Todas las capas se entrenan simultáneamente con el objetivo de minimizar la función de pérdida. A diferencia de la mayoría de los modelos de clasificación y reconocimiento de patrones, la extracción de características y la clasificación ocurren en la misma red. Todas las capas son similares en naturaleza y se entrenan a partir de los datos de forma integrada.

Un módulo básico de la Red Convolucional se compone de una Capa de extracción de características o Convolución, seguido de una Capa de Sub muestraje. Una Red Convolucional típica está compuesta por uno, dos o tres de estos modulos integrados en serie y seguidos de un módulo de clasificación.

Las entradas y salidas de cada capa se pueden ver como matrices de dos dimensiones denominados mapas de características que se generan a partir de la aplicación de una operación de convolución que utiliza un campo de recepción denominado kernel. La definición y uso de los kernel se inspiran en las investigaciones de Hubel y Wiesel.

Hubel y Wiesel en [5] estudian las propiedades de las neuronas en la corteza visual de los gatos, tomando registros de las células en gatos ligeramente anestesiados. Las retinas de los gatos fueron estimuladas individualmente o simultáneamente con puntos de luz de varios tamaños y formas. El análisis de dichos registros permiten identificar células que se activan y responden a estímulos específicos como barras horizontales o verticales de forma excluyente. Cada neurona es responsable por una región pequeña del campo visual y se

activan a estímulos localizados en esas regiones. El campo visual de las neuronas se pueden solapar para componer todo el campo visual. También notaron que algunas neuronas tienen un campo visual mayor y se activan con estímulos más complejos que son la combinación de estímulos más simples.

Estas observaciones sugieren que las neuronas con un mayor campo visual reciben información de las neuronas de menor campo visual. Esta arquitectura es la que se ha utilizado como base para construir un modelo de Redes Convolucionales que es capaz de identificar patrones en imágenes. En el artículo [6] se presenta el trabajo de Yann LeCun, Léon Bottou, Yoshua Bengio y Patrick Haffner, que será fundamental para el desarrollo futuro de dichas redes.

2. Motivación

El Aprendizaje estadístico (Machine Learning) es una rama de la Inteligencia Artificial y dentro de la misma encontramos los algoritmos de Aprendizaje Profundo que se basan en la construcción de Redes Neuronales con varias capas ocultas. En este trabajo serán de particular interés las Redes Neuronales Convolucionales[26] que han surgido recientemente para resolver problemas de identificación de objetos y patrones visuales en imágenes, de esta manera se logra la identificación y clasificación de los mismos. Dos prominentes ejemplos de el uso de este tipo de algoritmos es el servicio de Alexa de Amazon para reconocimiento de voz y también el reconocimiento de patrones visuales en imágenes desarrollado por Google o Facebook.

Los algoritmos de Aprendizaje estadísticos han ganado mucha popularidad ya que se han producido dos hechos fundamentales:

- Las empresas y proveedores de servicios en internet utilizan los métodos de Aprendizaje estadístico para explotar su negocio con modelos basados en los datos.
- La capacidad de procesamiento mediante las unidades de procesamiento gráficos (GPU) y el almacenamiento han crecido y están disponibles a bajo costo.

En este sentido es posible, para cualquier empresa que tenga la disciplina de recolectar sus datos, inferir y clasificar información sobre los clientes, que permite desarrollar estrategias comerciales más enfocadas, personalizadas, flexibles y adaptables a los hábitos y usos de los productos y servicios de los clientes.

Amazon desde sus inicios empezó con su portal de comercio electrónico para comercializar libros y desde su inicio realiza recomendaciones basadas en las búsquedas, listas de deseo y compras que los clientes realizan desde la página web. Estas técnicas y la infraestructura construida por Amazon se ofrece como servicio a través de Amazon Web Servicios (AWS). Otras empresas, grandes como Amazon, han seguido sus pasos en la oferta de infraestructura en la nube como, Microsoft, Google e IBM.

El entendimiento de una de las plataformas más importantes de servicios en la nube abona en la dirección de entender el resto de los proveedores con una referencia sólida, basada en el uso de AWS, que es la plataforma en la nube de uso más extendido en la industria.

Entre los servicios que se pueden implementar en la infraestructura AWS de Amazon se encuentran, SageMaker y Apache **MxNet**.

Amazon SageMaker es un servicio de aprendizaje estadístico completamente gestionado. Con este los científicos de datos y desarrolladores pueden construir y entrenar modelos de aprendizaje e implementarlos en ambientes preparados para dar servicios en producción en la nube.

3. Objetivos

3.1. Objetivos Generales.

- Estudiar y aplicar los algoritmos de aprendizaje estadístico provistos por Amazon y desarrollar una metodología de trabajo para ofrecer servicios a través de AWS utilizando los bancos de imágenes MNIST y CIFAR.
- Explorar AWS DeepLens como plataforma didáctica para la aplicación de los algoritmos de aprendizaje en el campo del reconocimiento visual de patrones mediante Redes Neuronales Convolucionales.

3.2. Objetivos Específicos.

- Estudiar y explorar los servicios provistos por Amazon a través de su plataforma AWS y repasar de forma general todos los servicios de aprendizaje estadísticos provistos.
- Explorar los algoritmos provistos por Amazon SageMaker: Linear Learner, Factorization Machines, XGBoost Algorithmm, Image Classification Algorithm, Sequence2Sequence, K-Means Algorithm, Principal Component Analysis (PCA),

Latent Dirichlet Allocation (LDA), Neural Topic Model (NTM), DeepAR Forecasting, BlazingText y Random Cut Forest. Entender las características teóricas y modelos subyacentes, así como las referencias principales.

- Especificar los pasos y métodos necesarios para aplicar los algoritmos de aprendizaje con ejemplos didácticos. Realizar consideraciones prácticas para la implementación de servicios mediante AWS.
- Activar y Configurar AWS DeepLens.
- Explorar los casos de uso y métodos de ejemplo con enfoque didáctico.
- Aplicar algoritmos provistos por Amazon SageMaker para el reconocimiento de patrones.
- Especificar las actividades y pasos necesarios para la aplicación de algoritmos de aprendizaje o inferencia própios o de terceros en Amazon SageMaker.

Capítulo 2

Marco industrial

1. Amazon Web Services (AWS)

Vamos a estudiar y explorar los servicios provistos por Amazon a través de su plataforma AWS y repasar de forma general todos los servicios provistos.

El reto de muchos científicos de datos, una vez desarrollado y probado un modelo matemático, es su publicación y uso en aplicaciones reales. Amazon ha desarrollado múltiples componentes, servicios y soluciones que permite llevar el modelo a un ambiente donde puede ser utilizado por otras aplicaciones. Los modelos se pueden entrenar y publicar como un servicio, lo cual permite que sea utilizado por las aplicaciones y generar una solución que se puede integrar como componente a otras aplicaciones.

A continuación vamos a realizar un repaso de los servicios provistos por Amazon desde el punto de vista de la Infraestructura, Inteligencia Artificial y Modelos de Aprendizaje.

1.1. Servicios de AWS. Amazon es un proveedor de Infraestructura en la Nube como Servicio (IaaS). Ofrece servicios de almacenamiento estructurado o no estructurado y capacidad de cómputo.

1.1.1. *Almacenamiento.* Son varias las soluciones de almacenamiento de archivos y con diversos propósitos. Los dos usos más importantes son el almacenamiento de archivos y el almacenamiento en Bases de Datos.

- Amazon S3 (Amazon Simple Storage Service), es el servicio de almacenamiento de objetos como por ejemplo: archivos para aplicaciones móviles o web, respaldo, copias de seguridad o restauración. Se puede controlar el acceso a los archivos, es escalable de acuerdo a las necesidades, de alta disponibilidad, seguro y alto rendimiento.
- Bases de datos. En Amazon se ofrecen diversas opciones de almacenamiento en Bases de datos relacionales y no relacionales. Dentro de las no relacionales se destaca Amazon DynamoDB, que ofrece una opción de almacenamiento no relacional clave-valor útil para aplicaciones Web, almacenamiento de

configuraciones, etc, y por otra parte se ofrece el acceso a Amazon RDS que provee el acceso al almacenamiento en bases de datos relacionales como MySQL, PostgreSQL, Aurora, MariaDB, Oracle y SQL Server. Por otra parte se ofrece el almacenamiento de grandes cantidades de datos y gran desempeño Amazon RedShift y por último el servicio de almacenamiento temporal en la nube a través de Amazon ElastiCache.

1.1.2. *Capacidad de Cómputo.* El servicio de Computación Elástica de Amazon se denomina Amazon EC2 y ofrece capacidad de cómputo basada en CPU o en GPU, además de memoría dinámica (RAM) elástica. Son diversas las opciones y tamaños, además de las versiones optimizadas para procesamiento normal o acelerado o instancias intensivas en memoria o almacenamiento. Dependiendo de los requerimientos de la aplicación que se desea subir a la nube se pueden escoger las características de procesamiento más idoneas.

Dentro de los servicios más importantes podemos contar con:

- GPU, para acelerar el cálculo complejo y bajar los tiempos de entrenamiento. Se utilizan diversas opciones como:
  - GPU NVIDIA Tesla V100.
  - GPU NVIDIA Tensor Core V100.
  - GPU NVIDIA K80.
  - GPU NVIDIA Tesla M60.
- Elastic Inference, es la opción para agregar aceleración mediante el uso de GPU en Amazon SageMaker para reducir costos y tiempo de entrenamiento mediante los ambientes de trabajo de aprendizaje.

  En la fase de entrenamiento es útil contar con procesadores gráficos (GPU) para acelerar el proceso de estimación y ajuste de los modelos. Luego que se logra un modelo entrenado entonces sólo se realizan ciclos de estimación o inferencia que no requieren la capacidad completa del GPU. Es por ello que es útil proporcionar capacidad de procesamiento a través del GPU de forma elástica. Amazon permite agregar la cantidad de procesamiento por GPU requerida por la aplicación para evitar recursos subutilizados y costos innecesarios.
- AWS Inferentia, es el chip especializado en aprendizaje, que pronto estará disponible, para realizar inferencias en modelos realizados en los ambientes de

trabajo de aprendizaje como **TensorFlow**, Apache **MxNet** y **PyTorch**. Es un chip especializado para las aplicaciones que requieren mucha capacidad y baja latencia para realizar inferencias.

- CPU, con gran capacidad y diversas características para optimizar el cálculo para aplicar cifrado de datos, cálculos con números reales y aceleración para Aprendizaje Profundo. En las distintas opciones se utilizan los procesadores siguientes:
  - AMD EPYC 7000 de 2,5 GHz.
  - Intel Xeon personalizados de segunda generación con escala ajustable (Cascade Lake) con una frecuencia Turbo estable en todos los núcleos de 3,6 GHz y una frecuencia Turbo de núcleo individual hasta 3,9 GHz.
  - Intel Xeon de segunda generación con escala ajustable (Cascade Lake) o en un procesador Intel Xeon Platinum serie 8000 (Skylake-SP) de primera generación con una frecuencia Turbo estable en todos los núcleos de 3,4 GHz y una frecuencia Turbo de núcleo individual hasta 3,5 GHz.
  - Intel Xeon E5 2686 v4 de 2,3 GHz (base) y 2,7 GHz (turbo).
  - Intel Xeon P-8175M 2,5-GHz (base) de alta frecuencia.
  - Intel Xeon Platinum 8175 hasta 3,1 GHz.
- FPGA, es la opción que ofrece Amazon para crear instancias de computación elástica aceleradas por hardware. Se ofrece un conjunto de herramientas de desarrollo para programar la instancia, permitiendo a los desarrolladores la creación de una imagen AFI (Amazon FPGA Image) que se puede reutilizar y compartir.
- Edge, representa las facilidades que ofrece Amazon para distribuir las aplicaciones en los distintos dispositivos. Los dispositivos pueden ejecutar funciones de AWS Lambda como la inferencia en algoritmos de aprendizaje, sincronización de datos y comunicación entre dispositivos. Con AWS IoT Greengrass se pueden utilizar diversos lenguajes y paradigmas de programación para crear aplicaciones que se van a distribuir a muchos dispositivos.

1.1.3. *Amazon Lambda*. Uno de los servicios recientes es Amazon Lambda, que consiste en la implementación de software en la nube sin necesidad de gestionar servidores. Sólo se sube el código que se desea ejecutar y se pueden utilizar los lenguajes siguientes: NodeJS, Java, Python, Ruby, Go, C# y PowerShell.

1.2. Servicios de Inteligencia Artificial. Son los servicios de alto nivel que permiten incorporar funcionalidades que aprovechan las técnicas de Inteligencia Artificial a las aplicaciones, sin necesidad de desarrollar un modelo. El objetivo es que las empresas puedan implementar servicios sin desarrollar tecnología propia, ni contar con equipos altamente especializados para desarrollar e implementar aplicaciones que incorporan las técnicas de la Inteligencia Artificial.

Por ejemplo se pueden utilizar los Servicios de Inteligencia Artificial para desarrollar Robots de atención al cliente a través de un centro de soporte telefónico (Call center), realizar proyecciones de demanda, análisis de imágenes, procesar lenguaje natural, implementar servicios de recomendaciones, asistentes virtuales, entre otros.

Los servicios se pueden utilizar de forma separada o se pueden componer para implementar servicios más sofisticados. La gran ventaja es que estos servicios utilizan la misma tecnología que Amazon utiliza en su propio negocio.

A continuación vamos a repasar cada uno de los Servicios de Inteligencia Artificial ofrecidos por Amazon.

1.2.1. *Amazon Comprehend.* Este servicio analiza textos en lenguaje natural e identifica el idioma, identifica entidades como (personas, lugares, marcas, productos, eventos, etc), frases, sentimiento (positivo, negativo, neutral o mixto), extrae frases en inglés o español. Adicionalmente es capaz de extraer tópicos de grandes cantidades de documentos para el análisis o la clasificación de documentos.

Amazon ofrece sus servicios de forma interactiva o por lotes. El acceso interactivo se realiza a través del acceso a servicios web en línea, los servicios ofrecidos por lotes consiste en proporcionar los datos al servicio por lotes y esperar que Amazon realice el análisis y genere los resultados del análisis. Las funciones disponibles de forma interactiva son:

- Detección del idioma.
- Categorización e identificación de entidades.
- Análisis de sentimiento.
- Extracción de frases clave.

Extracción de tópicos y la clasificación de documentos se realiza por lotes y la duración es proporcional al tamaño de los datos.

Se puede utilizar AutoML para contruir conjuntos de entidades o modelos de clasificación de texto que se adecuan a las necesidades específicas de la empresa.

Amazon Comprehend es gestionado por Amazon, no se necesita implementar servidores, y no amerita el desarrollo de modelos de aprendizaje.

Para mayores detalles se puede consultar el artículo sobre Amazon Comprehend.

1.2.2. *Ejemplo de conexión al API Amazon Comprehend.* Vamos a realizar la conexión al API Amazon Comprehend a través de la línea de comandos para mostrar el procedimiento y los resultados que genera la interfaz para programadores.

El primer paso es instalar las herramientas para línea de comandos. En los comandos descritos en 2.1 se realiza la descarga, instalación y configuración de la herramienta.

```
# Descarga de desempque del instalador
>curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-
   bundle.zip"
>unzip awscli-bundle.zip
# Instalacion y verificacion
>sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
>aws --version
# Configuracion de credenciales
>aws configure --profile personal
AWS Access Key ID [None]: xxx
AWS Secret Access Key [None]: xxxx
Default region name [None]: us-east-1
Default output format [None]: json
```

Código fuente 2.1. Instalación de las herramientas para la línea de comandos

Una vez instalado podemos utilizar la funcionalidad provista por el API. En primer lugar vamos a ver cómo se utiliza la funcionalidad de detección del lenguaje. El comando que se debe ejecutar es `aws comprehend detect-dominant-language`. Los comandos descritos en código fuente 2.2, se puede ver que se especifica la región de AWS donde se desea ejecutar y el texto. El resultado indica el código del lenguaje y la probabilidad.

```
>aws comprehend detect-dominant-language --region us-east-1 --text "It is
   a great day in Caracas."
```

```
{ "Languages": [{
            "LanguageCode": "en",
            "Score": 0.9683481454849243
        }]
}
>aws comprehend detect-dominant-language --region us-east-1 --text "Es un
    buen dia en caracas"
{ "Languages": [{
            "LanguageCode": "es",
            "Score": 0.9995386004447937
        }]
}
```

Código fuente 2.2. Detección del idioma

Amazon Comprehend ofrece la detección de sentimientos en el texto a través del comando `aws comprehend detect-sentiment`. En los comandos descritos en código fuente 2.3 se especifica la región el idioma y el texto y el resultado indica el sentimiento que predomina y la puntuación del sentimiento dividido en las categorías mixto, neutral, positivo o negativo. En el primer ejemplo código fuente 2.3 podemos ver la progresión de la puntuación del sentimiento de acuerdo a la frase y las palabras que se utilizan.

```
>aws comprehend detect-sentiment --region us-east-1 --language-code "es"
    --text "Es un dia en Caracas."
{ "SentimentScore": {
        "Mixed": 0.00826080609112978,
        "Positive": 0.06352242827415466,
        "Neutral": 0.9008195996284485,
        "Negative": 0.027397137135267258
    },
  "Sentiment": "NEUTRAL" }
>aws comprehend detect-sentiment --region us-east-1 --language-code "es"
    --text "Es un buen dia en Caracas."
{ "SentimentScore": {
        "Mixed": 0.025890501216053963,
```

```
        "Positive": 0.7085699439048767,
        "Neutral": 0.24981053173542023,
        "Negative": 0.015728970989584923
    },
  "Sentiment": "POSITIVE" }
>aws comprehend detect-sentiment --region us-east-1 --language-code "es"
   --text "Es un excelente dia en Caracas."
{ "SentimentScore": {
        "Mixed": 0.013180622830986977,
        "Positive": 0.9181721210479736,
        "Neutral": 0.06409947574138641,
        "Negative": 0.004547752905637026
    },
  "Sentiment": "POSITIVE" }
```

Código fuente 2.3. Análisis de sentimiento

En el segundo ejemplo en el código fuente 2.4 podemos ver cómo el sentimiento predominante detectado es positivo y luego negativo en consistencia con la frase.

```
>aws comprehend detect-sentiment --region us-east-1 --language-code "es"
   --text "La UCV es la mejor Universidad de Venezuela."
{
    "SentimentScore": {
        "Mixed": 0.044829096645116806,
        "Positive": 0.7631200551986694,
        "Neutral": 0.15071652829647064,
        "Negative": 0.04133433476090431
    },
    "Sentiment": "POSITIVE"
}
>aws comprehend detect-sentiment --region us-east-1 --language-code "es"
   --text "La UCV es una mala Universidad de Venezuela."
{
    "SentimentScore": {
        "Mixed": 0.019124921411275864,
```

```
        "Positive": 0.0036907221656292677,
        "Neutral": 0.038850728422403336,
        "Negative": 0.9383335113525391
    },
    "Sentiment": "NEGATIVE"
}
```

Código fuente 2.4. Análisis de sentimiento

En este tercer ejemplo en el código fuente 2.5 hemos utilizado la ironía y se puede apreciar que no se detecta y se clasifica como positiva. Este es un ejemplo de los límites de este tipo de servicios y nos sugiere que hay todavía mucho camino por andar en la detección de sentimientos.

```
>aws comprehend detect-sentiment --region us-east-1 --language-code "es"
   --text "La UCV es la mejor Universidad. Si, si.."
{
    "SentimentScore": {
        "Mixed": 0.10462002456188202,
        "Positive": 0.7700151801109314,
        "Neutral": 0.1027987003326416,
        "Negative": 0.02256607636809349
    },
    "Sentiment": "POSITIVE"
}
>aws comprehend detect-sentiment --region us-east-1 --language-code "es"
   --text "Sigue creyendo que la UCV es la mejor universidad. La UCV es la
    mejor Universidad. Si, si.."
{
    "SentimentScore": {
        "Mixed": 0.22400306165218353,
        "Positive": 0.6542226076126099,
        "Neutral": 0.0933767780661583,
        "Negative": 0.02839753031730652
    },
    "Sentiment": "POSITIVE"
```

```
}
```

Código fuente 2.5. Ironía

Por otra parte también podemos identificar las entidades, nombres, lugares u objetos que aparecen en el texto. Se utiliza el comando `aws comprehend detect-entities` permite obtener la lista de entidades detectadas y su tipo. En el ejemplo se detecta una persona y una localidad.

```
>aws comprehend detect-entities --region us-east-1 --language-code "es" --
   text "Alexander esta en venezuela y esta feliz"
{ "ResultList": [{
            "Index": 0,
            "Entities": [{
                    "Text": "Alexander",
                    "Score": 0.9653822779655457,
                    "Type": "PERSON",
                    "BeginOffset": 0,
                    "EndOffset": 9
                },{
                    "Text": "venezuela",
                    "Score": 0.7800759673118591,
                    "Type": "LOCATION",
                    "BeginOffset": 18,
                    "EndOffset": 27
                }]
        }],
    "ErrorList": []
}
```

Código fuente 2.6. Detección de entidades

Otra funcionaliad ofrecida es la identificación de frases importantes mediante el comando `aws comprehend detect-key-phrases`. El resultado consiste en una lista de las frases detectadas y su lugar en el texto.

```
>aws comprehend detect-key-phrases --region us-east-1 --language-code "es"
    --text "Alexander esta en venezuela y esta feliz"
{
  "KeyPhrases": [{
            "Text": "Alexander",
            "Score": 0.7463587522506714,
            "BeginOffset": 0,
            "EndOffset": 9
        },{
            "Text": "venezuela",
            "Score": 0.9625979661941528,
            "BeginOffset": 18,
            "EndOffset": 27
        },{
            "Text": "esta feliz",
            "Score": 0.7769358158111572,
            "BeginOffset": 30,
            "EndOffset": 40
        }]
}
```

Código fuente 2.7. Detección de frases importantes

1.2.3. *Amazon Comprehend Medical*. De forma análoga Amazon Comprehend Medical extrae información relevante de fuentes de textos de información médica, como informes médicos, diagnósticos, resultados de estudios o pruebas, etc.

Se puede recolectar información como estado de salud, medicamentos, dosis, cantidad, frecuencia, etc. Esta información se extrae de los reportes médicos, notas, registros médicos, etc.

Existe una gran cantidad de información en formato libre de la cual se puede extraer información que permite mejorar el cuidado y acelerar la investigación relacionando la información de los pacientes.

Los modelos de aprendizaje se adaptan e identifican relaciones entre los datos que mejoran con el tiempo. El acceso se realiza a través de llamadas a la interfaz para programadores (API).

1.2.4. *Amazon Forecast.* Este servicio se especializa en realizar proyecciones utilizando algortimos de aprendizaje a través de una plataforma gestionada por Amazon.

Amazon Forecast utiliza las técnicas de aprendizaje para combinar series de tiempo con variables adicionales para realizar proyecciones. Sólo se provee la data histórica y cualquier variable adicional que puede impactar la proyección.

La proyección de la demanda, recursos o desempeño financiero se realiza mediante el estudio de los datos históricos y se asume que el comportamiento pasado es un reflejo de lo que puede ocurrir en el futuro. Sin embargo, con las técnicas de aprendizaje se pueden incorporar otras variables que pueden tener impacto y generar proyecciones más ajustadas.

Este servicio es gestionado completamente por Amazon y se accede a través de la interfaz para programadores (API) provisto por Amazon.

1.2.5. *Amazon Lex.* Este servicio permite crear interfaces con capacidad para conversar a través de voz o texto. Amazon Lex provee funcionalidades avanzadas de Aprendizaje Profundo para el reconocimiento de la voz y la conversión de voz a texto además del reconocimiento del lenguaje natural para identificar la intención.

Con estas capacidades se pueden construir aplicaciones capaces de interactuar con los usuarios mediante conversaciones similares a las que se realizan con operadores reales. Las mismas tecnologías que habilitan el funcionamiento de Amazon Alexa se ofrecen a los desarrolladores para que puedan construir sus propias aplicaciones de atención al cliente.

1.2.6. *Amazon Personalize.* Este servicio habilita a las aplicaciones en la generación de recomendaciones a los clientes.

El servicio recibe la información sobre los clientes, sus accesos, páginas visitadas, interacciones, suscripciones, compras, búsquedas, intereses, inventario existente de los artículos, productos, videos y este produce recomendaciones ajustadas a las preferencias conocidas del usuario y las preferencias de otros usuarios similares. Se pueden incorporar datos demográficos como la edad y localización, entre otros.

Se pueden generar recomendaciones sobre productos, contenidos, mercadeo dirigido y resultados de búsqueda personalizados. Amazon Personalize examina los datos e identifica

lo que es relevante para seleccionar el algoritmo que mejor se ajusta, lo entrena y optimiza para generar un modelos de personalización ajustado a los datos particulares. Los datos proporcionados son privados y se protegen y se utilizan de forma exclusiva en el modelo.

El servicio de accede a través de la interfaz para programadores (API).

1.2.7. *Amazon Polly*. Este servicio realiza la conversión de texto a voz y permite la creación de aplicaciones que conversan mediante el uso de algoritmos de Aprendizaje Profundo que sintetizan la voz lo más parecido a la voz humana.

En el ejemplo siguiente nos conectamos al API Amazon Polly y le proporcionamos una frase para que se produzca la generación del audio respectivo. Se utiliza el comando `aws polly synthesize-speech` y se especifica la región, el formato del audio esperado, el texto, la voz y el nombre del archivo donde se desea almacenar el resultado.

```
aws polly synthesize-speech --engine "standard" --language-code "es-ES" --
    output-format "mp3" --text "La U C V es la mejor universidad de
    venezuela" --text-type "text" --voice-id "Conchita" --region us-east-1
    "ucv.mp3"
```

Código fuente 2.8. Texto a Voz

Haz click para escuchar el resultado de Amazon Polly

Se ofrece una variedad de idiomas y voces masculinas o femeninas de distintos estilos para desarrollar aplicaciones para distintos paises, se pueden hacer aplicaciones con un estilo de narración de noticias.

1.2.8. *Amazon Rekognition*. Este servicio ofrece el análisis de imágenes y videos para identificar objetos, personas, texto, paisajes, actividades o inclusive contenido inapropiado. Se ofrece la funcionalidad de análisis facial y reconocimiento facial en las imágenes o videos proporcionados. Se pueden detectar, analizar y comparar rostros con fines de seguridad y además conteo de personas.

Se utiliza a través de la interfaz para programadores y se proporciona el contenido a través de Amazon S3.

1.2.9. *Amazon Textract*. Es un servicio de digitalización inteligente de información. Se extrae información de documentos escaneados en formato libre o en formas o tablas. De esta manera no es necesario el proceso de digitalización de documentos manual. Mediante el uso

de las técnicas de aprendizaje se puede extraer texto de cualquier documento sin procesos manuales.

Se puede integrar con procesos de procesamiento de solicitudes y recepción de recaudos, además se pueden crear índices de búsquedas, construcción de flujos de trabajo para aprobaciones u otro proceso que requiera el procesamiento de datos en planillas.

1.2.10. *Amazon Translate*. Es el servicio de traducción de información basado en redes neuronales y Aprendizaje Profundo para producir traducciones más naturales y exactas. Se puede utilizar para traducir contenido de sitios web así como para el procesamiento de grandes cantidades de texto.

En la secuencia de comandos descrita en 2.9 se puede ver un ejemplo de conexión al API Amazon Translate desde la línea de comandos. Se especifica el texto, el idioma de origen y el idioma en el cual se desea el resultado.

```
>aws translate translate-text --text "UCV is the best university." --
    source "en" --target "es" --region us-east-1
{
    "TargetLanguageCode": "es",
    "TranslatedText": "UCV es la mejor universidad.",
    "SourceLanguageCode": "en"
}
>aws translate translate-text --text "La UCV es la mejor universidad." --
    source "es" --target "en" --region us-east-1
{
    "TargetLanguageCode": "en",
    "TranslatedText": "The UCV is the best university.",
    "SourceLanguageCode": "es"
}
```

Código fuente 2.9. Traducción de texto

1.2.11. *Amazon Transcribe*. Es el servicio que convierte voz a texto y realiza la transcripción mediante el reconocimiento automático de voz. Se pueden transcribir audios almacenados en Amazon S3 y se recibe como resultado un archivo de texto con la transcripción de la voz.

Adicionalmente, también se puede enviar un flujo de audio en vivo para realizar transcripciones en tiempo real. Se puede utilizar en casos de uso como la transcripción de llamadas a los servicios de soporte o generación de subtítulos para audios o videos.

Se pueden transcribir archivos en formato WAV o MP3.

1.2.12. *Soluciones de AWS.* A partir de los Servicios de Inteligencia Artificial se pueden componer soluciones que resuelven un problema en un dominio específico. Por ejemplo, se puede usar una solución que transcribe, traduce y analiza las interacciones de un usuario a través de un servicio telefónico de atención al cliente. Se realiza el Análisis de la voz y se genera la información que alimenta a los servicios de Inteligencia Artificial de Amazon para generar información útil que permita mejorar la experiencia de atención al cliente. Se puede apreciar en la figura 2.1 los componentes que intervienen en la generación del resultado. En este caso se utiliza Amazon Connect para procesar el audio de las llamadas teléfonicas, se utilizan algunas funciones para recibir el resultado y enviarlo a Amazon Transcribe, el resultado se almacena en Amazon S3 y en una Base de datos Amazon DynamoDB. Esta base de datos está accesible a través de una interfaz para programadores (API) que a su vez es consumida por la aplicación de atención al cliente que utiliza el Agente de atención. Ese texto a su vez se puede envíar a través de una interfaz para programadores a los servicios de traducción (Amazon Translate) y los servicios de procesamiento de lenguaje natural (Amazon Comprehend).

1.3. Servicios de Aprendizaje Automático.

- Amazon SageMaker

  Amazon SageMaker es un servicio de aprendizaje automatizado completamente administrado por Amazon. El servicio permite construir, entrenar y publicar modelos de aprendizaje en la nube para ser consultados y consumidos desde las aplicaciones de cualquier tipo, en particular aplicaciones web y móviles.

  Provee un cuarderno de notas de Jupyter para facilitar el acceso a los datos y realizar procesos de exploración de datos en los servidores de Amazon. Adicionalmente provee una serie de algoritmos de aprendizaje que están optimizados para procesar grandes cantidades de datos en ambientes distribuidos.

  Amazon SageMaker permite implementar algoritmos propios o inclusive utilizando otros ambientes de desarrollo como **Tensorflow** o **keras**. Los modelos

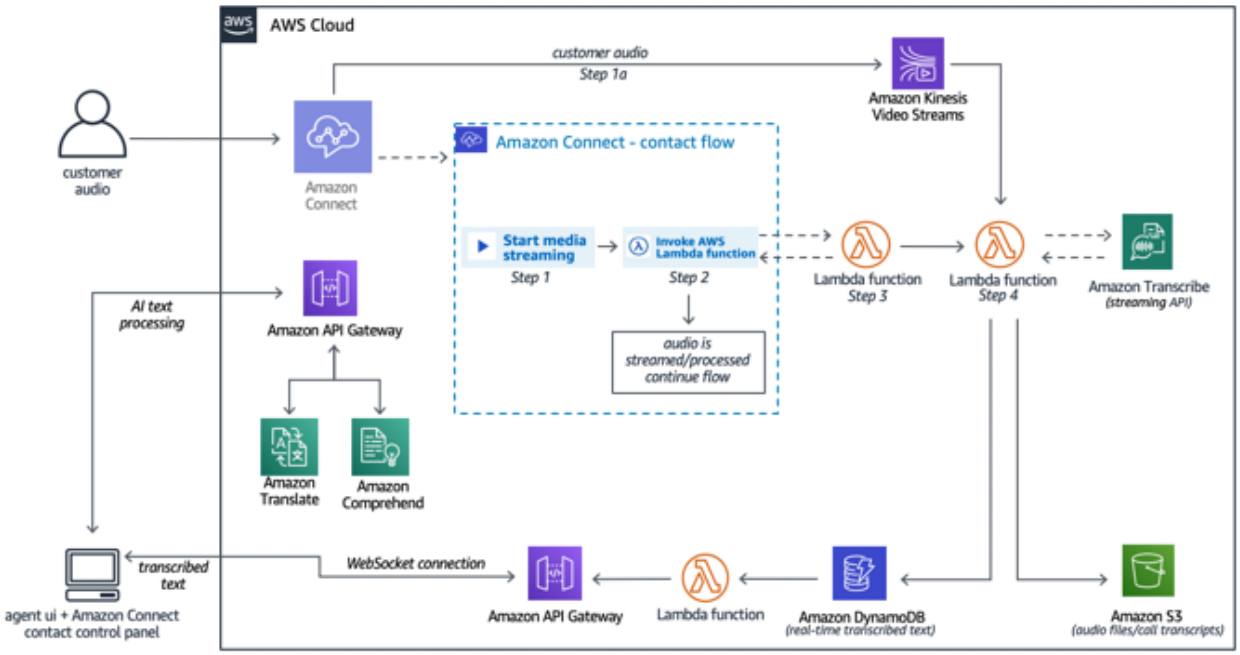


FIGURA 2.1. Análisis de voz

se pueden implementar aprovechando las facilidades de seguridad y escalabilidad ofrecida por Amazon Web Services. Se puede visualizar el proceso el proceso en la imagen 2.2.

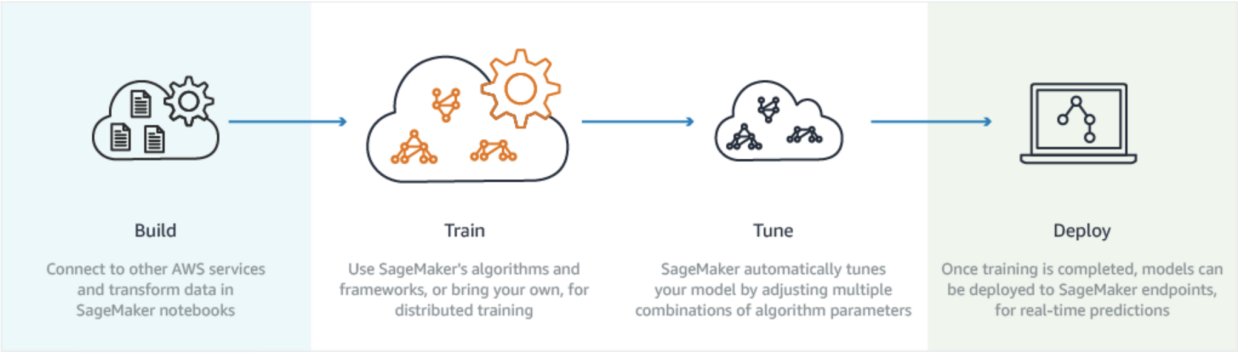


FIGURA 2.2. Amazon SageMaker

- Amazon SageMaker Ground Truth

  Con la popularización de los algoritmos de entrenamiento supervisados se ha hecho necesaria la generación de nuevas bases de datos etiquetadas para entrenar nuevos modelos. El servicio que ofrece Amazon consiste en etiquetar bases de datos de forma automatizada o manual.

Se puede utilizar la fuerza de trabajo de Amazon Mechanical Turk, algún proveedor propio o interno, así como los algoritmos de aprendizaje para crear la bases de datos etiquetadas.

El propósito es que estas bases de datos etiquetadas sirvan para entrenar modelos de aprendizaje con SageMaker. Amazon ofrece las herramientas automatizadas o manuales para realizar el proceso de etiquetado. En el caso manual se ofrece un conjunto de herramientas a través de páginas web que dan las instrucciones para etiquetar los datos. Los datos se almacenan en contenedores de Amazon S3 que contienen los datos a ser etiquetados, la descripción de los datos y la descripción del resultado esperado.

- Amazon SageMaker Neo

  Este servicio ofrece la capacidad para entrenar modelos y optimizarlos para ejecutarlos en la nube o en dispositivos de diversas arquitecturas.

  El objetivo es que se puedan utilizar los modelos de aprendizaje ya entrenados directamente en los dispositivos para evitar el consumo de servicios en la nube para generar inferencias. Por ejemplo, los sensores de los vehículos autónomos deben responder en milisegundos para ser útiles, es por ello que los modelos entrenados deben optimizarse para ser ejecutados en el vehículo sin necesidad de enviar datos o conectarse a otros servicios.

  Los modelos pueden optimizarse para ser ejecutados en procesadores Intel, NVIDIA o ARM. Los modelos se distribuyen con Amazon Greegrass de forma similar a como se distribuyen los modelos entrenados en Amazon Deeplens.

  Amazon Sagemaker Neo es de código fuente abierto lo cual permite que los desarrolladores puedan incorporar nuevos dispositivos. El repositorio está alojado en github.

1.4. Algoritmos de Aprendizaje Automático provistos en Amazon SageMaker.

Amazon SageMaker ofrece una diversidad de algoritmos pre programados y configurables que permite resolver una gran cantidad de problemas de regresión o clasificación supervisado o no supervisado.

Vamos a realizar una descripción breve de cada algoritmo provistos por Amazon SageMaker, vamos a repasar las características principales, referencias y ejemplos de uso.

Cada algoritmo cuenta con ejemplos provistos por Amazon para entender su uso con datos de ejemplo. Cada uno representa una línea de trabajo que permite solucionar problemas aplicando técnicas de Aprendizaje Automático.

Cabe destacar que el uso de estos algoritmos amerita el uso de servicios de almacenamiento para los datos de entrenamiento, datos de prueba, datos nuevos y resultados inferidos, así como la capacidad de procesamiento necesaria para realizar el proceso de entrenamiento y luego las inferencias mediante un servicio.

1.4.1. *BlazingText.* Es un algoritmo no supervisado para generar nuevas representaciones de una palabra que utiliza Word2Vec . El algoritmo intenta predecir la presencia de una palabra partiendo de las palabras adjacentes utilizando el saco de palabras (Bag-Of-Words - CBOW) o intenta predecir el contexto a partir de una palabra utilizando SGNS (Skip-Gram with Negative Sampling). La referencia se puede ver en [7].

1.4.2. *DeepAR Forecasting.* Es un algoritmo supervisado que realiza proyecciones siguiendo el método descrito en [8].

Se utiliza para realizar proyecciones en series de tiempo utilizando Redes Neuronales Recurrentes, que son Redes con ciclos que permiten que la información fluya con memoria, permitiendo la generación de inferencias sobre el valor siguiente.

Aunque se ensambla un modelo para cada serie de tiempo, se utiliza la información de otras series de tiempo para realizar proyecciones identificando las similitudes. Se puede utilizar para proyectar la demanda de productos o mano de obra. La documentación en Amazon está disponible, así como una descripción de cómo funciona en SageMaker.

1.4.3. *Factorization Machines.* Es un algoritmo de aprendizaje supervisado que se utiliza para hacer regresión o clasificación que sigue el método descrito en [9].

El método es una extensión de los modelos lineales que se diseña para extraer interacciones entre características o variables en datos dispersos (sparse) y multi dimensionales. La predicción de clicks o recomendaciones de productos pueden ser un buen problema de aplicación. Se puede seguir el ejemplo documentado por Amazon.

1.4.4. *Image Classification Algorithm.* Es un algoritmo supervisado que realiza clasificaciones múltiples. Toma una imagen y genera un vector de etiquetas asociadas a la imagen. Utiliza ResNet (ver 4.8) como arquitectura de una Red Neuronal Convolucional

que se puede entrenar utilizando aprendizaje por transferencia o partiendo de una Red sin entrenamiento. La referencia del método se puede ver en [10]

Para entender mejor el método se puede consultar el ejemplo en MxNet. Adicionalmente se puede seguir un tutorial del blog de AWS.

1.4.5. *IP Insights.* Es un algoritmo no supervisado con aplicaciones en seguridad de la información que detecta patrones de uso de las direcciones IPv4. Está diseñado para identificar patrones que asocian las direcciones IP con los usuarios y cuentas. Dos usuarios que utilizan la misma dirección IP se relacionan. Las relaciones generan una puntuación sobre el nivel de sospecha o anomalía que representa el patrón de un evento. En base a esta puntuación el servidor web podría decidir si solicitar credenciales o factores de seguridad adicionales. Se puede consultar el ejemplo disponible en la documentación del servicio.

1.4.6. *K-Means Algorithm.* Es un algoritmo no supervisado que agrupa objetos similares de acuerdo a una función de distancia en un espacio n-dimensional. Cada grupo tiene un centro y se calcula la distancia de cada punto con relación a los centros. El analista decide cuantos grupos utilizar.

1.4.7. *K-Nearest Neighbors (k-NN) Algorithm.* El algoritmo de los k-vecinos más cercanos utiliza un método no paramétrico para realizar clasificación o regresión. Para clasificar utiliza los k puntos más cercanos y utiliza el valor observado más frecuente en los datos para indicar la clase a la que pertenece el dato. Para problemas de regresión toma los k-vecinos más cercanos y utiliza el promedio de los vecinos para generar la inferencia. En el proceso se genera un índice que permite realizar búsquedas y generar un resultado más rápido para nuevos datos.

Se puede revisar el ejemplo en los cuadernos de notas de Amazon.

1.4.8. *Latent Dirichlet Allocation (LDA).* Es un algoritmo no supervisado que intenta describir cada observación como una mezcla de categorías. Las características pueden pertenecer a distintas categorías. Un ejemplo típico es la clasificación de documentos por materia dependiende de la presencia de ciertas palabras.

Las materias se aprenden, como la distribución de probabilidad sobre las palabras que aparecen en cada documento. Cada documento pertenece a varias materias.

Se provee un ejemplo que introduce el método en la documentación de AWS.

1.4.9. *Linear Learner.* Es un algoritmo supervisado que implementa un modelo lineal para resolver problemas de clasificación o regresión. Se provee un vector de dimensión p con un valor observado asociado al vector. Para realizar una clasificación binaria el valor observado debe ser un valor entre 0 o 1 o en el caso de m clases, debe ser un valor entre 0 y $m-1$. Para realizar una regresión el valor observado debe ser un número real.

Se pueden evaluar distintas funciones de pérdida y tomar la que mejor se ajuste al modelo utilizando una variedad de funciones de pérdida y parámetros. Se proveen algunos cuadernos de notas de ejemplo y la aplicación del método para realizar la clasificación binaria que detecta si un dígito en el banco de imágenes MNIST es un cero.

1.4.10. *Neural Topic Model (NTM).* Es un algoritmo no supervisado que se utiliza para organizar bases de datos de documentos por materias, mediante la distribución estadística de las grupos de palabras que contiene el mismo. Se le debe especificar al algoritmo el número de materias, no las materias en sí mismo. Los documentos se indexan para consiguir documentos en la base de entrenamiento o para clasificar nuevos documentos con características similares. El algoritmo está basado en el trabajo descrito en [11].

1.4.11. *Object2Vec.* Es un algoritmo que permite reducir la dimensión de los datos generando un vector de características de menor dimensión que la dimensión de los datos observados. De esta forma se extraen características que permite la clasificación de los datos originales utilizando la nueva representación en una menor dimensión. Este método se utiliza para análisis de sentimientos, clasificación de documentos y procesamiento de lenguaje natural.

Adicionalmente se puede utilizar para extraer características a las frases, datos de los clientes o productos y de esta forma agrupar datos similares. Se puede consultar el blog de AWS para más información sobre las aplicacioens.

1.4.12. *Object Detection Algorithm.* Es un algoritmo que detecta objetos que utiliza una Red Neuronal Profunda. Es un algoritmo de aprendizaje supervisado que toma las imágenes como entrada e identifica todas las repeticiones de un objeto en una imagen. Los objetos se clasifican de acuerdo a las clases definidas en los datos observados. La localización y escala de cada objeto identificado se devuelve como resultado. Utiliza SSD (Single Shot multibox Detector) con VGG y ResNet. Se puede entrenar la Red partiendo de un modelos

sin entrenamiento y desde un modelo pre entrenado. Se puede consultar el ejemplo para más detalles.

1.4.13. *Principal Component Analysis (PCA).* Es un algoritmo no supervisado que identifica las características que representan la mayor cantidad de información (varianza) en un conjunto de datos, de manera que se pueda reducir la dimensión de los datos. Cada nueva característica, que es una combinación de las características originales, serán independientes entre ellas. Pueden consultarl el blog de AWS para más información.

1.4.14. *Random Cut Forest.* Es un algoritmo no supervisado que sirve para identificar datos anómalos dentro de un conjunto de datos. De esta manera se pueden realizar procesos de limpieza en conjuntos de datos para bajar su complejidad y poder aplicar otros métodos. Consulta el ejemplo provisto por AWS para más información.

1.4.15. *Semantic Segmentation.* Consiste en clasificar cada punto de una imagen (pixel) de acuerdo a un conjunto de clases provisto. La clasificación usualmente se representa como una imagen con la misma dimensión que la imagen de entrada con una máscara de color que diferencia los objetos de la imagen.

Se utilizan los algoritmos descritos en [12], [13] y [14].

Puede consultar el ejemplo para más información.

1.4.16. *Sequence2Sequence.* Es un algoritmo supervisado que utiliza Redes Neuronales Recurrentes y Redes Neuronales Convolucionales para generar secuencias a partir de secuencias, útil en la traducción automática de textos o la conversión de texto a voz. Puede consultar un ejemplo provisto que realiza una traducción del Inglés al Alemán.

1.4.17. *XGBoost Algorithmm.* Es una implementación del algoritmo supervisado basado en árboles de decisión que intenta inferir una variable objetivo, realizando una combinación ponderada (boosting) de un conjunto de estimados producidos por otros modelos de menor desempeño . Maneja un gran cantidad de de tipos de datos, relaciones y distribuciones y se puede utilizar para ordenar, realizar regresiones o clasificación binaria o múltiple. Puede consultar el ejemplo provisto por AWS.

1.5. Marcos de trabajo. En los últimos años se han desarrollado diversos ambientes de trabajo para la investigación y desarrollo de aplicaciones con técnicas de aprendizaje. Entre ellos **TensorFlow**, **PyTorch** o Apache **MxNet**, entre otros.

1.5.1. *AMI de Aprendizaje Profundo de AWS.* Consiste en una variedad de instancias de computación elástica previamente configuradas para trabajar con diversos marcos de trabajo como **TensorFlow**, **PyTorch** o Apache **MxNet**.

Existen instancias con CPU, varios CPU, GPU, desde 1 o varios. En particular la instancia viene preconfigurada con NVIDIA CUDA o cnDNN. Estas máquinas aceleran la implementación de los ambientes de trabajo de Aprendizaje Profundo con diversas opciones de procesamiento.

1.5.2. *Contenedores de Aprendizaje Profundo de AWS.* Son diversas las opciones para entrenar modelos de aprendizaje, podemos contar con contenedores preconfigurados o utilizar Amazon SageMaker. Los contenedores AMI consisten en ambientes pre instalados y preconfigurados para realizar investigación y desarrollo de aplicaciones con Aprendizaje Profundo.

1.5.3. *Tensorflow en AWS.* **TensorFlow** es un ambiente de trabajo que ofrece las facilidades para realizar cálculos que aprovechan la capacidad de procesamiento disponible como CPU o GPU. En particular la librería **keras** utiliza **TensorFlow** para implementar Redes Neuronales Artificiales e inclusive Redes Neuronales Convolucionales.

Se pueden crear ambientes de trabajo con **TensorFlow** a través de Amazon SageMaker o implementar una instancia utilizando un AMI de Aprendizaje Profundo.

1.5.4. *PyTorch en AWS.* **PyTorch** es un ambiente de trabajo de Aprendizaje Profundo de código abierto que permite desarrollar modelos de aprendizaje. Se puede utilizar a través de Amazon SageMaker o un ambiente preconfigurado como un AMI de Aprendizaje Profundo.

1.5.5. *Apache MXnet en AWS.* Apache **MxNet** en un marco de trabajo para implementar algoritmos de aprendizaje que incluye la interfaz de Gluon para realizar modelos de Aprendizaje Profundo en dispositivos de diversas arquitecturas. Se puede utilizar a través de Amazon SageMaker o a través de un ambiente preconfigurado como una AMI de Aprendizaje Profundo.

1.6. Aprenda ML de AWS. Adicional a las facilidades para implementar ambientes de trabajo de Aprendizaje Profundo con Amazon SageMaker o las instancias de computación elástica preconfiguradas denominadas AMI, Amazon ha lanzado varios dispositivos que están configurados para utilizar las facilidades de Amazon para desplegar aplicaciones de Aprendizaje Profundo, entre ellos DeepRacer y DeepLens.

1.6.1. *DeepRacer.* Es un carro de carreras a escala que utiliza el aprendizaje reforzado (Reinforcement Learning). Tiene una escala de 1/18 y sirve para aprender y probar las técnicas de aprendizaje no supervisado para la conducción autónoma. Se implementan los algoritmos en la nube para correr las pistas virtuales del simulador de carreras 3D, para luego implementarlos en el carro de carreras.

FIGURA 2.3. Aprendizaje Reforzado con DeepRacer

1.6.2. *DeepLens.* Es una cámara inalámbrica con una interfaz para programadores, permite implementar los modelos de vision por computadora más recientes, con procesamiento en tiempo real. Los modelos son Redes Neuronales Convolucionales que implementan la identificación de patrones en imágenes.

Amazon ofrece la tecnología para implementar modelos que se despliegan en la cámara para probar el modelo y visualizar el resultado de aplicar un modelo a un flujo de video en tiempo real.

Como se puede apreciar en el gráfico 2.4 la cámara captura el video y produce dos salidas, la primera es el flujo de video sin procesar o flujo del dispositivo y la segunda la genera una función de inferencia que recibe el flujo de video sin procesar y lo pasa al modelo de inteligencia artificial para ser procesado, la función recibe el resultado del modelo y genera el flujo del proyecto.

Los modelos se pueden implementar en otros ambientes de desarrollo o a través de otras infraestructuras como Amazon SageMaker y se despliega en el dispositivo. En el dispositivo se puede implementar un modelo entrenado en ambientes de trabajo externos, el modelo debe contener la arquitectura de la red neuronal y los parámetros ajustados.

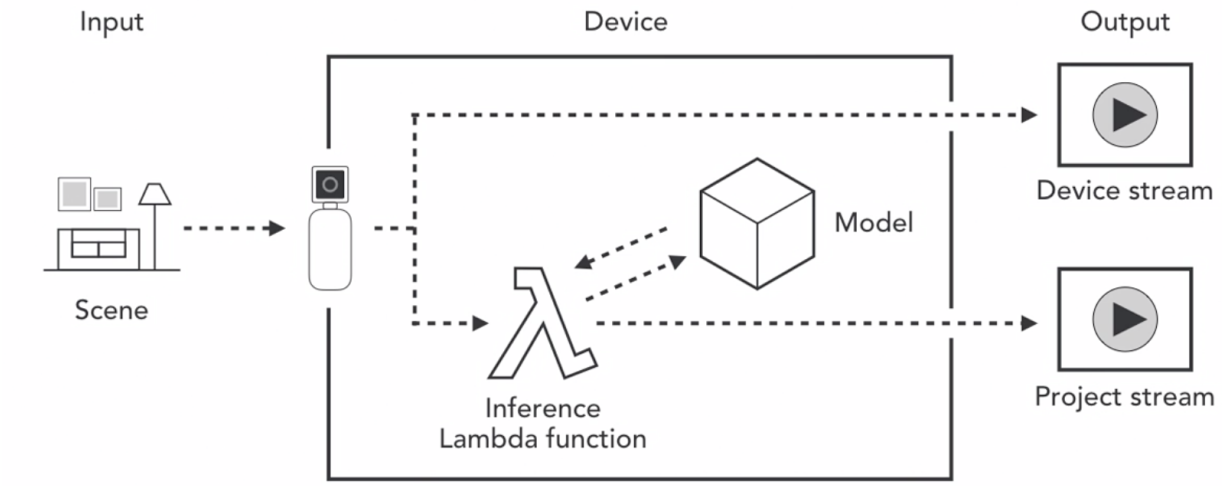


FIGURA 2.4. Procesamiento de la imagen con Deeplens

El dispositivo requiere conexión a internet cuando se despliega el modelo en la cámara y si alguna funcionalidad del modelo requiere conectividad. Si el modelo no requiere ningún tipo de conectividad entonces una vez desplegado el modelo, se puede visualizar el flujo del proyecto y los resultados de la inferencia en la cámara directamente.

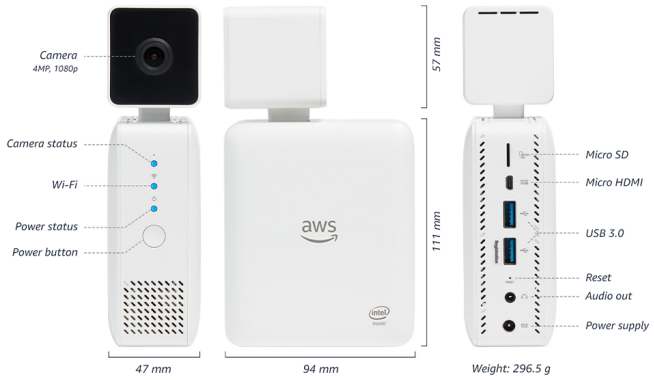


FIGURA 2.5. Camara Amazon DeepLens

AWS Deeplens utiliza los servicios siguientes:

- Amazon SageMaker, para entrenar y validar los modelos.
- AWS Lambda para realizar inferencias sobre modelos de redes neuronales convolucionales.
- AWS Greengrass para desplegar y distribuir los modelos en los dispositivos.

1.6.3. *Entrenamiento en AM (ML Training)*. Con todo el despliegue de tecnología y servicios Amazon ofrece diversos programas de entrenamiento y en particular programas de Aprendizaje Automática (Machine Learning) con el cual se pueden formar los desarrolladores y los científicos de datos. Se ofrecen más de 30 cursos en formato digital con prácticas y teoría.

Estos programas sirven a la comunidad en general y sirven para aprender a aplicar el aprendizaje automatizado, inteligencia artificial, Aprendizaje Profundo y aprendizaje reforzado. Tienen un enfoque aplicado y además un programa de certificación.

Capítulo 3

Marco teórico

1. Aprendizaje estadístico

El aprendizaje estádistico ha tomado relevancia en áreas como la ciencia, tecnología, medicina y finanzas. Hoy en día se pueden resolver problemas como:

- Predecir si un paciente sufrirá una enfermedad basados en sus datos demográficos y los resultados de exámenes.
- Reconocer objetos o patrones visuales en imágenes.
- Realizar traducciones de texto.
- Generar subtítulos de forma automatizada en un video.
- Programar agentes automatizados e interactivos para ofrecer ayuda a los clientes.
- Predecir el desempeño de una empresa basados en sus indicadores económicos.

Las técnicas de aprendizaje nos proporcionan métodos para estimar programas que resuelven algún problema sin ser explícitamente programados, es por ello que estas técnicas se utilizan en el campo de la Inteligencia Artificial. Se puede hablar de una forma nueva de programar, partimos de conjuntos de datos que proporcionan información sobre algún problema que se desea resolver y se entrenan algoritmos de aprendizaje que realizan un ciclo de estimación, cálculo del error o distancia entre el valor estimado y el valor observado y optimización para ajustar los parámetros de una función que minimiza la distancia entre el valor estimado y el observado la cual nos va a permitir inferir sobre datos nuevos.

Es decir, en lugar de programar la solución explícitamente vamos a aproximar una función que es capaz de llevar a cabo la actividad con un margen de error conocido en relación a los datos proporcionados.

En un escenario usual contamos con los datos, una medición como el precio de un activo financiero o datos categóricos sobre la presencia de una enfermedad o no, basándonos en las características propias de cada dato, deseamos realizar inferencias sobre datos nuevos. En el proceso de aprendizaje se utilizan características conocidas de los datos y se pueden

identificar o generar nuevas. Todas estas características proporcionan información que se utiliza para estimar el resultado y ajustar los parámetros del modelo hasta que logramos minimizar el error, sin sobreajustar o sesgar el modelo.

Las técnicas de aprendizaje se clasifican en supervisado y no supervisado. El aprendizaje supervisado consiste en ajustar modelos partiendo de datos que están etiquetados, es decir, que cada dato tiene asociado el valor de la característica que queremos predecir. El aprendizaje no supervisado consiste en proporcionar los datos y sus características, sin incluir o sin conocer las características que se desean predecir.

Los modelos de aprendizaje supervisado consisten en un proceso de entrenamiento que permite ajustar los parámetros del modelo mediante un proceso de optimización de una función que minimiza el error. Estos modelos plantean una distyuntiva entre el sesgo del modelo y el error aceptado. El sesgo indica que tan ajustado está el modelo a los datos observados y su capacidad para realizar inferencia sobre datos nuevos. En este caso los datos se dividen en dos conjuntos, uno de entrenamiento y otro de prueba, el primer conjunto de datos se utiliza para realizar el proceso de estimación y ajuste de parámetros y luego se prueba el modelo obtenido con el conjunto de datos de prueba. El aprendizaje se puede interpretar en este caso como el ajuste de parámetros de un modelo como resultado de un proceso de optimización que minimiza el error o maximiza los aciertos del modelo. Es un proceso de regresión, es decir, un proceso para conseguir una función que surge de los datos provistos. Vamos a seguir las definiciones realizadas en [15].

Sea X la variable de entrada que puede ser un valor o un vector de valores donde X_i representa la i-ésima entrada y lo denotamos en minúsculas x_i como el i-ésimo valor o dato observado que puede ser un escalar o un vector. En el caso que sea un p-vector este se denota como $x_i = [x_{i1}, ..., x_{ip}]$, donde x_{ij} denota la j-ésima característica y $j = 1, ..., p$. Los datos se pueden representar en una matríz $\mathbf{X}$ de dimensión $N \times p$, es decir, está confirmada por N p-vectores como se puede ver en 3.1,

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1p} \\ x_{21} & x_{22} & ... & x_{2p} \\ x_{31} & x_{32} & ... & x_{3p} \\ \vdots & ... & ... & \vdots \\ x_{N1} & x_{N2} & ... & x_{Np}. \end{bmatrix}_{\mathbf{N}\times\mathbf{p}} \tag{3.1}$$

El resultado provisto en los datos ya sea numérico lo denotamos con Y o cualitativo como G y los valores estimados como $\hat{Y}$ o $\hat{G}$.

En el modelo de aprendizaje supervisado asumimos que contamos con los datos de entrenamiento (x_i, y_i) o (x_i, g_i) donde $i = 1, ...N$.

A partir de estos datos vamos a contruir un modelo estadístico, primero vamos a considerar el caso de un resultado numérico y consideramos a X como un vector aleatorio tal que $X \in \mathcal{R}^p$ y $Y \in \mathcal{R}$ una variable aleatoria real. Estamos interesados en conseguir una función $f(X)$ para predecir Y dados los valores de entrada X.

Nos interesa realizar un modelo capaz de predecir basándonos en una muestra de entrenamiento $(x_1, y_1), ...(x_N, y_N)$. Es decir, el modelo produce $\hat{y}_i$ estimado para cada x_i y el proceso de aprendizaje consiste en minimizar una función de pérdida $L(y, \hat{y})$, es decir, minimizar la distancia entre el valor estimado y el valor observado. Si suponemos que (X, Y) son variables aleatorias representadas por una función de densidad conjunta $P(X, Y)$, entonces el aprendizaje supervisado se puede caracterizar como la estimación de una función de densidad, donde interesa conocer las propiedades de $P(Y \mid X)$. En este caso nos interesa conseguir los parámetros μ tal que para cada x:

$$\mu(x) = argmin_\theta E_{Y|X} L(Y, \hat{Y}(\theta))$$

Partiendo de las variables independientes, que son las características observadas, queremos estimar la variable dependiente mediante un modelo que minimize el error. En un modelo de regresión donde nos interesa estimar el precio de una acción entonces el valor a estimar es un número real o una cantidad, $y \in \mathbb{R}$. En un modelo de clasificación que nos indica cual dígito escrito a mano corresponde con una imagen vamos a ver que $y \in \{0, 1, ..., 9\}$, o en general $y \in \{g_1, .., g_k\}$ donde g_i representa la clase i a la cual pertenece el dato. Las clases también son denominadas categorías, variables discretas o factores. El resultado de un modelo de clasificación es una medida cualitativa, tanto la

regresión como la clasificación tienen muchos elementos comunes, en ambos casos estamos aproximando una función estimadora de error mínimo.

Desde el punto de vista del aprendizaje automatizado podemos suponer por simplicidad que los errores son aditivos y que el modelo aleatorio $Y = f(X)+\epsilon$ es un modelo razonable. El aprendizaje supervisado estima f mediante un proceso de estimación y ajuste. Partimos de los datos de entrenamiento $(x_i, y_i), i = 1, ..., N$. Se proporciona x_i como entrada al proceso de estimación que genera como resultado $\hat{y}_i$ y luego se realiza un ajuste de la estimación basándonos en la relación entre el valor estimado y el valor observado, es decir, se calcula el error o distancia $(y_i - \hat{y}_i)$ entre y_i y $\hat{y}_i = \hat{f}(x_i)$. Este es el proceso de aprendizaje donde se espera obtener una función de predicción que genera resultados suficientemente cercanos a los valores observados en los datos.

El proceso de aprendizaje previo ha sido la motivación que guía la investigación de las técnicas de aprendizaje supervisado como el aprendizaje automatizado, basados en el razonamiento humano y la redes neuronales con analogías al cerebro humano. El acercamiento desde el punto de vista de la matemática y estadísticas es desde la perspectiva de la aproximación y estimación de funciones.

Por otra parte el aprendizaje no supervisado consiste en trabajar con los datos, sin realizar ningún proceso de ajuste de parámetros a un resultado esperado. Sólo contamos con los datos X y nos interesa conseguir relaciones entre estos datos que hagan emerger nuevas características. Es decir, para N observaciones $(x_1, ..., x_N)$ de un vector aleatorio X con densidad conjunta $P(X)$, el objetivo es inferir propiedades sin información adicional.

1.1. Métricas de desempeño. Para entender el alcance y efectividad de un método de aprendizaje es necesario establecer las métricas que indicarán si el desempeño es el esperado. En los métodos de clasificación se desea contrastar el resultado inferido contra el resultado observado. Con esto en mente se realiza la división del conjunto de datos en datos de entrenamiento y datos de prueba. Se utilizan los datos de entrenamiento para estimar y ajustar el modelo y luego se utiliza el modelo ajustado para realizar inferencias sobre los datos de prueba.

En el caso de un modelo de clasificación binario o dicotómico, primero vamos a colocar en una tabla los valores observados y los inferidos como en el cuadro 1. Tenemos cuatro casos:

(1) Valores observados positivos y se infieren positivos, los denominamos vp por verdaderos positivos.
(2) Valores observados negativos y se infieren positivos, los denominamos fp por falsos positivos.
(3) Valores observados positivos y se infieren negativos, los denominamos fn por falsos negativos.
(4) Valores observados negativos y se infieren negativos, los denominamos vn por verdaderos negativos.

		Observado	
	Población	Positivos	Negativos
Inferidos	Inferidos positivos	vp	fp
	Inferidos negativos	fn	vn

CUADRO 1. Matriz de confusión

A partir de esta matriz vamos a definir las métricas siguientes para medir la eficacia del modelo de inferencia.

1.1.1. *Error.* Es la proporción de la suma entre los falsos positivos y falsos negativos y los datos, el cálculo se realiza con la fórmula siguiente:

$$\text{Error} = \frac{fp + fn}{\text{Población}}. \tag{3.2}$$

1.1.2. *Exactitud.* Es la proporción de la suma entre los verdaderos positivos y los verdaderos negativos y todos los datos y lo calculamos con la fórmula siguiente:

$$\text{Exactitud} = \frac{vp + vn}{\text{Población}}. \tag{3.3}$$

1.1.3. *Precisión.* Es la proporción entre los verdaderos positivos y todos los inferidos como positivos y la fórmula es la siguiente:

$$\text{Precisión} = \frac{vp}{vp + fp}. \tag{3.4}$$

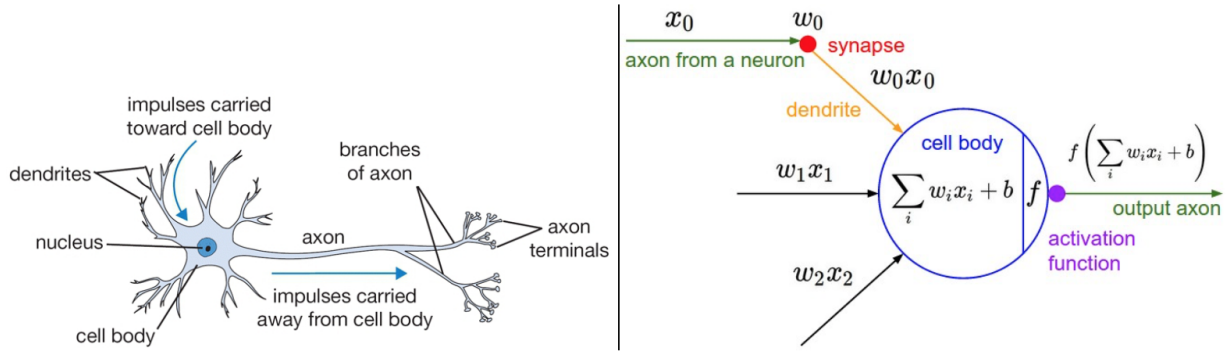
1.1.4. *Sensibilidad - Recall.* Es la proporción entre los verdaderos positivos y la suma entre los verdaderos positivos y los falsos negativos, la fórmula es como sigue:

$$\text{Sensibilidad} = \frac{vp}{vp + fn}. \tag{3.5}$$

2. Redes Neuronales

2.1. Modelo biológico. Las Redes Neuronales engloban una familia de modelos de aprendizaje supervisado que parte de un modelo del funcionamiento del cerebro biológico. También se conocen como modelos coneccionistas ya que consisten en la conexión de neuronas en configuración de redes con múltiples capas.

Las Redes Neuronales son un modelo de regresión no lineal, con respuestas simples o de clasificación, es decir con respuestás múltiples, que consiste en una red de nodos, denominados neuronas. Dichas neuronas modelan el funcionamiento de las neuronas cerebrales. Las neuronas reciben información de las capas previas y aplican una función de activación para modelar respuestas no lineales. Las redes incluyen varias capas que consisten en varias neuronas que representan nuevas características (features) sobre los datos. Es un modelo de aprendizaje supervisado que recibe un conjunto de datos etiquetados con los cuales se puede contrastar las respuestas del modelo para realizar un proceso de ajuste que minimice la distancia entre el valor estimado y los valores observados.



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

FIGURA 3.1. Modelo matemático de una neurona biológica

En el modelo biológico de una reurona, que se puede visualizar en la figura 3.1, partimos de las entradas $X = (x_1, .., x_p)$ y los pesos o variables $W = (w_1, .., w_p)$, se realiza la combinación lineal entre W y X, $W^T X$ y al resultado de la combinación lineal le sumamos el sesgo, quedando $\sum_i w_i x_i + b$ y luego aplicamos una función de activación f que genera el resultado de la neurona, $f(\sum_i w_i x_i + b)$. Los axones generan las entradas y las dendritas le dan importancia a dicho axón a través de las variables w_i, en el núcleo de la neurona se combinan las entradas

con las variables y se agrega el sesgo y luego se aplica la función de activación que genera los valores de salida del axón de la neurona. Como se puede apreciar la salida de una neurona se puede conectar con la entrada de otra neurona y por ello es posible configurarlas en redes interconectadas.

2.2. Redes Neuronales Multicapa. Vamos a revisar el modelo de estimación y ajuste de las Redes Neuronales Multicapa desarrollando en detalle el ejemplo proporcionado por Hastie et al en [15]. Vamos a revisar los cálculos para realizar la etapa de estimación también conocido como paso de alimentación hacia adelante (feed forward) y luego el proceso de ajuste de parámetros tambien conocido como propagación (de errores) hacia atrás (back propagation). Esta terminología surge del hecho de que las Redes Neuronales en este caso son un grafo dirigido acíclico, es decir, los nodos están conectados con las siguientes capas sólamente.

2.2.1. *Estimación.* Partiendo del modelo especificado en [15] vamos a desarrollar el modelo de una Red Neuronal de una capa oculta siguiendo el diagrama 3.2, vamos a plantear los cálculos y las formulas de estimación y ajuste de la red.

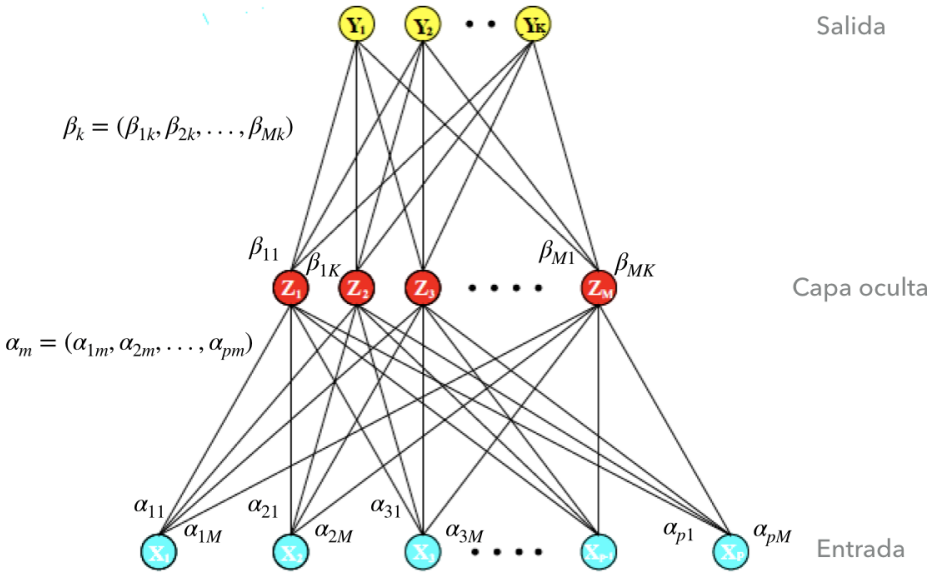


FIGURA 3.2. Red Neuronal Multicapa

Para clasificar en K-clases tenemos una capa de salida de K unidades, donde la K-ésima unidad modela la probabilidad de que las entradas X pertenecen a la clase K. Es por ello

que hay K medidas, cada una codificada como variables $0-1$ para la K-ésima clase. Las características derivadas son una combinación lineal de los datos de entrada y los pesos α_{ij}.

Supongamos que tenemos una red neuronal que consta de una capa oculta con M neuronas y que hay K objetivos de clasificación o funciones finales. El vector de entrada lo llamaremos X, es un vector con p componentes, $X = [X_1\ X_2\ ...\ X_p]^T$, cuando estamos tratando con los datos se denota en minúsculas.

Las características o neuronas en la capa oculta son creadas como la aplicación de una función de activación a una combinación lineal de las entradas, llamaremos a estas $Z_1, Z_2, ..., Z_M$, a partir de estas se calculan las salidas $Y_1, ..., Y_K$ que son el resultado de aplicar una función de activación a una combinación lineal de las salidas de la capa oculta $Z_1, Z_2, ..., Z_M$.

La función de activación que usaremos en las capas ocultas la denotaremos por $\sigma(\nu)$ y para la salida se aplica una función $g_k(T)$.

Comencemos definiendo las funciones que están involucradas en la capa oculta. Cada neurona de la capa oculta tendrá la siguiente representación,

$$\begin{aligned}
Z_1 &= \sigma(\alpha_{01}b_1 + \alpha_{11}x_1 + \alpha_{21}x_2 + \alpha_{31}x_3 + ... + \alpha_{p1}x_p)\\
Z_2 &= \sigma(\alpha_{02}b_1 + \alpha_{12}x_1 + \alpha_{22}x_2 + \alpha_{32}x_3 + ... + \alpha_{p2}x_p)\\
Z_3 &= \sigma(\alpha_{03}b_1 + \alpha_{13}x_1 + \alpha_{23}x_2 + \alpha_{33}x_3 + ... + \alpha_{p3}x_p)\\
\vdots &= \quad\vdots\\
Z_M &= \sigma(\alpha_{0M}b_1 + \alpha_{1M}x_1 + \alpha_{2M}x_2 + \alpha_{3M}x_3 + ... + \alpha_{pM}x_p),
\end{aligned}$$

donde α_{ij} denota el parámetro que corresponde a la entrada i y llega a la neurona j de la capa oculta.

Adicionalmente supongamos de entrada que $b_1 = 1$, así nos queda,

$$\begin{aligned}
Z_1 &= \sigma(\alpha_{01} + \alpha_{11}x_1 + \alpha_{21}x_2 + \alpha_{31}x_3 + ... + \alpha_{p1}x_p)\\
Z_2 &= \sigma(\alpha_{02} + \alpha_{12}x_1 + \alpha_{22}x_2 + \alpha_{32}x_3 + ... + \alpha_{p2}x_p)\\
Z_3 &= \sigma(\alpha_{03} + \alpha_{13}x_1 + \alpha_{23}x_2 + \alpha_{33}x_3 + ... + \alpha_{p3}x_p)\\
\vdots &= \quad\vdots\\
Z_M &= \sigma(\alpha_{0M} + \alpha_{1M}x_1 + \alpha_{2M}x_2 + \alpha_{3M}x_3 + ... + \alpha_{pM}x_p).
\end{aligned}$$

Utilizando la notación vectorial y tomando una neurona fija Z_m con $1 \leq m \leq M$, los parámetros correspondientes se encuentran en el vector fila $\alpha_m^T = [\alpha_{1m}\ \alpha_{2m}\ ...\ \alpha_{pm}]$, donde el vector de entrada es $X = [x_1\ x_2\ ...\ x_p]^T$, el producto entre el vector α_m^T y X queda,

$$\begin{aligned}\alpha_m^T X &= \begin{bmatrix}\alpha_{1m} & \alpha_{2m} & ... & \alpha_{pm}\end{bmatrix}\begin{bmatrix}x_1\\x_2\\\vdots\\x_p\end{bmatrix}\\ \alpha_m^T X &= \alpha_{1m}x_1 + \alpha_{2m}x_2 + \alpha_{3m}x_3 + ... + \alpha_{pm}x_p = \sum_{i=1}^{p}\alpha_{im}x_i.\end{aligned}$$

Utilizando esta notación la neurona Z_m queda de la siguiente manera,

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X).$$

Así cada neurona Z_m con $m = 1, ..., M$ nos queda,

$$\begin{aligned}Z_1 &= \sigma(\alpha_{01} + \alpha_1^T X)\\ Z_2 &= \sigma(\alpha_{02} + \alpha_2^T X)\\ Z_3 &= \sigma(\alpha_{03} + \alpha_3^T X)\\ \vdots &= \quad \vdots\\ Z_M &= \sigma(\alpha_{0M} + \alpha_M^T X).\end{aligned}$$

Para esta primera parte de la red neuronal se tienen que estimar $(p+1)\times M$ parámetros.

Luego con estas salidas se generan los valores que se utilizan en la última capa, $Y_1, Y_2, ..., Y_K$. A la combinanción lineal entre las salidas de las neuronas $Z_1, ..., Z_M$ y los parámetros β_{mk}, le aplicamos una función de salida g_k, es decir,

$$\begin{aligned}Y_1 &= g_k(\beta_{01}b_2 + \beta_{11}Z_1 + \beta_{21}Z_2 + \beta_{31}Z_3 + ... + \beta_{M1}Z_M)\\ Y_2 &= g_k(\beta_{02}b_2 + \beta_{12}Z_1 + \beta_{22}Z_2 + \beta_{32}Z_3 + ... + \beta_{M2}Z_M)\\ Y_3 &= g_k(\beta_{03}b_2 + \beta_{13}Z_1 + \beta_{23}Z_2 + \beta_{33}Z_3 + ... + \beta_{M3}Z_M)\\ \vdots &= \quad \vdots\\ Y_K &= g_k(\beta_{0K}b_2 + \beta_{1K}Z_1 + \beta_{2K}Z_2 + \beta_{3K}Z_3 + ... + \beta_{MK}Z_M)\end{aligned}$$

Si $b_2 = 1$, nos queda,

$$\begin{aligned}Y_1 &= g_k(\beta_{01} + \beta_{11}Z_1 + \beta_{21}Z_2 + \beta_{31}Z_3 + ... + \beta_{M1}Z_M)\\ Y_2 &= g_k(\beta_{02} + \beta_{12}Z_1 + \beta_{22}Z_2 + \beta_{32}Z_3 + ... + \beta_{M2}Z_M)\\ Y_3 &= g_k(\beta_{03} + \beta_{13}Z_1 + \beta_{23}Z_2 + \beta_{33}Z_3 + ... + \beta_{M3}Z_M)\\ \vdots &= \quad \vdots\\ Y_K &= g_k(\beta_{0K} + \beta_{1K}Z_1 + \beta_{2K}Z_2 + \beta_{3K}Z_3 + ... + \beta_{MK}Z_M)\end{aligned}$$

Ahora, para un k fijo, con $1 \leq k \leq K$, sea Z el vector de neuronas de la capa oculta, el producto entre $Z = [Z_1\ Z_2\ ...\ Z_M]^T$ y el vector de parámetros $\beta_k^T = [\beta_{1k}\ \beta_{2k}\ \beta_{3k}\ ...\ \beta_{Mk}]$ nos queda,

$$\begin{aligned}\beta_k^T Z &= \begin{bmatrix}\beta_{1k} & \beta_{2k} & ... & \beta_{Mk}\end{bmatrix}\begin{bmatrix}Z_1\\ Z_2\\ \vdots\\ Z_M\end{bmatrix}\\ \beta_k^T Z &= \beta_{1k}Z_1 + \beta_{2k}Z_2 + \beta_{3k}Z_3 + ... + \beta_{Mk}Z_M\end{aligned}$$

Así, las salidas Y_k quedan de la forma siguiente,

$$\begin{aligned}Y_k &= g_k(\beta_{0k} + \beta_{1k}Z_1 + \beta_{2k}Z_2 + \beta_{3k}Z_3 + ... + \beta_{Mk}Z_M)\\ &= g_k(\beta_{0k} + \beta_k^T Z) = g_k(T_k),\end{aligned}$$

donde $T_k = \beta_{0k} + \beta_k^T Z$. Utilizando esta notación tenemos,

$$\begin{aligned}Y_1 &= g_k(\beta_{01} + \beta_1^T Z) = g_k(T_1)\\ Y_2 &= g_k(\beta_{02} + \beta_2^T Z) = g_k(T_2)\\ Y_3 &= g_l(\beta_{03} + \beta_3^T Z) = g_k(T_3)\\ \vdots &= \quad\vdots\\ Y_K &= g_k(\beta_{0K} + \beta_K^T Z) = g_k(T_K).\end{aligned} \tag{3.6}$$

En esta última capa tenemos que estimar $(M+1) \times K$ parámetros. Para entender un poco la cantidad de parámetros se que tienen que estimar veamos las matrices de parámetros en cada capa.

- Parámetros para la primera capa 3.7,

$$\begin{bmatrix}\alpha_{01} & \alpha_{11} & \alpha_{21} & ... & \alpha_{p1}\\ \alpha_{02} & \alpha_{12} & \alpha_{22} & ... & \alpha_{p2}\\ \vdots & \vdots & \vdots & & \vdots\\ \alpha_{0M} & \alpha_{1M} & \alpha_{2M} & ... & \alpha_{pM}\end{bmatrix}^T_{M\times(p+1)} \tag{3.7}$$

- Parámetros para la segunda capa 3.8,

$$\begin{bmatrix} \beta_{01} & \beta_{11} & \beta_{21} & \dots & \beta_{M1} \\ \beta_{02} & \beta_{12} & \beta_{22} & \dots & \beta_{M2} \\ \vdots & \vdots & \vdots & & \vdots \\ \beta_{0K} & \beta_{1K} & \beta_{2K} & \dots & \beta_{MK} \end{bmatrix}^T_{K\times(M+1)} \tag{3.8}$$

Los resultados $Y = [Y_1\ Y_2\ ...\ Y_K]^T$ proporcionan el resultado de la estimación que genera la Red Neuronal. En este paso se proporcionan las entradas X y los parámetros α_{im}, α_{0m}, β_{mk} y β_{0k}, con $i = 1, ..., p$, $m = 1, ..., M$ y $k = 1, ..., K$. Cada Y_k queda expresado de la forma siguiente,

$$Y_k = f_k(X) = g_k(T_k) = g_k(\beta_{0k} + \sum_{m=1}^{M} \beta_{mk}\ \sigma(\alpha_{0m} + \sum_{i=1}^{p} \alpha_{im}x_i)). \tag{3.9}$$

2.2.2. *Pérdida o Error.* El proceso de aprendizaje o ajuste de los parámetros de la Red Neuronal consiste en dos pasos fundamentales, el primero es el cálculo del error o distancia entre la estimación que produce la Red Neuronal y el valor observado y el segundo es el ajuste de los parámetros mediante el proceso de optimización que minimiza el error.

Como hemos visto la Red Neuronal cuenta con los parámetros definidos previamente, tambien conocidos como pesos,

$$\{\alpha_{0m}, \alpha_{im}, i = 1, ..., p, m = 1, ..., M\} \quad (p+1) \times M \text{ pesos},$$
$$\{\beta_{0k}, \beta_{mk}, k = 1, ..., K\} \quad (M+1) \times K \text{ pesos}.$$

Si se desea realizar una regresión se suele utilizar la suma del cuadrado de los errores, donde se calcula el cuadrado de la diferencia entre el valor observado y el valor estimado por la red. La fórmula queda como sigue,

$$L = \sum_{i=1}^{N} L_i = \sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} - f_k(x_i))^2. \tag{3.10}$$

Notemos que N es el tamaño de la muestra y x_i es un p-vector correspondiente al i-ésimo dato de entrenamiento. Cabe destacar que en este caso estamos considerando la suma de los errores, cuando el proceso de ajuste se realiza de esta forma se denomina aprendizaje por lotes (batch learning), cuando se considera sólo un dato se denomina aprendizaje en línea

(online learning). Hay otras variantes que consideran mini lotes o subconjuntos de los datos para realizar el cálculo del error y realizar el proceso de ajuste.

Para clasificar se puede utilizar el cuadrado de los errores o la devianza (cross entropy),

$$L = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i), \tag{3.11}$$

y el clasificador sería $G(x) = argmax_k f_k(x_i)$. Con la función de activación softmax 3.21 y la función de devianza para generar la estimación del error, la Red Neuronal es exáctamente un modelo de regresión logística en las capas ocultas y los parámetros se estiman mediante el método de máxima verosimilitud.

2.2.3. *Aprendizaje.* El método que usaremos para ajustar los parámetros es el método iterativo de optimización que encuentra el mínimo de la función de pérdida o error mediante el ajuste de los parámetros considerando el recálculo del gradiente, también conocido como gradiente descendente o descenso del gradiente.

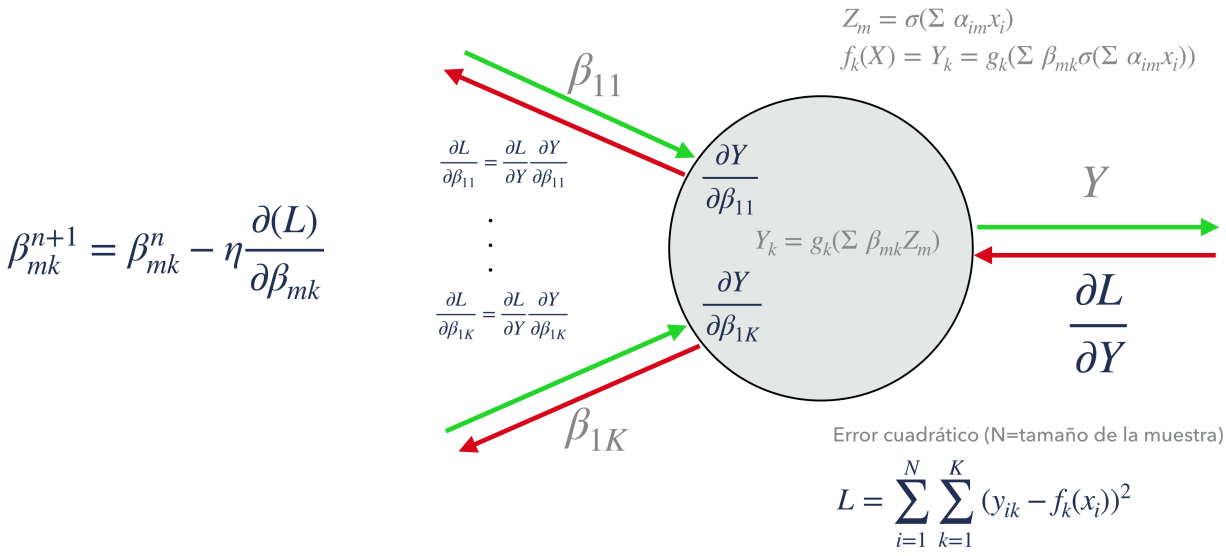


FIGURA 3.3. Aprendizaje y ajuste de la Red Neuronal

No se desea encontrar un mínimo global ya que esto puede producir un conjunto de parámetros sobreajustados, es decir, que vamos a utilizar una condición de parada para definir un nivel aceptable.

El algoritmo del gradiente descendente es iterativo, es decir, el nuevo valor del parámetro se calcula en base valor previo, fíjese en la figura 3.4. El gradiente descendente

realiza un proceso de optimización para hallar el mínimo de la función utilizando la dirección del gradiente y una tasa de aprendizaje dada.

Dadas las derivadas y η la tasa de aprendizaje, los parámetros α_{im} y β_{mk}, se ajustan iterativamente, suponiendo que estamos en la iteración $(n+1)$ el ajuste queda,

$$\begin{aligned}\beta_{mk}^{(n+1)} &= \beta_{mk}^{(n)} - \eta \frac{\partial L_i}{\partial \beta_{mk}},\\ \alpha_{im}^{(n+1)} &= \alpha_{im}^{(n)} - \eta \frac{\partial L_i}{\partial \alpha_{im}}\end{aligned}$$

Recordemos que tenemos un p-vector de datos de entrada $x_i = [x_{1i}\ x_{2i}\ ...\ x_{pi}]^T$ y para este tenemos el resultado observado $y_i = [y_{1i}\ y_{2i}\ ...\ y_{Ki}]^T$ entonces queremos medir las diferencias de este vector con la estimación generada por la Red Neuronal.

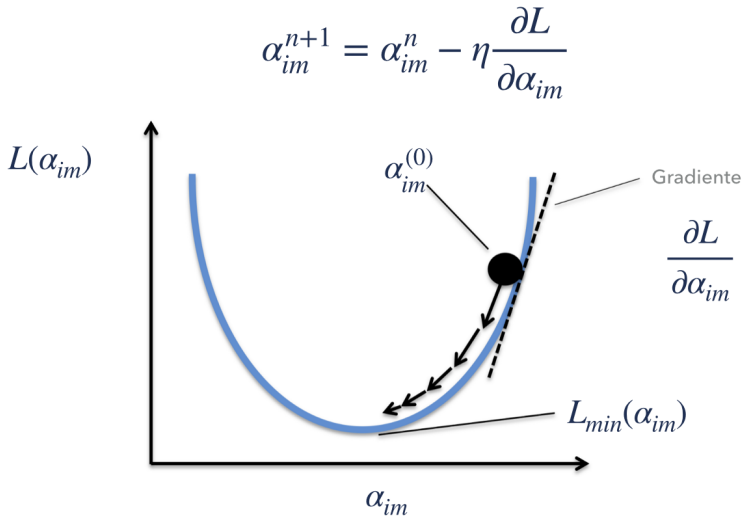


FIGURA 3.4. Optimización iterativa con saltos en la dirección del gradiente

Partiendo de los valores estimados en 3.9 y la función de pérdida definida en 3.10 vamos a calcular las derivadas parciales de la función de pérdida en relación a cada parámetro aplicando la regla de la cadena,

$$\frac{\partial L}{\partial \beta_{mk}} = \frac{\partial \left(\sum_{i=1}^{N} L_i\right)}{\partial \beta_{mk}} = \sum_{i=1}^{N} \frac{\partial L_i}{\partial \beta_{mk}}, \tag{3.12}$$

$$\frac{\partial L}{\partial \alpha_{im}} = \frac{\partial \left(\sum_{i=1}^{N} L_i\right)}{\partial \alpha_{im}} = \sum_{i=1}^{N} \frac{\partial L_i}{\partial \alpha_{im}}. \tag{3.13}$$

Las derivadas parciales de la función de pérdida con respecto a los parámetros queda de la forma siguiente,

$$\begin{aligned}\frac{\partial L_i}{\partial \beta_{mk}} &= -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi},\\ \frac{\partial L_i}{\partial \alpha_{im}} &= -\sum_{k=1}^{K}(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{mk}\sigma'(\alpha_m^T x_i)x_{im}\end{aligned}$$

Si reescribimos la derivada de la función de pérdida de la forma siguiente,

$$\begin{aligned}\frac{\partial L_i}{\partial \beta_{mk}} &= \delta_{ki}\ z_{mi},\\ \frac{\partial L_i}{\partial \alpha_{im}} &= s_{mi}\ x_{im},\end{aligned}$$

donde δ_{ki} y s_{mi} son los errores de la capa de resultado y capa oculta y s_{mi} es tal que,

$$s_{mi} = \sigma'(\alpha_m^T x_i)\sum_{k=1}^{K}\beta_{km}\delta_{ki}.$$

Estas son las formulas de ajuste de la Red Neuronal y se implementan en dos fases, la primera de estimación y la segunda de ajuste o aprendizaje. En la segunda fase las derivadas nos sirven para calcular el gradiente y ajustar los parámetros. Este proceso de ajuste se conoce como propagación reversa (back propagation).

El proceso de aprendizaje se realiza por iteraciones sobre todos los datos y estos se toman en lotes para realizar las fases de estimación y ajuste, estas iteraciones sobre todo los datos se denominan épicas (epochs).

La tasa de aprendizaje usualmente es constante y se puede ajustar para minimizar el error en cada actualización/ajuste de los parámetros.

2.3. Funciones de activación. El modelos de Redes Neuronales es no lineal debido al empleo de funciones de activación como se observan en 3.5. Vamos a revisar las más comunes.

2.3.1. *Función de paso 0 - 1.*

$$f(x) = \begin{cases} 0, & \text{si } x < 0,\\ 1, & \text{si } x \geq 0.\end{cases} \tag{3.14}$$

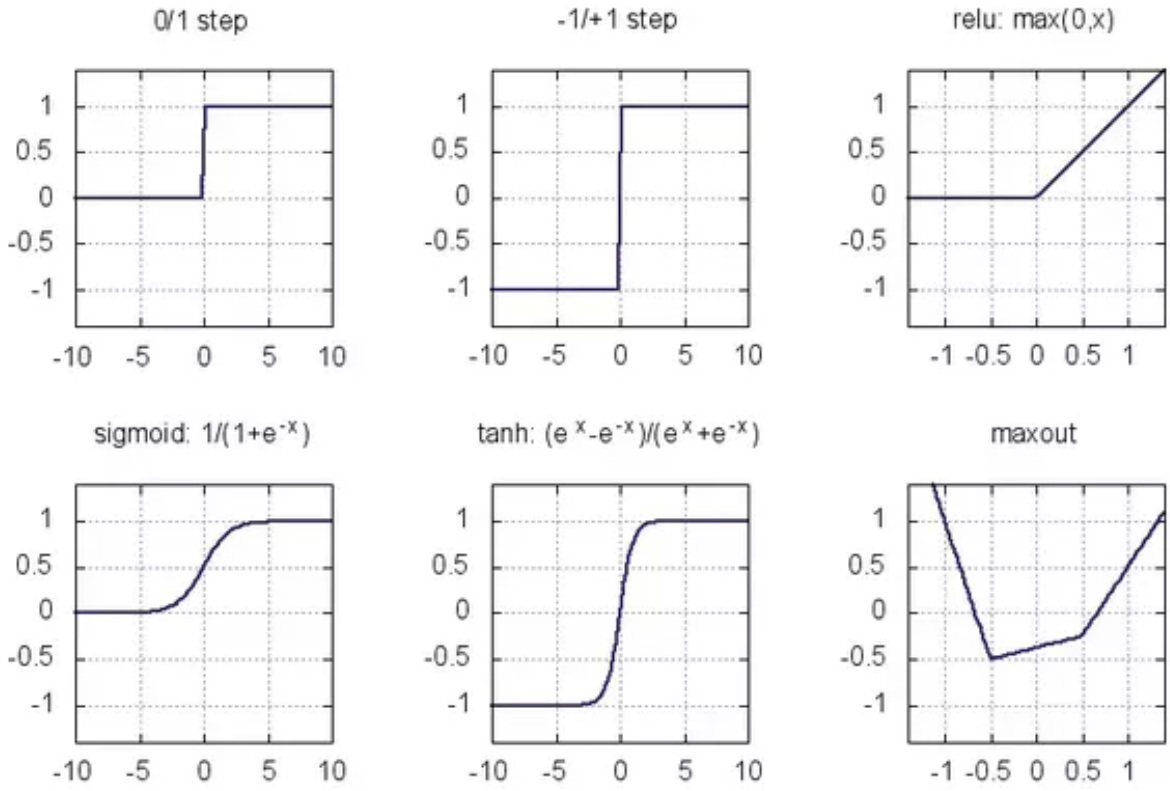


FIGURA 3.5. Funciones de activación

2.3.2. *Función de paso -1 - 1.*

$$f(x) = \begin{cases} -1, & \text{si } x < 0, \\ 1, & \text{si } x \geq 0. \end{cases} \tag{3.15}$$

2.3.3. *Sigmoide.* Es la más popular en términos académicos ya que se utilizó en las primeras investigaciones sobre Redes Neuronales.

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.16}$$

2.3.4. *Rectificador lineal (ReLU).*

$$f(x) = max(0, x) = \begin{cases} 0, & \text{si } x \leq 0, \\ x, & \text{si } x > 0. \end{cases} \tag{3.17}$$

2.3.5. *Rectificador lineal con filtración (Leaky ReLU).*

$$f(x) = max(0{,}1x, x) = \begin{cases} 0{,}1x, & \text{si } x \leq 0, \\ x, & \text{si } x > 0. \end{cases} \tag{3.18}$$

2.3.6. *Tangente hiperbólica.*

$$f(x) = tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{3.19}$$

2.3.7. *Maxout.*

$$f(\vec{x}) = \max_i x_i = \max\left(w_1^T x + b_1, w_2^T x + b_2, ...\right) \tag{3.20}$$

2.3.8. *Softmax.* Utilizada como función de salida para generar probabilidades de clase cuando el propósito de la Red Neuronal es clasificar.

$$g_k(T_k) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}} \tag{3.21}$$

2.4. Aprendizaje Profundo. El Aprendizaje Profundo consiste en construir arquitecturas de Redes Neuronales con múltiples capas ocultas para generar modelos de regresión o clasificación con datos más complejos. En las capas ocultas se aprenden representaciones de los datos que se expresan en términos de otras representaciones más simples, esto nos permite reconocer conceptos complejos en términos de otros conceptos más simples. En el gráfico 3.6 se puede apreciar como se representa un concepto de una persona en una imagen, combinando conceptos simples como las esquinas y contornos que a la vez se expresan en términos de los bordes.

El ejemplo más importante de un modelo de Aprendizaje Profundo es una Red Neuronal Multicapa con alimentación hacia adelante, es decir, una red de perceptrones multicapa. El perceptrón multicapa es una función matemática que relaciona un conjunto de valores de entrada a otro de salida. Esta función se compone de varias funciones más simples. Cada perceptrón ofrece una nueva representación de la entrada. El modelo genera representaciones de los datos que sirven para realizar inferencias, como se puede ver en la imagen 3.7, el proceso consiste en aprender nuevas características de los datos. LeCun et al. en [6] señalan que,

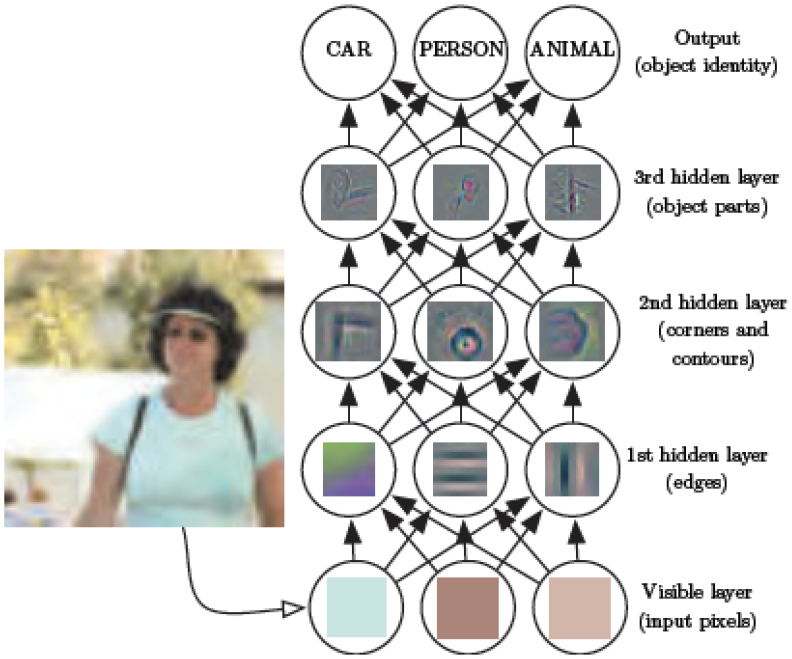


FIGURA 3.6. Aprendizaje Profundo

las Redes Neuronales Multicapa entrenadas con el algoritmo de propagación reversa constituye el mejor ejemplo de una técnica de aprendizaje basada en gradientes. Dada la arquitectura apropiada, los algoritmos basados en aprendizaje basado en gradientes se pueden utilizar para simplificar una superficie de decisión compleja que puede clasificar patrones en varias dimensiones tales como los caracteres escritos a mano con bajo procesamiento... Las Redes Neuronales Convolucionales que están diseñadas específicamente para tratar con la variabilidad de los objetos en dos dimensiones, han mostrado que superan todas las otras técnicas de aprendizaje.

2.4.1. *LeNet.* En [16] Lecun et al, señalan que es esencial extraer las características apropiadas en el diseño de sistemas de reconocimiento de objetos. Luego señala que para reconocer objetos con alta variabilidad, como los dígitos escritos a mano, es preferible y ventajoso alimentar al sistema con datos sin mucho procesamiento y permitir que un proceso de aprendizaje mediante Redes Neuronales Convolucionales extraiga el conjunto apropiado de características.

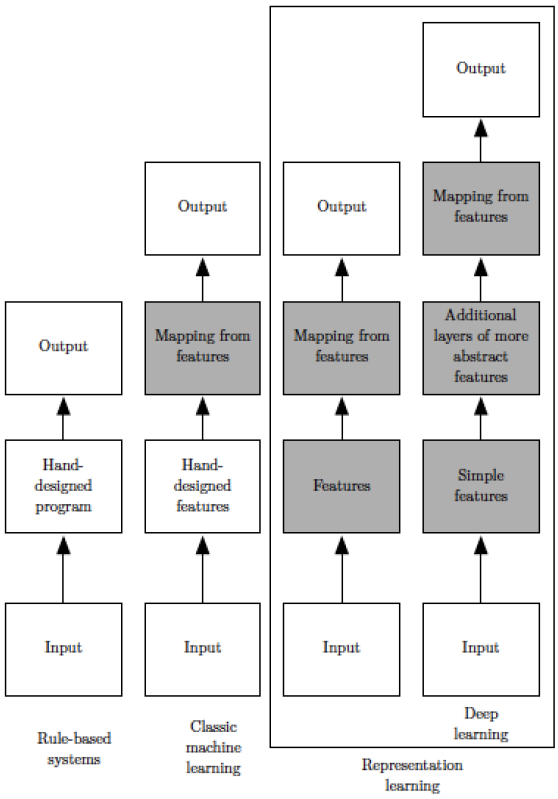


FIGURA 3.7. Aprendizaje de características (Feature learning)

Este es uno de los principales trabajos de investigación que crea una red multicapa para el reconocimiento de patrones visuales.

A continuación vamos a describir el desarrollo de métodos de visión por computadora y en particular las Redes Neuronales Convolucionales.

3. Visión por Computadora

En las ciencias de la computación hay una rama que se dedica a procesar imágenes digitales para obtener información de interés. Es un área multidisciplinada que utiliza técnicas de la matemática, estadística y la computación para extraer información de imágenes digitales de forma automatizada. Como plantea Shanmugamani en [17] el objetivo es desarrollar técnicas que imiten las capacidades de la visión humana en una computadora.

Las actividades de la visión por computadora son: adquisición, procesamiento, análisis y entendimiento de imágenes digitales. Entendimiento en este caso se refiere a la descripción del mundo, los objetos y sus relaciones a partir de las imágenes digitales.

Existen diversos sensores que producen imágenes que pueden ser interpretadas por una computadora mediante la geometría, física, estadistica y aprendizaje se pueden contruir modelos para realizar las tareas de la visión por computadora. Una de las técnicas que permite la extracción de información de imáges es el Aprendizaje Profundo.

Las imágenes se pueden generar mediante procesos diversos como videos, secuencias de imágenes, cámaras en distintos ángulos, datos de un tomógrafo, ecosonograma, entre otros.

Algunos de los problemas de la visión por computadora son:

- Reconocimiento de objetos,
- Clasificación de imágenes,
- Detección e identificación de rostros,
- Estimación de posturas,
- Detección de eventos.

Para lograr el entendimiento automático de imágenes se requiere el desarrollo de teorías y algoritmos que extraigan información y estructuren un modelo de los objetos y el mundo. Esta área se desarrolló a la par con el desarrollo de las computadoras desde 1960 y se pensaba que en poco tiempo ya contaríamos con agentes (robots) con capacidad para describir e interactuar con el mundo.

Recientemente se ha visto el desarrollo y resurgimiento de los métodos basados en las técnicas de aprendizaje y optimización, en particular el desarrollo de las Redes Neuronales Profundas, que han logrado resultados que superan los métodos previos en tareas como la clasificación, entre otros.

En el campo de la Inteligencia Artificial se han desarrollado métodos para el reconocimiento de patrones y la navegación autónoma de autos y robots. Para este propósito se requiere un entendimiento del entorno para navegarlo.

3.1. ¿Qué es una imagen digital? Una imagen está compuesta por pixeles que representan un punto indivisible en la pantalla. Cada punto en la imagen está representado por números que conforman el color y la transparencia. El color se representa como una combinación de colores primarios (Rojo, Verde y Azul) o en escala de grises como una valor entre blanco y negro.

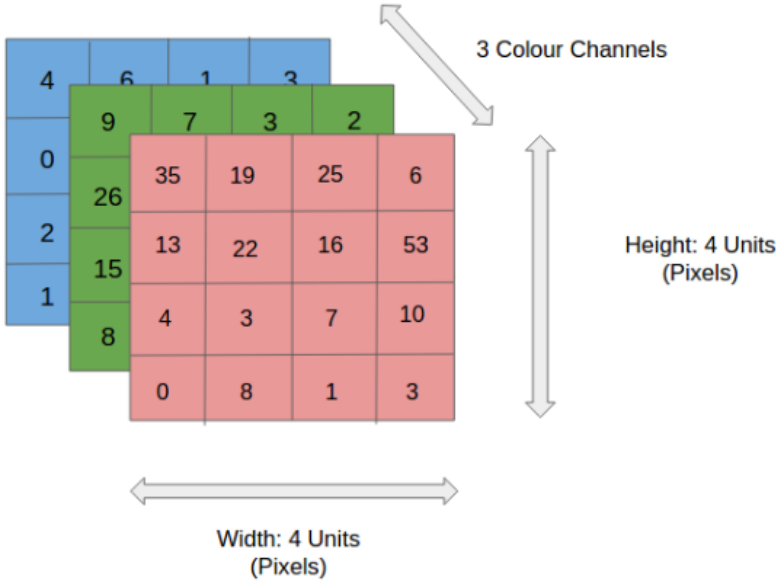


FIGURA 3.8. Características de una imagen digital

Hay diversos formatos de imágenes que principalmente ofrecen mecanismos para comprimir la información sin pérdida significativa de la calidad de la imagen. En 3.8 se puede apreciar una representación de una imagen digital como la conjunción de múltiples capas con valores numéricos que representan el color de cada punto.

Los dos problemas fundamentales que se deben resolver para procesar las imágenes y extraer información son:

- Reducir la dimensión. En una imagen en escala de grises de 100x100 puntos se requieren 10 mil entradas en una red neuronal y si la primera capa tiene 1000 neuronas entonces necesitamos 10 millones de conexiones. Para implementar una Red Neuronal que procese imágenes digitales se requiere una gran capacidad de cómputo, además en [16] Lecun et al. señalan que el tamaño de la muestra de entrenamiento debe ser muy grande y las Redes Neuronales sin estructura tienen la deficiencia de que no incorporan invarianza respecto a la traslación, cambios de escala o distorsiones geométricas.
- Extraer características en las imágenes. La extracción de características en las imágenes es distinto a la extracción de características en una Red Neuronal Profunda, ya que si se fija de forma arbitraria el orden de las entradas no afecta el

proceso de aprendizaje, en el caso de las imágenes hay una estructura local y una alta correlación entre los puntos que son adyacentes.

Es decir, se debe poder reconocer un objeto, así esté estirado, movido, alargado, parcialmente oculto o en perspectiva y además se deben reconocer las características locales que luego se combinan para generar características más complejas, y entonces reconocer objetos en el espacio y el tiempo, mediante una composición de características simples a complejas.

Las Redes Neuronales Convolucionales se desarrollan para resolver estos problemas.

4. Redes Neuronales Convolucionales

De acuerdo a Lecun at al. en [16] y para asegurar algún grado de invarianza bajo traslaciones, escala o distorsiones, las Redes Neuronales Convolucionales combinan tres ideas principales:

- Campos visuales locales, estos funcionan como ventanas que recogen características.
- Parámetros compartidos (weight replication), veremos que en cada mapa de características se comparten los mismos parámetros, pesos y sesgo.
- Sub muestraje espacial, que reduce la dimensión del problema y evita el sobre ajuste por cantidad de parámetros.

A partir de esta definición se genera una nuevo campo de estudio donde se empiezan a diseñar arquitecturas de capas cada vez más profundas para resolver problemas de procesamiento de imágenes cada vez más complejos.

Vamos a describir cada capa y sus características principales.

4.1. Capa de Convolución. La Capa de Convolución es la pieza más importante de las Redes Neuronales Convolucionales. En las Redes Neuronales cada neurona está conectada con todas las neuronas de la capa previa y de la siguiente capa. En el caso de las Redes Neuronales Convolucionales las neuronas de la primera capa está relacionada con los puntos de un pequeño rectángulo de la imagen de entrada y las neuronas de la segunda capa están conectadas a las neuronas en un pequeño rectángulo de la primera capa. De esta forma se dá importancia sólo a las características locales, lo cual es útil en el procesamiento de imágenes, como hemos dicho los puntos de la imagen están fuertemente correlacionados con los puntos que están en una vecindad, que son adjacentes.

Siguiendo el proceso de las Redes Neuronales, en las primeras capas vamos a extraer características simples que se van a combinar en las capas siguientes para componer características más complejas.

En el caso del procesamiento de imágenes con Redes Neuronales Lecun et al. en [16] plantea el uso de campos de recepción que extraen características simples que luego se componen en más complejas. Estos campos de recepción se denominan kernel que se aplican a cada entrada para extraer información. Al aplicar el kernel a la entrada se genera un mapa de características que comparten los mismos parámetros de un kernel. Una Capa de Convolución se conforma entonces de múltiples mapas de características generandos mediante la operación de convolución entre la entrada y un kernel. La idea es que un kernel se especializa en detectar las mismas características sin importar donde se presentan en la entrada. Si un kernel tiene dimensión 5×5 entonces está conformado por 25 parámetros y a estos se le suma un parámetro adicional denominado sesgo. Un kernel es una matriz con valores de los parámetros que permiten extraer características al ser aplicados a la entrada de la capa previa. Estos parámetros se ajustan mediante un proceso de aprendizaje similar a la propagación reversa de las Redes Neuronales Multicapa. La Capa de Convolución se especializa en extraer características, sin importar donde se presentan, lo cual permite que sea invariante ante cambios de escala o posición.

En la Capa de Convolución se define cual es el tamaño del kernel, el desplazamiento del kernel sobre la entrada y el relleno. La Capa de Convolución toma el kernel y lo aplica a la entrada realizando en barrido y realizando una operación de multiplicación de valores para generar el mapa de características. Se generan tantos mapas de características como kernel se definan en la capa, los mapas de características son el resultado de aplicar la Capa de Convolución a la entrada. Estos valores de definición de la Capa de Convolución se denominan hiperparámetros. El desplazamiento indica los saltos que se realizan durante el barrido de la entrada y el relleno indica si se desea agregar un borde de ceros a la entrada para preservar y no perder información en los bordes de la imagen. El barrido de la entrada se realiza colocando el kernel como un campo de recepción sobre la entrada y se aplica la operación de izquierda a derecha y de arriba a abajo como se aprecia en la imagen 3.9.

El cálculo se realiza mediante un proceso iterativo. Para calcular el valor de una neurona en la capa l se toman mapas de características de la capa $l-1$ y se realiza la operación

de convolución que consiste en realizar el producto componente por componente entre dos matrices, entre los valores del mapa de características y los valores correspondientes en el kernel. Hay tantos kernel como mapas de características se desean generar y si hacemos la analogía con las Redes Neuronales los kernel son los parámetros que se ajustan, sólo que esta vez están agrupados en matrices. En la figura 3.9 se puede apreciar el recorrido que se realiza.

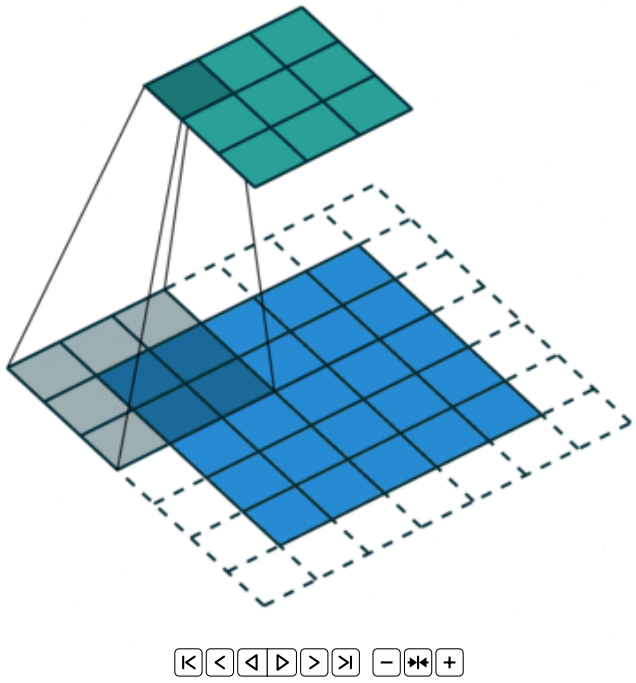


FIGURA 3.9. Convolución

Cada capa l cuenta con $k_q^{(l)}$ mapas de características que se utiliza uno a uno para realizar la operación de convolución entre la capa l y la capa $l-1$. En la capa $l-1$ podemos tener la imagen de entrada o el resultado de las capas previas. El resultado puede ser una capa de convolución o de una operación de sub muestraje que veremos más adelante.

Los kernel nos sirven como filtros que resaltan características, como contornos o esquinas, en la figura 3.11 se puede apreciar el efecto de aplicar distintos tipos de kernel y la forma como se resaltan ciertas características dependiendo del kernel. Se pueden resaltar contornos, curvas, esquinas y la combinación de dichas características componen un objeto. En el proceso se realiza una fase de extracción de características y luego se componen en la Red completamente conectada.

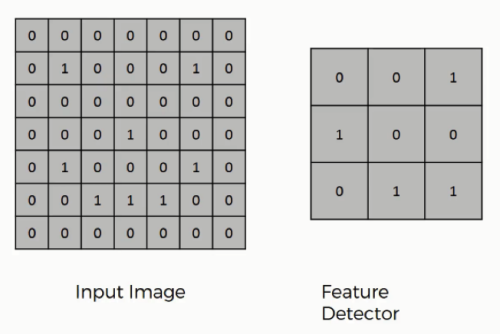


FIGURA 3.10. Imagen de entrada y núcleo detector de características

Por cada kernel que se aplica a la entrada se genera un nuevo mapa de características y cada neurona del mapa de características comparte los mismos parámetros. Esta característica es importante para disminuir la complejidad del cálculo. Un mapa de características es el resultado de aplicar el mismo kernel para generar cada neurona y así reconocer características locales en la imagen. Cada mapa servirá para detectar distintas características que se combinan en las capas siguientes para reconocer características más complejas. Adicionalmente estas características se reconocen en cualquier lugar que se presenten dentro de la imagen, lo cual hace que la Capa de Convolución produzca un resultado invariante bajo traslaciones, cambios de escala, estiramiento o transformaciones geométricas.

(a) Lenna (b) Horizontal edge (c) Vertical edge

FIGURA 3.11. Efecto de los diferentes núcleos

Dependiendo de los datos de entrada estos kernel se ajustan para captar características y en las capas siguientes se componen para identificar características más complejas. Es un proceso análogo al que se realiza con las Redes Neuronales donde las neuronas de las capas previas se combinan para identificar características más complejas en los datos. La gran

diferencia es que en el caso de las Redes Neuronales Convolucionales se realiza un proceso de extracción de características locales a través de la aplicación de los kernel a la entrada y la generación de los mapas de características.

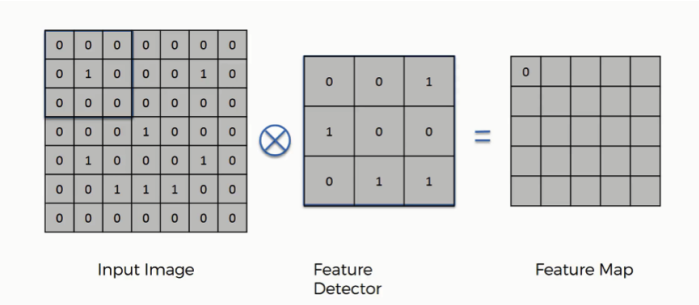


FIGURA 3.12. Mapa de características

Dado un kernel de tamaño $k_w \times k_h$ y un desplazamiento de tamaño $s = 1$ y asumiendo que es el mismo desplazamiento en la dirección horizontal y vertical, una neurona localizada en la fila i columna j de una capa dada, está conectada a las salidas de las neuronas de la capa previa localizada en la fila $(i \times s)$ a $(i * s) + k_h - 1$, columna $(j \times s)$ a $(j \times s) + k_w - 1$, donde k_w es el ancho del kernel y k_h es la altura del kernel.

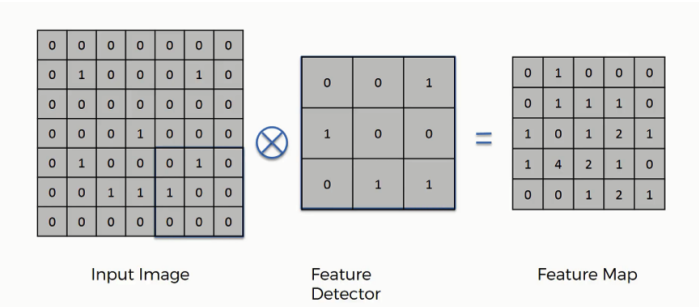


FIGURA 3.13. Operación de convolución

Si queremos obtener un mapa del mismo tamaño que el mapa de origen o entrada se agrega un relleno (padding) con ceros en el borde de la entrada. Si queremos obtener mapas más pequeños, entonces podemos cambiar el tamaño del desplazamiento y tomamos $s > 1$, de esta forma también se reduce la dimensión.

Cada kernel se aplica a la capa previa para generar los mapas de características, en la imagen 3.14, se puede apreciar cómo se aplica el kernel a cada mapa de características o a la imagen de entrada y se generan nuevos mapas de características basados en los diferentes

kernel. El proceso de aprendizaje de la Red Neuronal Convolucional consiste en ajustar los kernel para que sean más eficaces en la generación de nuevos mapas de características.

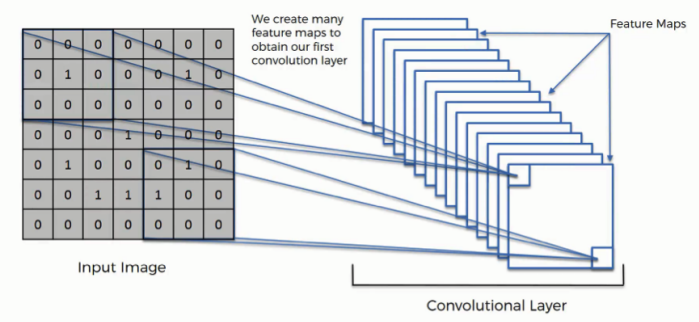


FIGURA 3.14. La capa de convolución genera un mapa de características

El cálculo de cada neurona consiste en el producto de cada valor en una ventana del tamaño del kernel por el valor correspondiente en el kernel componente a componente. Sea $z^{l}_{(i,j,k)}$ el valor de la neurona en la fila i, columna j y el mapa k de la capa l, $k^{(l)}_{w}$ es el ancho del kernel, $k^{(l)}_{h}$ es la altura del kernel, $k^{(l)}_{q}$ la cantidad de mapas de características de la capa l, s el desplazamiento y $b^{(l)}_{k}$ es el sesgo del mapa k en la capa l.

$$z^{(l)}_{(i,j,k)} = \sum_{h=0}^{k_h-1} \sum_{v=0}^{k_w-1} \sum_{f=0}^{k_q^{(l-1)}-1} z^{l-1}_{([(i\times s)+h],[(j\times s)+v],[f])} \cdot w^{(l)}_{(u,v,f,k)} + b^{(l)}_{k}, \tag{3.22}$$

Como hemos visto la imagen de entrada puede tener varias capas que representan la intensidad o valor de cada color primario de cada punto en la imagen. En la operación de convolución todos estos valores se combinan mediante la aplicación de los kernel para generar los mapas de características. Este hecho resalta debido a que en cada capa se está realizando un proceso de reducción o combinación de los datos, que es uno de los problemas que se desea resolver para hacer viable el ajuste de una Red Neuronal Convolucional.

Luego de realizar la operación de convolución, la capa siguiente puede ser una Capa de Convolución o una Capa de Sub muestraje. Las Capas de Convolución generan mapas de características y las Capas de Sub muestraje reducen la dimensión de los datos.

Recordemos que las imágenes están compuestas por múltiples capas que representan los colores que se combinan en cada punto de la imagen o en el caso de imágenes satelitales pueden haber más capas que representan la captura de otras frecuencias, como la infraroja.

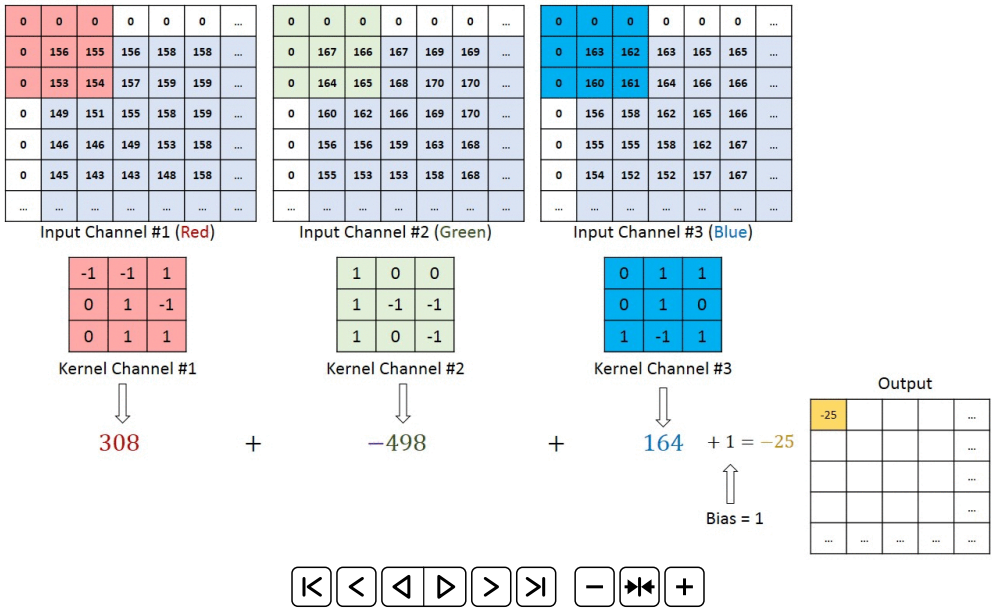


FIGURA 3.15. Capa de Convolución

Luego de la operación de convolución, se genera un mapa de características, dependiendo de los valores de los hiperparámetros la dimensión del mapa de características varía. Asumiendo que la entrada y el kernel sean cuadrados, sea $d^{(l-1)}$ la dimensión de la capa $l-1$, que es la entrada, $k_h^{(l)} = k_v^{(l)}$ la dimensión del kernel, $p^{(l)}$ el relleno y $s^{(l)}$ el desplazamiento en la capa l. La fórmula para calcular la dimensión resultante del mapa de características de la capa l, $d^{(l)}$, es la siguiente,

$$d^{(l)} = \left[\frac{d^{(l-1)} + (2 \times p^{(l)}) - k_h^{(l)}}{s^{(l)}}\right] + 1, \tag{3.23}$$

donde el operador $[x]$ denota la parte entera de x.

Por ejemplo, si la entrada es de dimensión 5×5 y le aplicamos un kernel de dimensión 3×3 y el desplazamiento es 1, sin relleno, nos queda un mapa de características de dimensión 3×3.

Siendo $n^{(l)}$ la cantidad de neuronas de la capa l, la fórmula para obtener $n^{(l)}$ es la siguiente,

$$n^{(l)} = [d^{(l)}]^2 \times k_q^{(l)}. \tag{3.24}$$

Sea $w^{(l)}$ la cantidad de parámetros de la capa l, $k_h^{(l)}$ y $k_v^{(l)}$ el ancho y el alto del kernel y $k_q^{(l)}$ la cantidad de mapas de características en la misma capa, la cantidad de parámetros de cada Capa de Convolución se calcula mediante la fórmula siguiente,

$$w^{(l)} = [k_h^{(l)} \times k_v^{(l)} \times k_q^{(l-1)} + 1] \times k_q^{(l)}. \tag{3.25}$$

Se puede notar que se suma 1 para considerar el sesgo y se considera la cantidad de mapas de características de la capa anterior.

Para generar la invarianza del método también se reduce la dimensión de los datos y así evitar que la posición sea relevante, para ello se implementa la Capa de Sub muestraje.

4.2. Capa de Sub muestraje. En la Capa de Sub muestraje o pooling se realiza una operación simple para reducir la dimensión de los datos. El propósito es reducir la carga computacional, el uso de memoria y el número de parámetros y este hecho en consecuencia contribuye en limitar el riesgo de sobre ajuste del modelo al disminuir el número de parámetros.

En la imagen 3.16 se puede ver que el tamaño se reduce a la mitad si aplicamos la operación de Sub muestraje con un kernel de 2×2 donde $k_h = 2$, $k_v = 2$, $s = 2$ y $p = 1$.

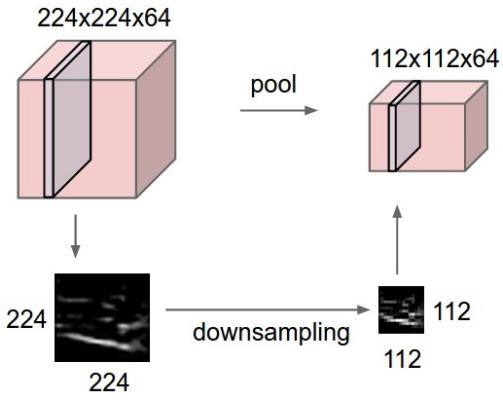


FIGURA 3.16. Sub muestraje

En esta capa se define el tamaño de la ventana sobre la cual se toma un valor que pasa a la capa siguiente. Es decir, se realiza un barrido sobre los mapas de características utilizando una ventana de un tamaño fijo y se toma una valor que va a representar la información de la ventana.

Cada neurona de una capa está relacionada con un conjunto de neuronas de la capa previa localizadas en la ventana que se define. Esto permite relacionar la información adjacente o local, lo cual permite reconocer patrones. Los hiperparámetros de la Capa de Sub muestraje son: el tamaño de la ventana o campo receptor, el desplazamiento y el tipo de relleno. La capa

no genera nuevos parámetros, sólo utiliza una función que genera un valor representativo de cada ventana de recepción y se descarta el resto de la información.

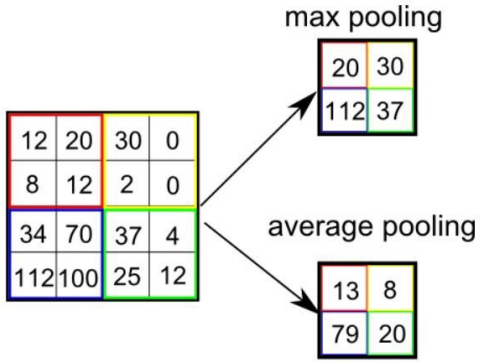


FIGURA 3.17. Sub muestraje por máximo o por promedio

Principalmente se utilizan dos formas para realizar la operación de reducción de la dimensión o Sub muestraje, Sub muestraje por máximo (max pooling) o Sub muestraje por promedio (average pooling). Como se puede ver en la imagen 3.17 y considerando una ventana de 2×2 y un desplazamiento $s = 2$, en el primer caso se toma el valor máximo de la ventana y en el segundo caso se toma el promedio de los cuatro valores. Sólo el valor máximo de la ventana de recepción pasa a la capa siguiente o en otro caso sólo el promedio sigue a la capa siguiente, el resto de la información en la ventana se descarta.

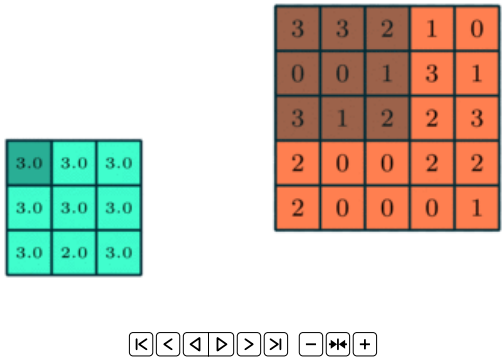


FIGURA 3.18. Sub muestraje por máximo sin dezplazamiento

Es evidente la pérdida de información en una capa de esta naturaleza. Para reducir aun más la cantidad de información que pasa a la capa siguiente también se puede definir el desplazamiento. El recorrido de la ventana sobre cada mapa de características se puede realizar con un desplazamiento 1 a 1 como se puede ver en la imagen 3.18.

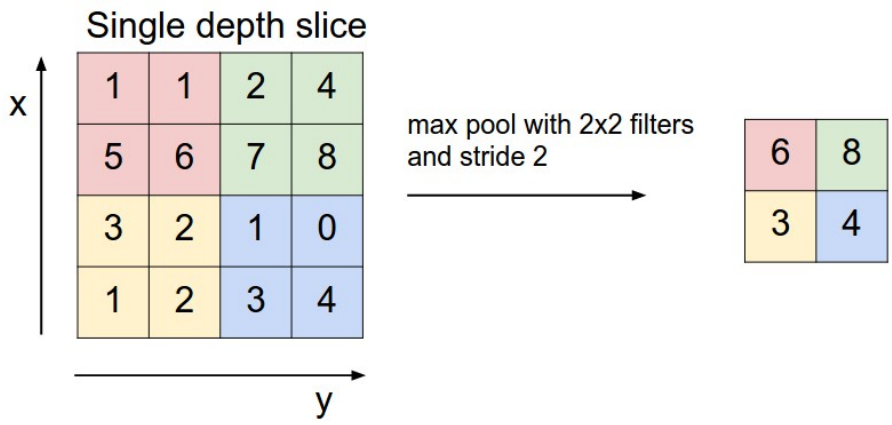


FIGURA 3.19. Sub muestraje por máximo

Para reducir la dimensión aún más, se puede realizar el recorrido con desplazamiento. En la figura 3.19 se ver un ejemplo del cálculo y el efecto de reducir la dimensión a un cuarto, de una matriz de 4×4 generamos una matriz de 2×2 luego de la operación de Sub muestraje.

4.2.1. *Ajuste de las Redes Neuronales Convolucionales.* El ajuste se realiza mediante el método de propagación reversa similar al método aplicado a las Redes Neuronales Multicapa, con la diferencia de que se deben propagar los gradientes a las neuronas que sobreviven a las Capas de Convolución y Sub muestraje.

Recordemos que cada neurona en un Red Neuronal Convolucional está relacionada con un kernel que comparte los parámetros con todas las neuronas del mapa de características al cual la neurona pertenece. Es por ello que al realizar la propagación reversa sólo se deben actualizar los gradientes de las neuronas que sobreviven.

4.3. Arquitectura de las Redes Neuronales Convolucionales. Las Redes Neuronales Convolucionales consisten en arquitecturas de capas que combinan las operaciones de convolución y sub muestraje en varias capas para luego alimentar una red neuronal completamente conectada. Se puede visualizar como un proceso de extracción de características, combinación de características, estimación y ajuste.

El trabajo realizado por LeCun et al. en [6] y [16] representa un avance fundamental en el diseño y entrenamiento de las Redes Neuronales Convolucionales Profundas para procesar imágenes. La entrada de LeNet5 consiste en una imagen que representa un dígito escrito a mano centrado y normalizado. Cada neurona en una capa está relacionada con un conjunto de neuronas que están en una vecindad de la capa previa, así se preserva la estructura de la

imagen. Esta idea proviene de los trabajo de Hubel y Wiesel en [5] y consiste en la existencia de campos de recepción visual especializados que detectan características, esquinas y bordes y estas características luego se combinan para conformar figuras más complejas.

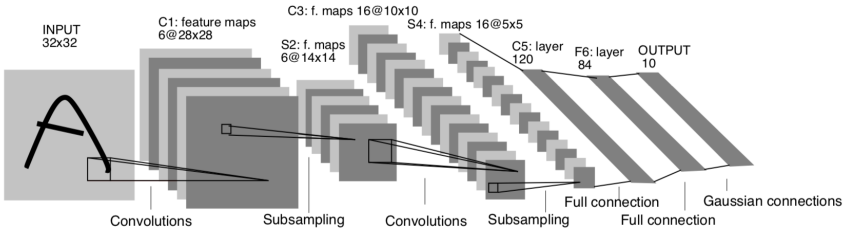


Fig. 1. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

FIGURA 3.20. Arquitectura de LeNet5

El resultado del proceso de clasificación mediante el uso de una Red Neuronal Convolucional para clasificar dígitos escritos a mano se aprecia en 3.21. Se puede ver que hay una imagen de entrada, cada capa aplica la operación de convolución y sub muestraje en el proceso de entrenamiento. Una Red ya ajustada genera las probabilidades de pertenercer a alguna de las 10 clases y se toma la mayor como el resultado inferido por la Red.

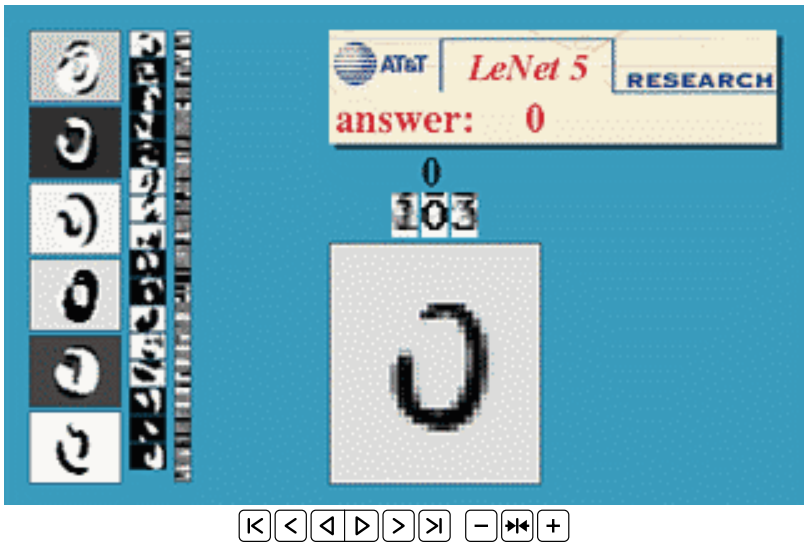


FIGURA 3.21. Resultado de LeNet5

En la arquitectura se puede apreciar como aumenta la cantidad de mapas de características, k_q, en la medida que se reduce la dimensión de los datos mediante la operación de convolución y sub muestraje con desplazamiento y relleno respectivamente. La idea es que los mapas sintenizan las características presentes en las imágenes. La combinación de Capas de Convolución y Capas de Sub muestraje se inspiran en los trabajos de Hubel y Wiesel en [5] donde se realiza el planteamiento de la existencia de neuronas que se activan con figuras o estímulos simples y otras se activan con figuras o estímulos complejas. LeCun indica que se puede lograr un alto grado de invarianza ante transformaciones geométricas de la entrada con la reducción espacio temporal de la entrada que se compensa con el incremento de los mapas de características.

Arquitectura de LeNet5						
Capa	Nombre	k_q	Tamaño	$k_h \times k_v$	s	Activación
Entrada	Imagen	1	32x32x1	-	-	-
1	Convolución	6	28x28x6	5x5	1	tanh
	Sub muestraje por promedio	6	14x14x6	2x2	2	tanh
2	Convolución	16	10x10x16	5x5	1	tanh
	Sub muestraje por promedio	16	5x5x16	2x2	2	tanh
3	Convolución	120	1x1x120	5x5	1	tanh
4	Completamente conectadas	-	84	-	-	tanh
Salida	Completamente conectadas	-	10	-	-	RBF

CUADRO 2. Arquitectura de LeNet5

4.4. Arquitecturas recientes. Gran parte del desarrollo reciente en el reconocimiento de patrones en imágenes mediante el uso de Redes Neuronales Convolucionales Profundas se debe a la competencia promovida por el Stanford Vision Lab que utiliza ImageNet como datos de entrenamiento denominada ILSVRC (ImageNet Large Scale Visual Recognition Challenge).

En la ILSVRC se evaluan los algoritmos para detección de objetos y clasificación de imágenes a gran escala. La idea general es permitir que los investigadores comparen sus

avances en la detección de objetos utilizando una gran variedad de conceptos y aprovechando el esfuerzo que se realizó en la recolección y etiquetado de ImageNet.

Como se describe en http://image-net.org, ImageNet es una base de datos de imágenes organizadas siguiendo la jerarquía de WordNet. Cada concepto se agrupa en conjuntos de sinónimos cognitivos, distinguiendo así un concepto. WordNet es una base de datos que se utiliza en actividades de procesamiento de lenguaje natural.

Existen más de 100 mil conjuntos de sinónimos de los cuales la mayoría (más de 80 mil) son nombres. Cada grupo cuenta con un promedio de 1000 imágenes representativas. ImageNet pretende proveer una base de datos que represente todos los conceptos en la jerarquía de WordNet.

4.4.1. *AlexNet 2012.* El avance más importante y que genera mayor interés en las Redes Neuronales Convolucionales Profundas es AlexNet en el 2012. Esta red se entrena con los datos de la competencia promovida por ImageNet. El trabajo está detallado en [18] y la descripción que realizan los autores es la siguiente:

> Hemos entrenado una gran Red Neuronal Convolucional Profunda para clasificar 1.2 millones de imágenes de alta resolución, en la competencia ImageNet LSVRC-2010, en 1000 clases diferentes. En los datos de prueba hemos alcanzado el primer lugar y el quinto lugar con tasas de error de 37.5 % y 17 %, las cuales son considerablemente mejores que el estado del arte. La Red Neuronal cuanta con 60 millones de parámetros y 650 mil neuronas que consisten de cinco (5) Capas de Convolución, algunas de las cuales están seguidas de Capas de Sub muestraje por máximo y tres capas completamente conectadas a una capa softmax de 1000 clases. Para acelerar el entrenamiento, se utilizan neuronas que no se saturan y una implementación muy eficiente de las capas de convolución en GPU. Para reducir el sobreajuste en las capas completamente conectadas utilizamos un método de regularización desarrollado recientemente denominado descarte (dropout), que ha probado ser bien efectivo. Participamos en la ILSVRC-2012 con una variante de este modelo y alcanzamos el primer lugar con una tasa de error de 15.3 % comparada con 26.2 % alcanzada por el segundo lugar.

Los avances más importantes de AlexNet se resumen en el uso de ReLU como función de activación para evitar la saturación de las neuronas y el uso de tarjetas de video, en particular la programación de múltiples unidades de procesamiento gráfico (GPU) para acelerar el entrenamiento de la Red.

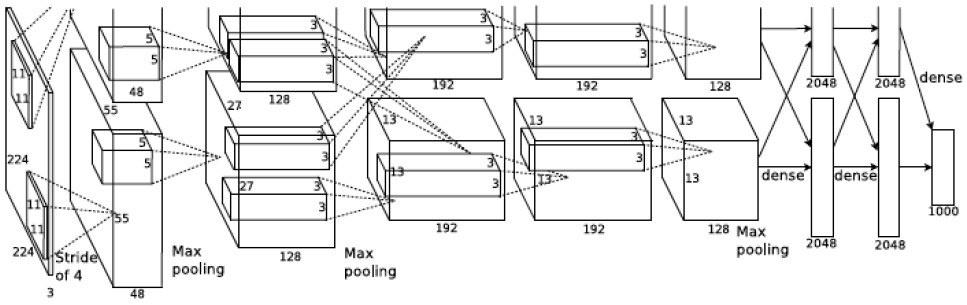


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

FIGURA 3.22. AlexNet

En [18], la arquitectura es descrita de la forma siguientes,

> La arquitectura de la Red Neuronal Convolucional, como se puede apreciar en la figura 3.22, cuenta con ocho (8) capas con parámetros. Las primeras cinco (5) son Capas de Convolución y tres (3) son de una Red Neuronal completamente conectada.
>
> Al resultado de la última capa se le aplica la función softmax para producir una clasificación de mil (1000) clases. La Red maximiza la función de regresión logística multinomial, que es equivalente a maximizar la media del logaritmo de la probabilidad de los datos observados bajo la distribución de predicción de los datos de prueba.
>
> Los kernel de la segunda, cuarta y quinta Capas de Convolución se conectan sólamente a aquellos mapas de características de la capa previa que fueron calculadas con el mismo GPU. Los kernel de la tercera Capa de Convolución están conectados a todos los mapas de características de la segunda capa. Las neuronas de las capas completamente conectadas están conectadas a todas las neuronas en la capa previa. Las respuestas de la primera y segunda Capa de

Convolución se normalizan. Las Capas de Sub muestraje por el máximo, se aplican a la primera primera, segunda y quinta Capa de Convolución.

La función de activación ReLU se aplica a la salida de cada Capa de Convolución y a cada capa completamente conectada. La primera Capa de Convolución procesa la imagen de dimensión $227 \times 227 \times 3$ con 96 kernel de tamaño $11 \times 11 \times 3$ con un desplazamiento de 4 (esta es la distancia entre los centros de los kernel de neuronas adjacentes en el mapa de características). La segunda Capa de Convolución toma como entrada (después de normalizar y realizar el sub muestraje) la salida de la primera Capa de Convolución y la procesa con 256 kernel de dimensión $5 \times 5 \times 48$. La tercera, cuarta y quinta Capa de Convolución están conectadas directamente sin realizar ningún proceso de normalización y sub muestraje. La tercera Capa de Convolución tiene 384 kernel de dimensión $3 \times 3 \times 256$ conectado al resultado, normalizado y reducido por sub muestraje, de la segunda Capa de Convolución. La cuarta Capa de Convolución cuenta con 384 kernel de dimensión $3 \times 3 \times 192$ y la quinta Capa de Convolución tiene 256 kernel de tamaño $3 \times 3 \times 192$. Las capas completamente conectadas tienen 4096 neuronas cada una.

La dimensión de cada mapa de características se calcula mediante la fórmula 3.23, en la imagen 3.23 se puede ver el proceso de cálculo capa por capa.

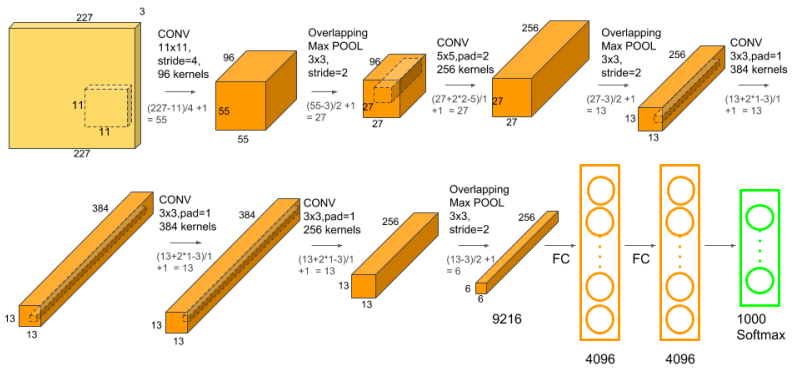


FIGURA 3.23. Dimensiones de cada capa en AlexNet

En el cuadro 3 se puede ver el detalle de la arquitectura y los hiperparámetros de la red.

Arquitectura de AlexNet						
Capa	Nombre	k_q	Tamaño	$k_h \times k_v$	s	Activación
Entrada	Imagen	1	227x227x3	-	-	-
1	Convolución	96	55x55x96	11x11	4	relu
	Sub muestraje por máximo	96	27x27x96	3x3	2	relu
2	Convolución	256	27x27x256	5x5	1	relu
	Sub muestraje por máximo	256	13x13x256	3x3	2	relu
3	Convolución	384	13x13x384	3x3	1	relu
4	Convolución	384	13x13x384	3x3	1	relu
5	Convolución	256	13x13x256	3x3	1	relu
	Sub muestraje por máximo	256	6x6x256	3x3	2	relu
6	Completamente conectadas	-	9216	-	-	relu
7	Completamente conectadas	-	4096	-	-	relu
8	Completamente conectadas	-	4096	-	-	relu
Salida	Completamente conectadas	-	1000	-	-	Softmax

CUADRO 3. Arquitectura de AlexNet

Esta arquitectura representa un gran avance debido a aspectos como la selección de la función de activación, que se vienen estudiando desde finales de los años 80, en la literatura previa al 2012 como [4], todavía aparece el uso de *tanh* que se satura para valores grandes. Este pequeño cambio permitió la convergencia de estas redes con la ventaja de la simplificación de los cálculos, dándo relevancia al estudio de funciones de activación que no se saturan como ReLU y Leaky ReLU.

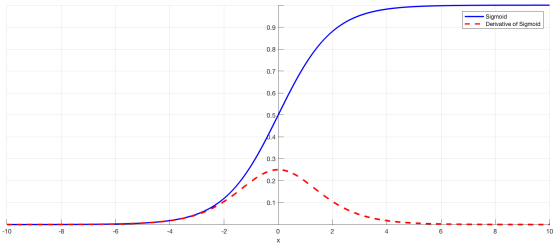


FIGURA 3.24. Sigmoide y su derivada

Recordemos que si $\sigma(x)$ representa la sigmoide entonces su derivada es,

$$\sigma(x)' = \sigma(x)(1 - \sigma(x))$$

. La saturación de una neurona consiste en que para ciertos valores la derivada de la función de activación se anula. En la imagen 3.24 se puede apreciar que para valores altos o bajos la sigmoide es 0 o 1 y la derivada es muy cercana a cero. Cuando se realiza el proceso de ajuste y se utiliza la derivada de la función de activación, no hay nada que propagar.

En el caso de la tangente hiperbólica, su derivada es,

$$tanh(x)' = 1 - tanh(x)^2$$

, y ocurre una situación similar como se puede ver en 3.25. Este comportamiento se evita mediante el uso de funciones de activación como ReLU en AlexNet.

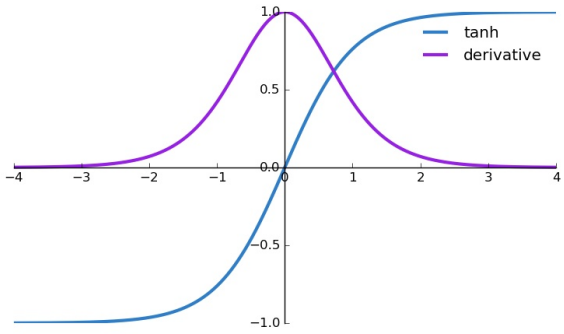


FIGURA 3.25. Tanh y su derivada

A continuación una descripción de las arquitecturas que ganan la competencia de la ILSVRC desde el 2013 al 2015, siguiendo algunas notas del curso sobre Redes Convolucionales del MIT dictado por Andrej Karpathy en [19].

4.5. ZF Net 2013. En el año 2013 el ganador de la competencia ILSVRC 2013 es el equipo compuesto por Matthew Zeiler y Rob Fergus. La red se denominó ZFNet por las siglas de los nombres de los autores. Esta red es una modificación de AlexNet, se realizan cambios en los hiperparámetros, en particular se engrandece el tamaño de las Capas de Convolución intermedias y haciendo el desplazamiento y el tamaño de los kernel más pequeños. El error pasó de 15,3 % a 11,2 %.

4.6. GoogLeNet 2014. En el 2014, en la competencia de la ILSVRC el error pasa de 11,2 % a 6,7 % mediante una Red Convolucional hecha por el equipo de Szegedy de Google. Su principal contribución fué el desarrollo del modulo denominado Inception que reduce los parámetros significativamente. Pasa a ser 4 millones de parámetros y AlexNet cuenta con 60 millones de parámetros. Adicionalmente se utiliza sub muestraje por promedio. Esta red ha generado nuevas versiones conocidas por el nombre Inception en sus distintas versiones.

4.7. VGGNet 2014. En el mismo año 2014 el error pasa a ser de 11,2 % a 7,3 % con el equipo de Karen Simonyan y Andrew Zisserman y la red es conocida como VGGNet. Su principal contribución fue mostrar que la profundidad de la red es importante para su buen desempeño. La arquitectura final tiene 16 Capas de Convolución con hiperparámetros homogéneos con kernel de 3×3 y sub muestraje de 2×2. La desventaja es la cantidad de parámetros y memoria, consumiendo cerca de 140 millones.

4.8. ResNet 2015. Finalmente en 2015 gana la competencia el equipo de Kaiming He y el error pasa de 6,7 % a 3,57 %. La arquitectura utiliza conexiones de salto y normalización por lotes. Además no se utilizan capas completamente conectadas al final de la arquitectura.

Cabe destacar que el error humano en la clasificación de las imágenes es de 5,1 % que se supera finalmente con ResNet en el 2015.

4.9. Otros modelos. Se han desarrollado muchos módelos con técnicas más avanzadas que las expuestas en este trabajo. En la documentación de keras podemos conseguir una lista de modelos pre entrenados que se pueden utilizar como punto de partida para desarrollar soluciones a problemas en la industria.

En la lista 3.26 de ambientes de desarrollo pre entrenados podemos ver cuanta memoria requieren sólamente los parámetros del modelo y van de un rango de 14Mb a 550Mb y de 23 capas a 572 con un máximo de 82 % de exactitud en Top-1 y 96 % en Top-5. Este tipo de requerimientos nos indica que se requieren sistemas medianamente complejos para realizar inferencias con los modelos pre entrenados. En particular llama la atención el modelo MobileNetV2 que requiere apenas 14Mb para cargar los parámetros del modelo y de esta manera realizar actividades de inferencia con imágenes en un dispositivo móvil inteligente.

Se vienen desarrollando soluciones en Hardware para acelerar el procesamiento de imágenes en tiempo real para aplicaciones de misión crítica. El Piloto Automático de Tesla

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

FIGURA 3.26. Modelos pre entrenados con Keras

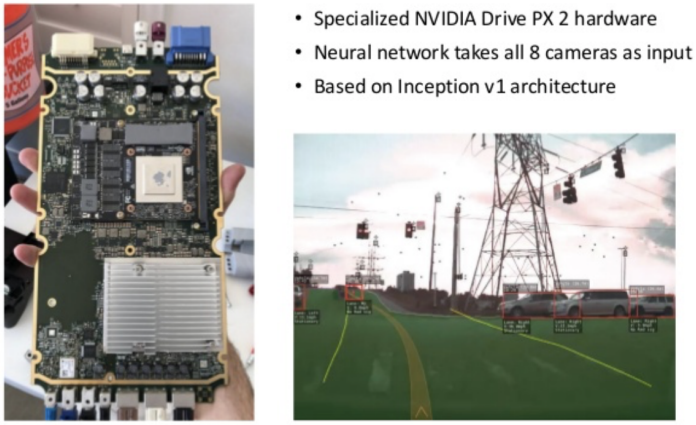


FIGURA 3.27. Hardware para el Piloto Automático de un Carro Tesla

utiliza un nuevo Hardware NVIDIA, como el de la imagen 3.27, para procesar la imágenes generadas por 8 cámaras y utiliza un modelo basado en Inception v1. De esta manera se realizan las actividades de inferencia necesarias para el manejo autónomo como, detección de señales, semáforos, vías, vecindad, obstáculos, peatones, entre otros.

Cabe destacar que la implementación de nuevos modelos puede partir de los modelos pre entrenados y desde estos construir soluciones. Es costoso en tiempo y dinero entrenar estas redes, ya que ameritan recursos de memoria y computacionales que se pueden ahorrar mediante el uso de modelos pre entrenados.

Capítulo 4

Metodología

Desde el punto de vista metodológico hemos realizado las actividades de forma ordenada partiendo de las bases de datos que se van a utilizar, el ajuste de modelos de aprendizaje con Redes Neuronales Convolucionales para identificar objetos en imágenes, implementación de modelos de detección de objetos en Amazon Deeplens, repaso de las posibles aplicaciones y cómo se puede utilizar la plataforma de Amazon para desplegar los modelos.

Para cada modelo que hemos trabajado se ha escrito un cuaderno de notas que recoge todos los pasos para ajustar los modelos. Estos detalles los hemos dejado en los apéndices ya que reflejan los aspectos más prácticos y concretos de la implementación de modelos de aprendizaje con Python.

Para ajustar los modelos se utilizó una computadora personal Mac Book Pro 2017 con un procesador de 3,1 GHz Intel Core i7 y 16Gb de RAM.

Iniciemos haciendo una breve descripción de los datos utilizados.

1. Bases de Datos

Para realizar el trabajo hemos utilizado dos conjuntos de datos MNIST en [20] y CIFAR en [21]. Ambas bases de datos de imágenes son referencias para realizar experimentos e investigación sobre detección de objetos en imágenes. La selección de ambas bases de datos se debe a que son manejables por un computador personal desde el punto de vista del almacenamiento y el procesamiento.

1.1. MNIST. La base de datos de imágenes MNIST consiste en un conjunto de entrenamiento de 60 mil imágenes de dígitos escritos a mano y un conjunto de prueba de 10 mil imágenes. Los dígitos se normalizaron en tamaño y están centrados en una imagen de tamaño 28×28.

La base de datos fue creada para probar algoritmos de aprendizaje y métodos de reconocimiento de patrones sin requerir grandes esfuerzos de pre procesamiento y formateo.

Las imágenes se proveen en archivos comprimidos con las características siguientes:

- train-images-idx3-ubyte.gz: conjunto de entrenamiento (9,9Mb).
- train-labels-idx1-ubyte.gz: etiquetas del conjunto de entrenamiento (28,8k).
- t10k-images-idx3-ubyte.gz: conjunto de prueba (1,65Mb).
- t10k-labels-idx1-ubyte.gz: etiquetas del conjunto de prueba (4,5k).

Este conjunto de datos se ha utilizado extensamente en la biliografía y la investigación sobre identificación de patrones en imágenes, en 4.1 se puede ver un ejemplo de las imágenes que hay en la base de datos.

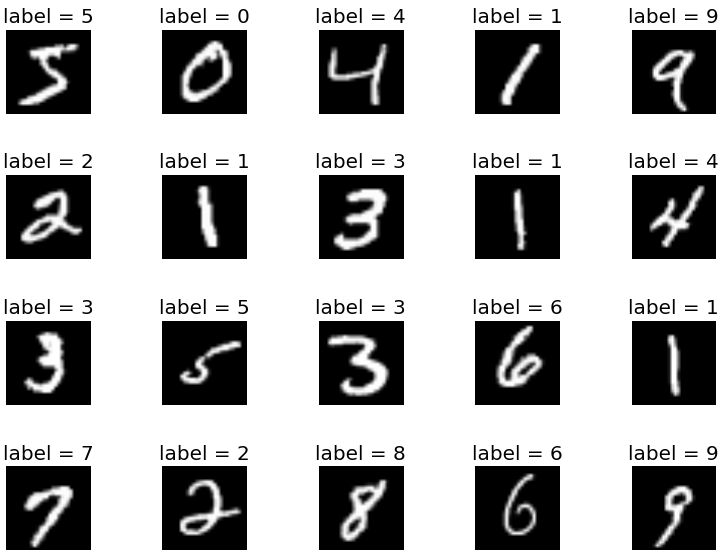


FIGURA 4.1. Base de datos de dígitos escritos a mano MNIST

1.2. CIFAR. La base de datos CIFAR-10 está conformada por 60 mil imágenes a color, correspondientes a 10 clases, de dimensión 32×32 de las cuales hay 6 mil por cada clase. Hay 50 mil imágenes en el conjunto de entrenamiento y 10 mil imágenes en el conjunto de prueba. El conjunto de datos está dividido en seis lotes de 10 mil imágenes. El conjunto de entrenamiento cuenta con mil imágenes de cada clase ordenadas aleatoriamente.

Las clases de la base de datos son las siguientes:

- Aeroplano
- Automóvil
- Pájaro

- Gato
- Venado
- Perro
- Rana
- Caballo
- Barco
- Camión

Las clases son excluyentes mutuamente y no hay solapamiento entre las clases.

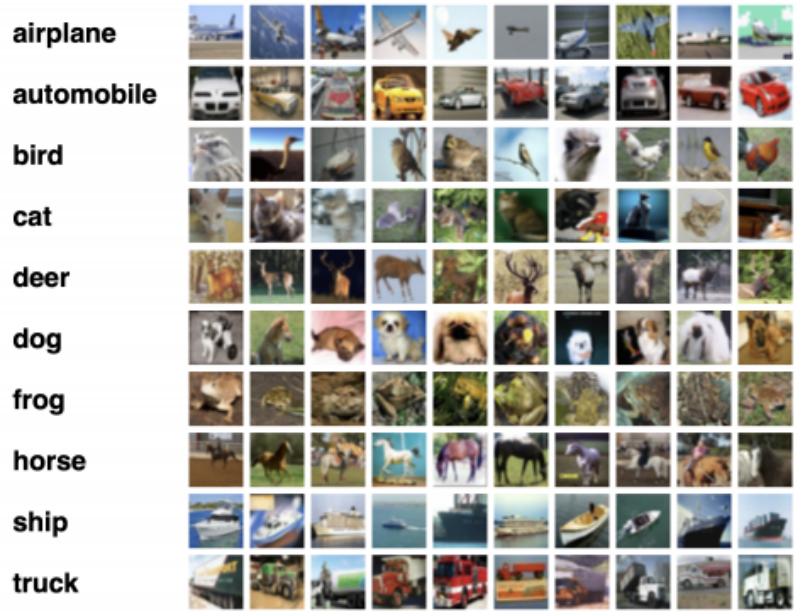


FIGURA 4.2. Base de datos de objetos

2. Ajuste de Redes neuronales

En el apéndice A y a modo de ejemplo didáctico hemos escrito un ejemplo de cómo se realiza el proceso de estimación y ajuste a un conjunto de datos lineales generados manualmente.

Se generaron los datos a partir de una función lineal $f(x) = 2x - 10$ y se hace un modelo que muestra paso a paso el proceso de estimación y ajuste de un perceptrón simple, que es la Red Neuronal más pequeña y así mostrar cómo se comporta una neurona.

Se definen las funciones de ayuda para realizar el proceso, entre ellas,

- `linear` es la función de activación.
- `forward` es la fase de estimación que le aplica la función de activación a la entrada multiplicada por el parámetro y se suma el sesgo.
- `backward` es la fase de ajuste del parámetro mediante el descenso del gradiente, que utiliza la tasa de aprendizaje **alpha** y el gradiente. El gradiente es la derivada de la función de error con respecto al parámetro, que en nuestro caso es la función de error cuadrático.
- `error` es la función que calcula el error cuadrático entre el valor estimado y el valor observado en los datos.

Además se programa un ciclo que realiza el proceso de estimación y ajuste hasta que se alcanza el nivel de error aceptable.

El ciclo consiste en:

- Realizar el paso de estimación denominado `forward` que combina la entrada y el peso $w1$ y le suma el sesgo $b1$, para luego aplicar la función de activación.
- Se calcula el error cuadrático entre el valor estimado en la fase previa y el valor observado mediante la función `error`.
- Calcular los gradientes, que consiste en calcular el valor de la derivada de la función de error con respecto a los parámetros $w1$ y con respecto a $b1$.
- Aplicar el paso de ajuste `backward` que aplica el descenso del gradiente para obtener los nuevos valores de los parámetros.
- Se almacenan los valores obtenidos para graficar los resultados.

En el modelo se logra obtener los parámetros de acuerdo al proceso de generación de los datos y el proceso converge alrededor de la iteración 50.

Este ejemplo muestra el proceso que se sigue en el ajuste de cualquier red neuronal y es el mismo para el resto de las arquitecturas salvo que se agregan nuevas técnicas para mejorar el ajuste de los modelos y su desempeño.

3. Ajuste de Redes Neuronales Convolucionales

Para realizar el ajuste de las Redes Neuronales Convolucionales hemos utilizado la librería y ambiente de desarrollo de algoritmos de aprendizaje profundo keras programadas en Python.

Partiendo de la base de datos MNIST y CIFAR de imágenes de dígitos escritos a mano e imágenes de objetos, se realizó el ajuste de una Red Neuronal Convolucional con cada conjunto de datos y se siguieron los pasos siguientes:

- Incluir librerías y definir parámetros.
- Cargar los datos.
- Ajustar los datos.
- Definir el modelo en Keras.
- Ajustar el modelo.
- Evaluar el modelo.

Se crea la matriz de confusión en cada caso y se observan particularidades en los datos.

En el apéndice B se incluye todo el programa que realiza el ajuste y se explica paso a paso el proceso siguiendo los pasos descritos. En el caso de MNIST y siguiendo las fuentes y ejemplos en la documentación oficial de keras ajustamos un modelo con dos Capas de Convolución de 32 y 64 mapas de características respectivamente, con kernel de 3×3 y ReLU como función de activación. Luego se aplica una Capa de Sub muestraje con ventana de 2×2 y finalmente una red reuronal conectada con 128 neuronas.

Para el modelo se utiliza la entropía, Adadelta como algoritmo de optimización y la métrica es la exactitud. El modelo ajustado tiene una exactitud de 99 % con los datos de prueba.

En el apéndice C se muestra el cuaderno de notas donde se realiza ajuste de una Red Neuronal Convolucional con los datos CIFAR-10 utilizando una arquitectura similar, pero con más Capas de Convolución y más Capas de Sub muestraje.

De nuevo se hace un proceso detallado paso a paso del proceso de obtención de los datos, creación del modelo, ajuste, pruebas y generación de métricas de desempeño.

4. Implementación de una Red neuronal convolucional en Amazon Deeplens

Partiendo de la documentación provista por Amazon se activó un modelo pre entrenado de reconocimiento de objetos basado en un modelo de **MxNet** en [22].

4.1. Activar y Configurar AWS DeepLens. El dispositivo Amazon Deeplens está hecho de tal manera que se protege la forma como se despliegan los proyectos y se

establecen políticas de seguridad para evitar la manipulación maliciosa de los dispositivos y sus resultados.

Es por ello que se realiza un proceso de activación del dispositivo en la plataformna de Amazon, que sigue los pasos siguientes:

4.1.1. *Ingreso.* Luego de ingresar a la cónsola de AWS con sus credenciales personales hay que buscar el servicio Amazon Deeplens utilizando la barra de búsqueda como se puede ver en 4.3.

Luego aparece la pantalla de gestión de dispositivos donde se inicia el registro presionando el botón **Registrar el dispotivo** como se puede ver en la imagen 4.4.

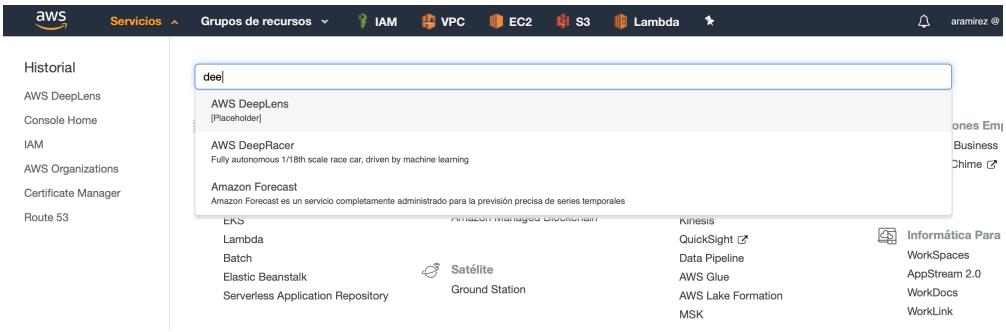


FIGURA 4.3. Deeplens en cónsola de AWS

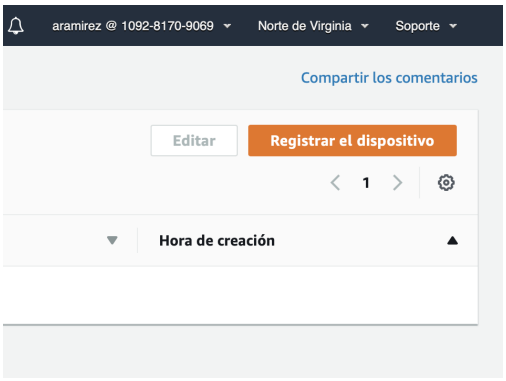


FIGURA 4.4. Registro del dispositivo

4.1.2. *Selección de dispositivo y configuración.* Amazon Deeplens ya lanzó una segunda versión y es por ello que es necesario identificar la versión del dispositivo para iniciar el proceso de configuración como se muestra en la imagen 4.5. Luego aparece la pantalla de configuración, como en la imagen 4.6, donde se asigna un nombre al dispositivo, se asignan

los permisos para gestionar dispositivos Deeplens y se genera y descarga el certificado de conexión del dispositivo.

Este certificado luego se coloca en el dispositivo directamente para que se pueda comunicar con Amazon de forma segura.

FIGURA 4.5. Selección de la versión del dispositivo

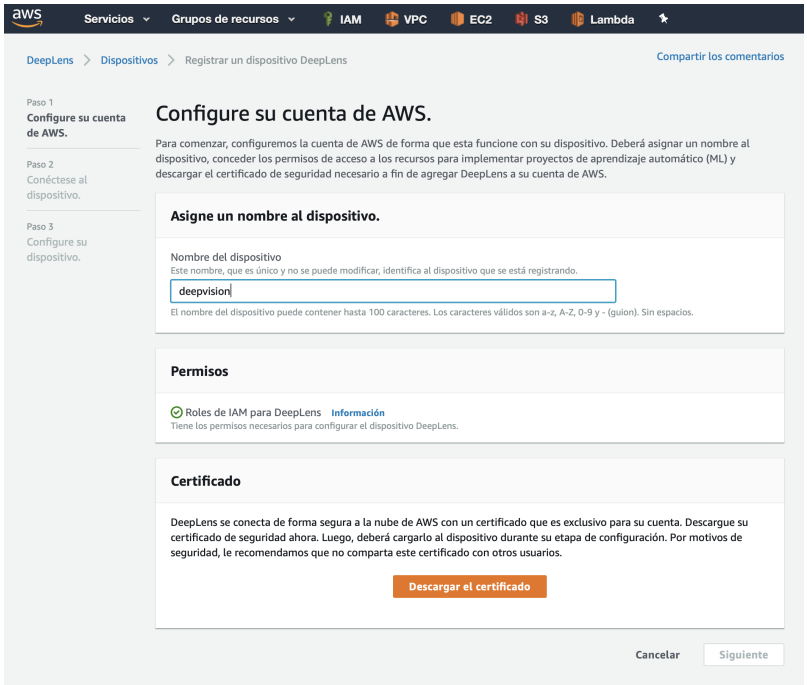


FIGURA 4.6. Configuración del dispositivo

Después de configurar los parámetros puede continuar el proceso y ahora corresponde realizar la configuración del dispositivo fuera de línea. Es decir, se enciendo una red wifi en el dispotivos para conectarnos directamente en una red privada y aparece la pantalla 4.8 y la pantalla 4.9 donde se establece la configuración de la red wifi que el dispositivo va a utilizar, se configura el certificado de conexión y se establece una clave de acceso.

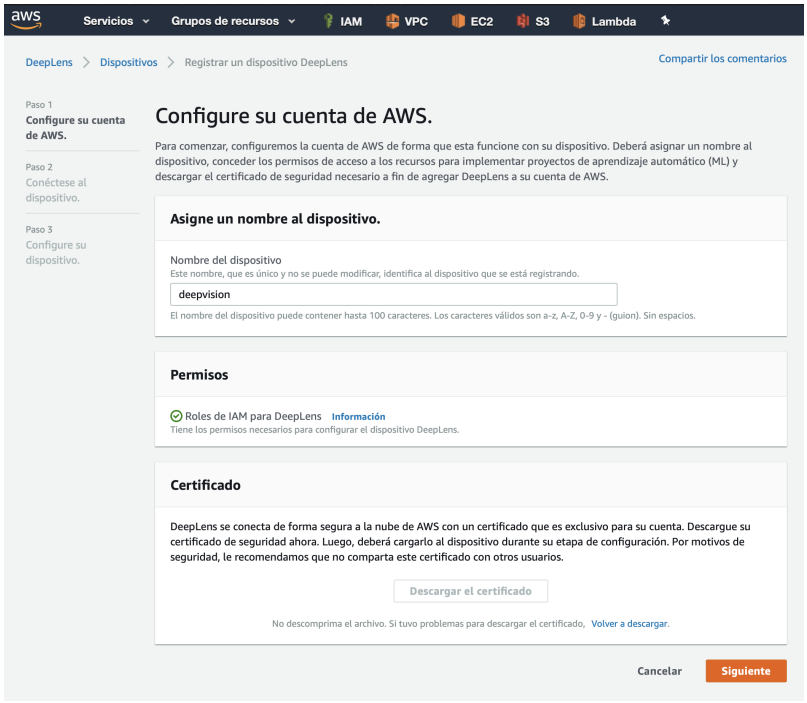


FIGURA 4.7. Permisos y certificados de conexión

Este proceso aunque muy seguro, es muy engorroso y no es fluido en el sentido de que mientras se realizan los pasos hay errores que implican iniciar de nuevo. Finalmente se confirma la configuración y ahora aparece el dispositivo activado en la cónsola de AWS como en la imagen 4.10.

Si hay alguna actualización del dispositivo pendiente entonces esta se realiza inmediatamente y como se muestra en la imagen 4.11 aparece registrado y en línea.

Sólo después de estos pasos de configuración del dispositivo, permisos, credenciales y acceso a las redes es que es posible implementar un modelo.

Cabe destacar que la configuración del WiFi es estática. El dispositivo no va a aparecer activado si se conecta a través de otra red. De hecho si se desea implementar un modelo en

FIGURA 4.8. Conexión vía WiFi al dispositivo para configurar certificado

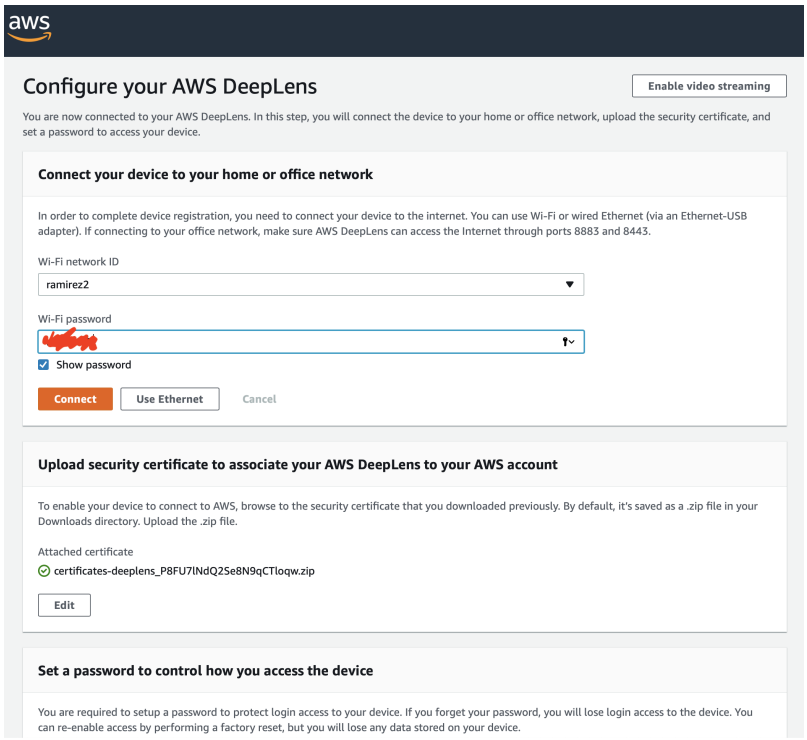


FIGURA 4.9. Configuración de WiFi, certificado y clave

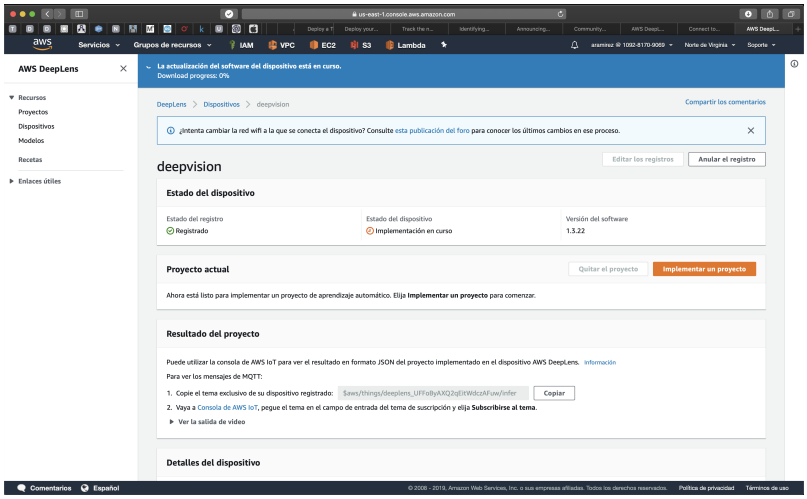


FIGURA 4.10. Final de la configuración del dispositivo

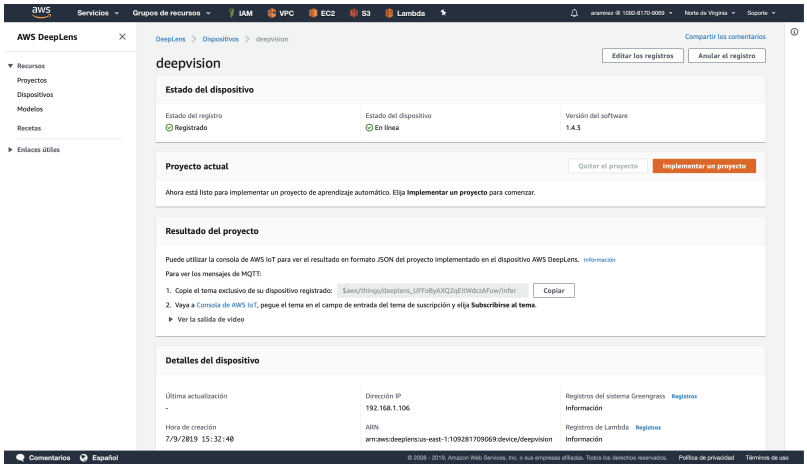


FIGURA 4.11. Deeplens activado

el dispositivo desde otra red WiFi, entonces hay que hacer todo el proceso de nuevo desde el principio.

4.2. Implementación de identificación de objetos en Deeplens. Una vez que el dispositivo está activo y en línea entonces es posible implementar los modelos. El proceso es realmente sencillo y consiste en escoger el modelo que se desea como en 4.12, se selecciona el o los dispositivos a los cuales se desea implementar el modelo, esto es interesante ya que se pueden desplegar modelos entrenados de forma masiva desde esta cónsola, como se ve en la imagen 4.13. El paso de confirmación es de los más importantes ya que se advierte que el

despliegue utiliza recursos que son pagos en la plataforma de Amazon y que se realizarán los cargos respectivos, se confirma la selección y se despliega como en 4.15.

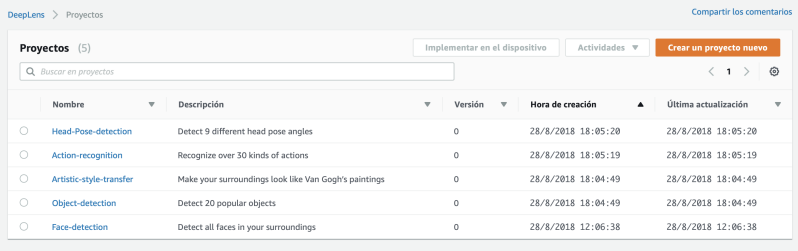


FIGURA 4.12. Selección de proyecto de detección de objetos

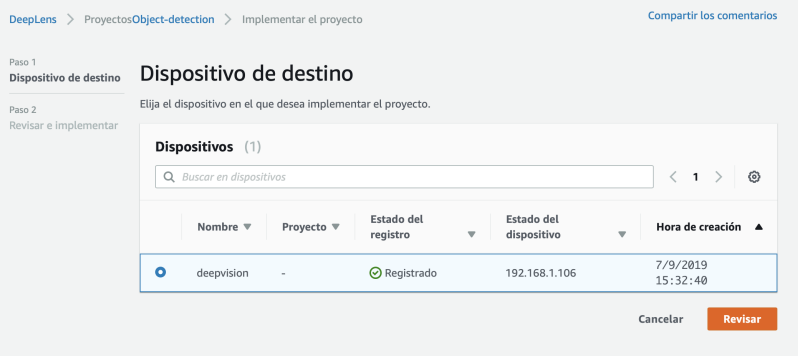


FIGURA 4.13. Despliegue del proyecto en los dispositivos seleccionados

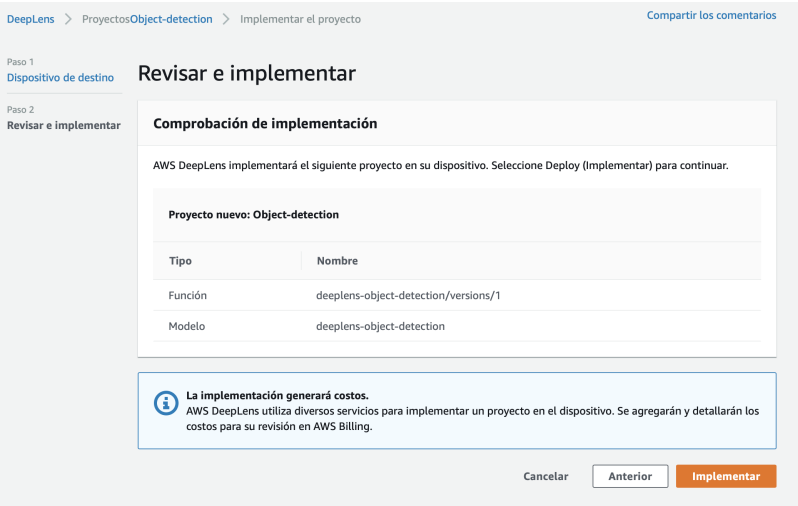


FIGURA 4.14. Confirmación de despliegue

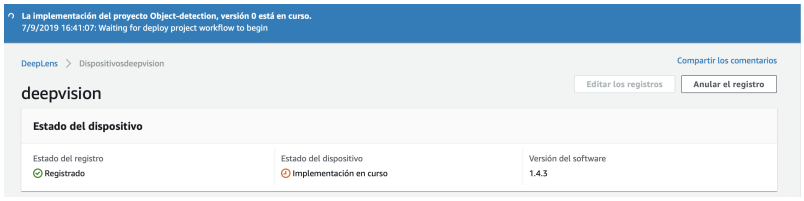


FIGURA 4.15. Proyecto desplegado en el dispositivo

Una vez que el modelo está implementado se puede ver el resultado del modelo en una pantalla conectada al dispositivo a través de una conexión en la nube. Hay una referencia para trabajar con proyectos tipo en Deeplens.

Todo este proceso está muy bien documentado en [23].

4.3. Explorar los casos de uso y métodos de ejemplo con enfoque didáctico. Hay diversos casos de uso que cruzan los servicios de Inteligencia Artificial ofrecidos por Amazon con otros servicios de infraestructura en la nube.

Los casos de uso que hemos visto implementados son los siguientes:

4.3.1. *Lectura de contenidos.* Hay una aplicación disponible que toma las imágenes de la cámara, las envía al servicio de reconocimiento de caracteres de Amazon para convertirla en texto y luego la convierte en audio que se reproduce en el dispositivo. Lo interesante es cómo se utilizan el resto de los servicios que hemos visto para ensamblar una solución.

4.3.2. *Juegos interactivos y educativos.* Se puede programar Deeplens para que reproduzca audio que realiza preguntas y las respuestas consisten en mostrar imágenes al dispositivo. De nuevo se combina la funcionalidad de texto a voz y además el reconocimiento de objetos en imágenes para ensamblar un juego interactivo.

4.3.3. *Seguridad.* El dispositivo se puede programar para reconocer rostros y al colocarlo en la entrada de la casa, si no reconoce el rostro del visitante entonces envía una alerta por SMS. El reconocimiento de rostros tiene muchas aplicaciones de seguridad y control, también es controversial su uso en espacios públicos.

Se pueden realizar aplicaciones que registran los visitantes, las entradas y salidas, entre otras aplicaciones.

4.3.4. *Análisis de sentimientos.* Se puede detectar el ánimo de las personas mediante sus rostros y a partir de ahí generar información sobre el ánimo de la audiencia en una conferencia.

4.3.5. *Mercadeo.* Una vez que se reconocen las personas entonces se muestra un anuncio que se adecue a sus gustos y necesidades.

Lo más interesante es que además de las posibilidades que ofrece la cámara, también se pueden integrar otros servicios que permiten ensamblar soluciones completas.

4.4. Aplicación de algoritmos de aprendizaje o inferencia própios o de terceros con Amazon SageMaker. Afortunadamente hay mucha documentación que describe los pasos necesarios para implementar un algoritmo propio y hemos conseguido varias fuentes de información que facilitan todo el proceso y permite, a cualquier persona interesada, avanzar en el uso de los servicios provistos por Amazon.

El modelo que hemos implementado en la cámara parte de un modelo pre entrenado provisto por **MxNet** denominado SSD: Single Shot MultiBox Object Detector. Este modelo utiliza ResNet como base y genera las cajas que aparecen en la imagen identificando los objetos detectados. Este es el modelo que se desplegó en el dispositivo. El resultado del modelo se puede apreciar en la imagen 4.16.

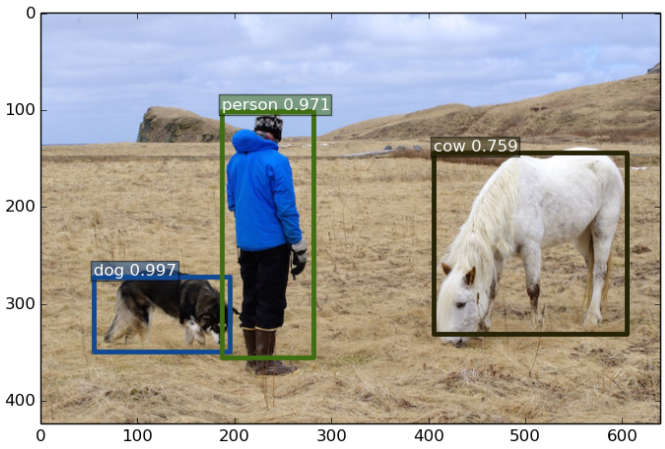


FIGURA 4.16. Detección de objetos

En el tutorial sobre detección de objetos se puede seguir paso a paso la implementación de un modelo en el dispositivo. Este proceso amerita la configuración de espacio de

almacenamiento y el uso de SageMaker para entrenar y luego desplegar el modelo. Finalmente se crea la estructura que permite desplegarlo en el dispositivo.

Dentro de la documentación que hemos revisado están los proyectos de ejemplo que hemos desplegado y que sirven de referencia para profundizar, además están los proyectos de ejemplo de SageMaker que muestran cada uno de los algoritmos disponibles y cómo se pueden utilizar.

MxNet cuenta con muchos ejemplos que sirven como punto de partida para implementar modelos y soluciones, además, como ya hemos mencionado, SageMaker permite la implementación de estos modelos en la nube. El repositorio principal también ofrece información muy útil para implementar modelos de **MxNet** en otros lenguajes como R, entre otros.

Cabe destacar que cada uno de estos algoritmos amerita el pago del almacenamiento de la entrada y la salida, el entrenamiento del modelo y el despliegue de un servicio web que realiza las inferencias. Hay que ser cuidadoso en el uso de estos servicios para evitar cargos adicionales.

Capítulo 5

Resultados y conclusiones

1. Resultados

1.1. Servicios de Inteligencia Artificial de Amazon. Es extensa y creciente la oferta de servicios de Amazon y muy completa para que nuestros modelos puedan pasar de un ejercicio académico a un servicio que otros pueden consumir en la Nube. Hemos visto adicionalmente que estos servicios se pueden combinar para conformar soluciones más complejas. Hemos conseguido una documentación extensa y muchos ejemplos que han servido para entender cómo se aplican estos modelos.

Amazon SageMaker sirve como plataforma para entrenar y publicar como servicio los modelos desarrollados, lo cual permite pasar del ambito académico al ambito industrial y de soluciones.

Hemos visto que los servicios tienen sus límites y de ahí se pueden derivar nuevas líneas de trabajo. Aunque hemos trabajado solo con Amazon, hemos visto que otros proveedores están desarrollando sus propias soluciones que bien vale la pena estudiar.

1.2. Ajuste de Redes Neuronales Convolucionales. Se desarrolló un cuaderno de notas en Python con el proceso de ajuste de una Red Neuronal Convolucional descrito paso a paso con finalidad didáctica. Se utilizó **keras** y esto facilitó de forma significativa el trabajo debido a que ya incorporan los conjuntos de datos con los cuales trabajamos, además utilizamos **scikit learn** para generar las matrices de confusión, permitiendo visualizar los datos que sirven para medir el desempeño del modelo ajustado.

1.2.1. *MNIST.* Con el conjunto de datos MNIST el entrenamiento que se hizo de la Red Neuronal Convolucional duró un minuto y medio por cada ciclo de entrenamiento, para un tiempo total de 20 minutos. El ajuste que se realizó alcanzó una exactitud de 99 % y se aplicó la técnica de regularización por descarte (dropout). Este conjunto de datos representa el primer paso para cualquier investigador que desea trabajar con modelos de aprendizaje para realizar regresión o clasificación supervisada o no supervisada.

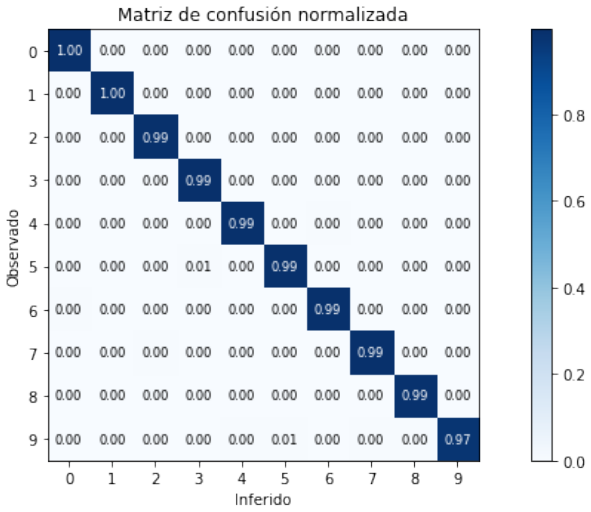


FIGURA 5.1. Matriz de confusión de las inferencias sobre los datos de prueba MNIST

Los modelos se almacenaron y no es necesario ajustarlos de nuevo si sólo se desea hacer un proceso de inferencia. Este hecho es muy ventajoso y ayuda a trabajar con el modelo sin necesidad de volver a ajustarlo.

En la matriz de confusión, en la imagen 5.1, se puede apreciar un nivel de error muy bajo para el ajuste de los datos MNIST.

1.2.2. *CIFAR.* Con el conjunto de datos CIFAR el entrenamiento duró 5 horas, ya que se hicieron 100 ciclos de entrenamiento con todos los datos. El ajuste que se realizó alcanzó una exactitud de 78 %. El proceso ha sido más lento aunque la cantidad de parámetros a ajustar son un millón doscientos cincuenta mil parámetros comparado con cerca de un millón doscientos mil en el caso de MNIST.

Los datos en el caso de CIFAR son más complejos ya que las imágenes son a color y más variadas y las clases representan un mayor reto por las similitudes entre algunas clases. Por ejemplo los animales de 4 patas se confunden entre sí, un carro se puede confundir con un camión, entre otros. Esto muestra la dificultad en el ajuste de este tipo de modelos y la necesidad de técnicas adicionales para mejorar el desempeño.

En la matriz de confusión sobre las predicciones del modelos ajustado con los datos CIFAR, que se pueden ver en la imagen 5.2, podemos visualizar los errores que comete el modelo. En particular, la clase de los gatos tiene el nivel de verdaderos positivos más bajo con 53 % y el modelo, partiendo de la foto de un gato, infiere que es un perro el 11 % de las

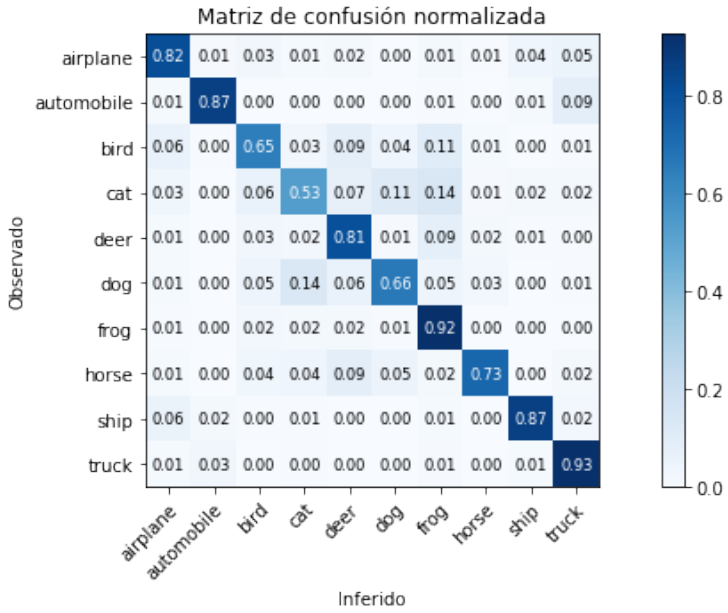


FIGURA 5.2. Matriz de confusión de las inferencias sobre los datos de prueba CIFAR

veces e infiere que es una rana el 14 % de las veces. Los siguientes más bajos son los pájaros y los perros. En las clases que tienen que ver con animales es que se comenten la mayor cantidad de errores.

1.3. Reconocimiento de objetos con Deeplens. Se implementó un modelo de reconocimiento de objetos en el dispositivo Deeplens y se estudiaron las características del modelo **SSD** que parte de la arquitectura **ResNet** para generar las inferencias. En las imágenes 5.3 y en 5.4 podemos apreciar el resultado de la inferencia realizada.

Trabajando con Amazon Deeplens hemos observado que es un kit para desarrolladores e investigadores, su desempeño y la capacidad del equipo no permite que sea utilizado en soluciones de misión crítica. Sin embargo sirve para los propósitos que nos hemos planteado.

Los objetos fueron reconocidos como las computadoras, personas y un perro. Sin embargo, confundió una hamaca recogida con una botella, lo cual da una idea de los límites de estos métodos.

Se implementó un modelo pre entrenado en Amazon Deeplens y se estudió la forma como se despliegan nuevos modelos, el modelo de negocios de Amazon genera cargos y nuevos pagos cada vez que se despliega un modelo en el dispositivo, fuimos bien conservadores para evitar gastos adicionales.

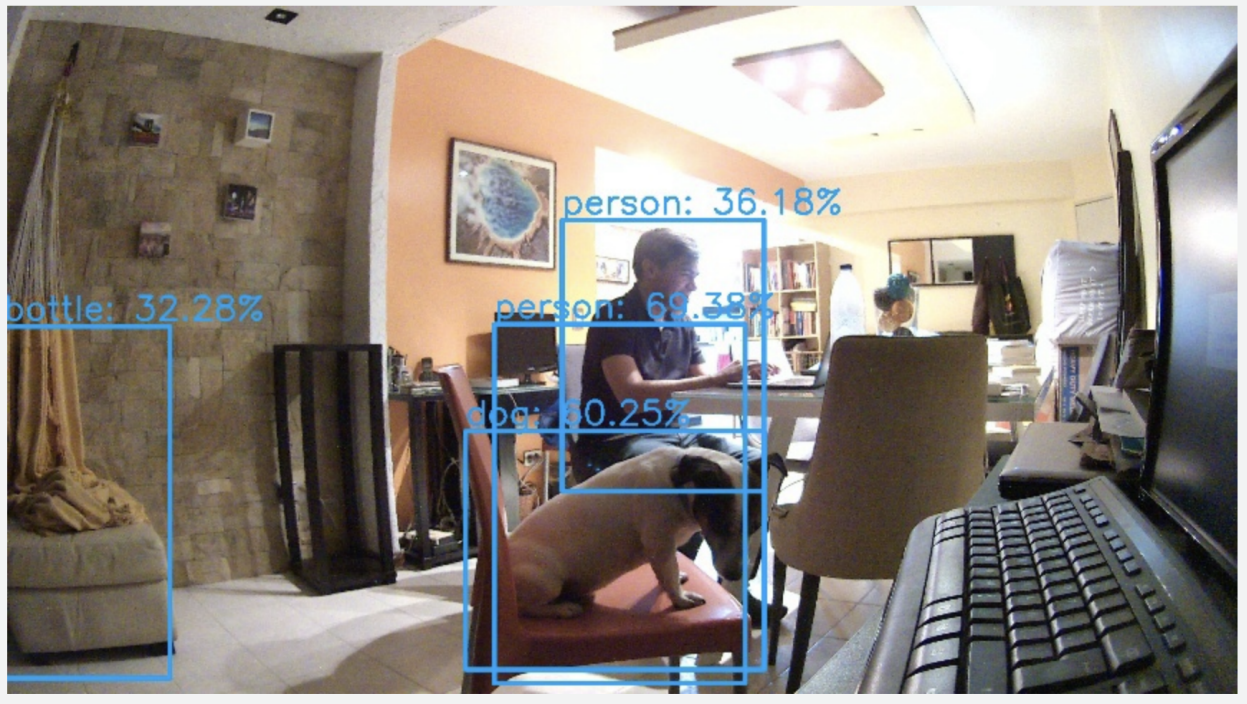


FIGURA 5.3. Primer ejemplo de reconocimiento de objetos con Deeplens

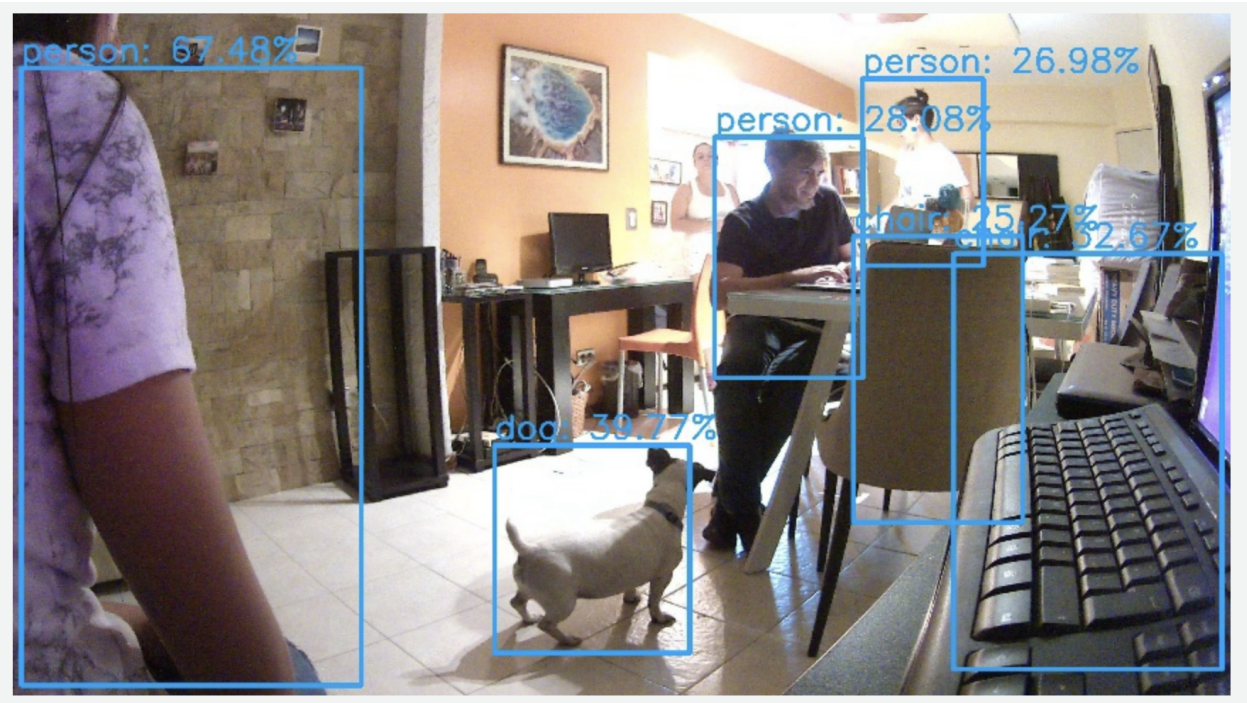


FIGURA 5.4. Segundo ejemplo de reconocimiento de objetos con Deeplens

2. Conclusiones

2.1. Servicios en la Nube. Los servicios ofrecidos por Amazon son un excelente punto de partida para entender el alcance y los límites de los métodos de Inteligencia Artificial y Aprendizaje Automatizado. Hay una brecha entre la práctica teórica/académica y la implementación de soluciones a problemas reales. Amazon ofrece una solución para salvar esta brecha, desplegando los modelos en la nube y publicándolos como servicios.

Hemos visto que hay límites a la práctica actual, en particular dimos un ejemplo con el análisis de sentimientos y el reconocimiento de la ironía. Estas brechas o límites identificados perfectamente pueden convertirse en líneas de investigación. Así que los proveedores importantes de tecnología y soluciones, son una fuente importante de conocimiento para aprovechar sus bondades y ampliar el alcance de nuestras soluciones y al mismo tiempo son una fuente de problemas de interés para la investigación.

2.2. Redes Neuronales y Redes Neuronales Convolucionales. Las Redes Neuronales y más recientemente las Redes Neuronales Convolucionales representan un movimiento renovado de investigación sobre los métodos de aprendizaje, optimización, regularización, cálculo, computación paralela, entre otros, y representan un avance decidido en la solución a muchos problemas en áreas diversas y en particular en el área de visión por computadora.

El uso de procesadores gráficos, nuevas funciones de activación y arquitecturas de capas innovadoras ha renovado el interés de los investigadores y se han desarrollado nuevos métodos para mejorar su desempeño.

Es claro el avance, pero también queda muy claro el reto. Los modelos requieren grandes cantidades de datos que son costosos para recolectar y procesar. El entrenamiento de Redes Neuronales es costoso en almacenamiento y en procesamiento, una vez ajustado un modelo entonces podemos almacenarlo y utilizarlo como base para generar soluciones.

Esta práctica se conoce como aprendizaje por transferencia, es decir, no se parte de un modelo nuevo sino desde un modelo pre entrenado. Nos imaginamos que es posible la creación de bases de datos de modelos pre entrenados y especializados que se podrán ensamblar en soluciones a diversos problemas. El diseño y creación de este tipo de bases de datos está

por verse en el futuro, por lo pronto sólo contamos con algunos modelos pre entrenados disponibles a través de los ambientes de trabajo.

El ajuste de modelos con imágenes sigue siendo interesante y retador, hemos visto que con arquitecturas similares pero conjuntos de datos distintos, el ajuste y la exactitud es diferente en cada problema y esto genera el reto de continuar trabajando en modelos de propósito general y ajustar modelos a como de lugar, para aumentar la exactitud y sobre ajustar un modelo para trabajar con un conjunto de datos específico.

En este sentido el ajuste de un modelo hoy en día depende fuertemente de los datos disponibles y todavía hace falta el ojo experto para ajustar las arquitecturas y los hiper parámetros de los modelos para llegar a resultados más favorables, con el riesgo o la ventaja de generar modelos especializados.

Nos parece interesante como tesis para trabajos futuros el ajuste de modelos óptimos especializados y su conformación en arquitecturas para resolver problemas de clasificación y detección de objetos en imágenes. Si contamos con un modelo de clasificación binario para cada objeto con una exactitud muy alta, en lugar de entrenar un modelo general, entrenamos dos modelos especializados y los combinamos en una arquitectura que le envía la entrada a cada modelo y cada uno produce un resultado. Nos parece que el entrenamiento de modelos especializados puede ser menos costoso e intensivo, el proceso actual es top-down y nuestra propuesta sería bottom-up.

Hay una gran variedad de opciones para trabajar modelos basados en Redes Neuronales, utilizamos **keras**, **scikit learn** y modelos de **MxNet** durante el trabajo. Afortunadamente hemos encontrado una gran diversidad de recursos para entender el funcionamiento de las Redes Neuronales y además implementarlas.

2.3. Estado del arte. Hay muchos avances en la creación de nuevos modelos que resuelven diversos problemas específicos. Es importante notar que los modelos que se han desarrollado son especializados y no son de propósito general en el sentido de clasificar millones de objetos. El proceso de aprendizaje es completamente dirigido y los modelos se ajustan a un costo importante. La necesidad de grandes cantidades de datos con mayor resolución y detalle y la gran capacidad de procesamiento requerida para ajustar modelos especializados muestra los límites y al mismo tiempo las posibilidades.

Los métodos de aprendizaje todavía siguen siendo un área de desarrollo utilizando métodos bayesianos como en [24] con enfoques no supervisados y que no requieren grandes cantidades de datos.

Por otra parte el trabajo de Goodfellow en [25] ha generado gran atención con un acercamiento innovador donde dos redes neuronales compiten, una con el objetivo de generar nuevos datos que simulan ser los originales y la otra red debe inferir si el dato provisto por la primera, es original o es un dato generado. El objetivo finalmente es ajustar la red para producir datos que se distribuyen como los originales.

El reto es y sigue siendo desarrollar modelos de inteligencia y aprendizaje generales, similares a las capacidades humanas, aunque hemos avanzado todavía estamos lejos de alcanzar la meta.

Sigue siendo importante la creación de nuevos conjuntos de datos para entrenamiento, las competencias de ImageNet ha generado un ambiente propicio para la investigación con conjuntos de datos y un problema claro. Generar bases de datos para propiciar la investigación es una muy buena idea de entrada.

El considerado abuelo del Aprendizaje Profundo Geoffrey Hinton ha dicho que está sospechando profundamente del método de propagación reversa y que para avanzar de forma importante se deben desarrollar nuevos métodos. Cita a Max Planck diciendo que,

> La ciencia progresa un funeral tras otro.

Para finalizar dice,

> El futuro depende de un estudiante que está sospechando profundamente de todo lo que he dicho.

Apéndice A

Ajuste de un modelo lineal con un Perceptrón simple y descenso del gradiente

Lineal

October 3, 2019

1 Ajuste de un modelo lineal

Este ejemplo muestra el proceso de ajuste de un modelo lineal con un perceptrón simple y como función de activación la función lineal.

El perceptrón recibe x y el sesgo b como entrada y se utiliza una tasa de aprendizaje de $0,05$. Los datos corresponden a la función lineal $f(x) = 2x - 10$. Queremos conseguir mediante el método del descenso del gradiente los dos parámetros $w1$ y $b1$.

Vamos a utilizar `numpy` y `pandas` para dibujar los gráficos.

```
In [1]: import numpy as np
        import pandas as pd
        %matplotlib inline
```

Los datos de entrada corresponden a la función lineal descrita y se utilizan 13 datos para entrenar el modelo.

```
In [2]: # Datos de entrada
        xList = [ -5,  -4,  -3,  -2,  -1,   0,  1,  2,  3,  4, 5, 6, 7]

        # Datos de salida (la función es 2*x - 10)
        yList = [-20, -18, -16, -14, -12, -10, -8, -6, -4, -2, 0, 2, 4]

        alpha = 0.05
```

Ahora vamos a definir las funciones del modelo.

- `linear` es la función de activación.
- `forward` es la fase de estimación que le aplica la función de activación a la entrada multiplicada por el parámetro y suma el sesgo.
- `backward` es la fase de ajuste del parámetro mediante el descenso del gradiente, que utiliza la tasa de aprendizaje `alpha` y el gradiente. El gradiente es la derivada de la función de error con respecto al parámetro, que en nuestro caso es la función de error cuadrático.
- `error` es la función que calcula el error cuadratico entre el valor estimado y el valor observado en los datos.

```
In [3]: def linear(x):
            return x
```

```
In [4]: def forward(x, w, b):
            yEst = linear(w*x + b)
            return yEst

In [5]: def backward(w, dE_w):
            return w - (alpha*dE_w)

In [6]: def error(yEst, y):
            return np.power(yEst-y, 2)
```

Ahora vamos a establecer los valores de las variables. En particular la condición de parada para el proceso de ajuste iterativo que es 10^{-20}. Los valores iniciales de los parámetros son arbitrarios. Definimos la lista donde se almacenarán los valores durante el ajuste para graficar su evolución. Se define la variable que indica el número máximo de iteraciones que en este caso es `1000` y una variable de ayuda para almacenar el tamaño de los datos para recorrerlos en un ciclo.

```
In [7]: stopAt = 1e-20

        w1 = 0.46
        b1 = 0.27

        w1List = [w1]
        b1List = [b1]
        errList = []

        numIter = 1000

        sizeOfData = np.size(xList)
```

Este es el ciclo principal que se ejecuta a lo sumo `numIter` veces. Se define una variable `index` para saber cual valor de los datos tomar y cual valor observado tomar para evaluar el error. Se realizan varias operación es de impresión para dar una idea de cómo va la ejecución del programa y el avance dentro del ciclo.

El proceso consiste en el ciclo siguiente:

- Realizar el paso de estimación denominado `forward`.
- Calcular el error cuadrático entre el valor estimado en la fase previa y el valor observado mediante la función `error`.
- Calcular los gradientes que consiste en calcular la derivada de la función de error con respecto a `w1` y con respecto a `b1`.
- Aplicar el paso de ajuste `backward`.
- Almacenar los valores obtenidos para graficar los resultados.

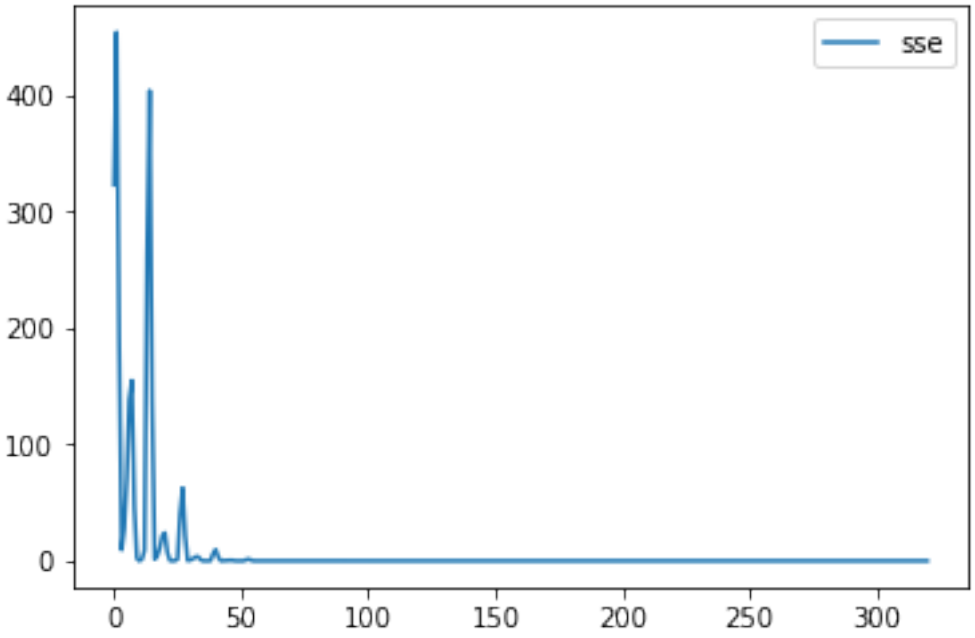
```
In [16]: echo=False
         for i in range(numIter):

             index = i%sizeOfData
             x = xList[index]
             y = yList[index]
```

```
        if echo:
            # Imprimir valores
            print('Iteración '+str(i+1)+str(' de ')+str(numIter))
            print('Datos: x'+str(index)+':'+str(x)+', w1:'+str(w1)+', b1:'+str(b1))

        # Cálculo de y Estimado
        yEst = forward(x, w1, b1)
        if echo:
            print('y:'+str(y)+', yEst:'+str(yEst))

        # Cálculo del Error
        sse = error(yEst, y)
        if echo:
            print('error '+str(sse))

        # Cálculo de las derivadas parciales
        dE_w1 = 2*(yEst-y)*x
        dE_b1 = 2*(yEst-y)
        if echo:
            print('d3_w1:'+str(dE_w1)+', dE_b1:'+str(dE_b1))

        # Ajustando los parámetros w1 y b1
        w1 = backward(w1, dE_w1)
        b1 = backward(b1, dE_b1)
        if echo:
            print('After adjusting')
            print('w1:'+str(w1)+', b1:'+str(b1))
            print('')

        # Guardar valores
        errList.append(sse)
        w1List.append(w1)
        b1List.append(b1)

        # Chequear condición de parada
        if (sse<stopAt):
            break
```

Para chequer el resultado se dibuja cómo evoluciona el error durante el proceso de estimación y ajuste.

```
In [17]: pd.DataFrame(errList, columns=['sse']).plot()
```

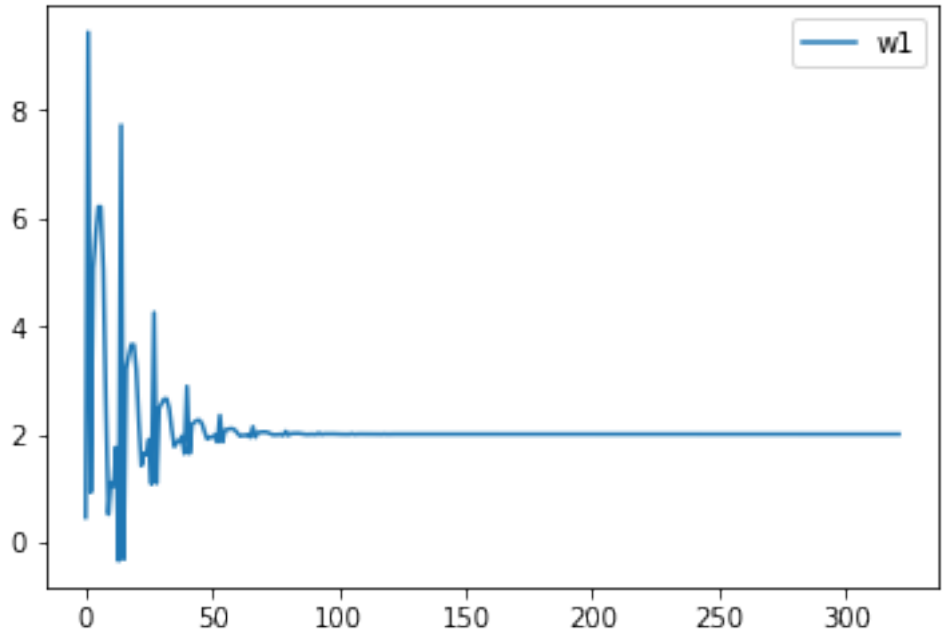
```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1192bc2d0>
```



Se dibuja cómo evoluciona el parámetro `w1`. Se puede ver que alrededor de la iteración 50 ya converge el proceso de aprendizaje.

```
In [18]: pd.DataFrame(w1List, columns=['w1']).plot()
```

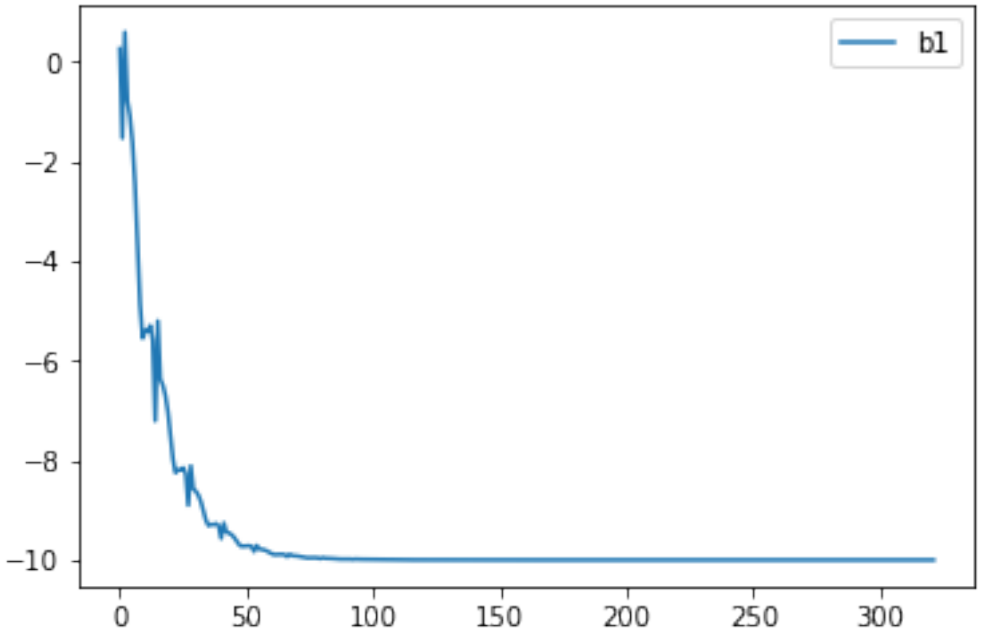
```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x11939e610>
```



Se dibuja el gráfico de cómo evoluciona el ajuste del sesgo.

```
In [19]: pd.DataFrame(b1List, columns=['b1']).plot()
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x119488d90>
```




```
In [20]: print(w1, b1)
```

```
(1.9999999997943871, -9.999999999022553)
```

Recordemos que los datos provienen de la función $f(x) = 2x - 10$ y los valores que se aprenden están cerca con un error de 10^{-20}.

A continuación el código sin imprimir ni almacenar datos.

```
In [21]: stopCond = 1e-20

         w1 = 0.46
         b1 = 0.27

         numIter = 1000

         sizeOfData = np.size(xList)
```

```
        for i in range(numIter):
            # Variables
            index = i%sizeOfData
            x = xList[index]
            y = yList[index]

            # Cálculo de y Estimado
            yEst = forward(x, w1, b1)

            # Cálculo del error
            sse = error(yEst, y)

            # Cálculo de las derivadas parciales del Error con
            # respecto a los parámetros
            dE_w1 = 2*(yEst-y)*x
            dE_b1 = 2*(yEst-y)

            # Ajustando los parámetros w1 y b1
            w1 = backward(w1, dE_w1)
            b1 = backward(b1, dE_b1)

            # Chequear condición de parada
            if (sse<stopCond):
                break
```

```
In [22]: print(w1, b1)
```

```
(1.9999999995335829, -9.999999997782723)
```

Apéndice B

Ajuste de una Red Neuronal Convolucional con MNIST en Keras

Ajuste de una Red Neuronal Convolucional con MNIST en Keras

October 6, 2019

1 MNIST en Keras

Partiendo de la base de datos MNIST de imágenes de dígitos escritos a mano, vamos a realizar el ajuste de una Red Neuronal Convolucional utilizando la librería Keras. El proceso se realizará en varios pasos:

- Incluir librerías y definir parámetros.
- Cargar los datos.
- Ajustar los datos.
- Definir el modelo en Keras.
- Ajustar el modelo.
- Evaluar el modelo.

Adicionalmente vamos crear la matriz de confusión.
Finalmente vamos a revisar algunas particularidades con los datos.

1.1 Inclusión de librerías

Principalmente se incluye keras y desde keras se pueden cargar los datos MNIST directamente. Vamos a utilizar scikit learn para hacer la matriz de confusión y matplotlib para realizar los gráficos.

```
In [1]: import keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten
        from keras.layers import Conv2D, MaxPooling2D
        from keras import backend as K

        from sklearn import metrics

        import numpy as np
        from matplotlib import pyplot as plt
        %matplotlib inline
```

```
Using TensorFlow backend.
```

1.2 Definición de parámetros

- `batch_size` define el tamaño del lote para entrenar.
- `num_classes` indica cuantas clases genera la clasificación.
- `epochs` indica el número de vueltas completas que se hacen con el conjunto de datos de entrenamiento.
- `img_rows`, `img_cols` indica el tamaño de las imágenes en pixeles.

```
In [2]: batch_size = 128
        num_classes = 10
        epochs = 12

        # Dimensión de las imágenes
        img_rows, img_cols = 28, 28
```

1.3 Obtención de los datos

Los datos se obtienen con `mnist.load_data()` provisto por keras y se dividen en muestra de entrenamiento y prueba.

```
In [3]: # Dividir los datos entre los datos de entrenamiento y prueba.
        (x_train, y_train), (x_test, y_test) = mnist.load_data()
        (x_data, y_data), (x_data_test, y_data_test) = mnist.load_data()

        if K.image_data_format() == 'channels_first':
            x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
            x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
            input_shape = (1, img_rows, img_cols)
        else:
            x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
            x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
            input_shape = (img_rows, img_cols, 1)
```

1.4 Normalización de los datos

Los valores de las imágenes se pasan de una escala entre 0 y 1. Son 60 mil imágenes de entrenamiento y 10 mil de prueba de dimensión 28×28 en escala de grises y por eso es sólo 1 canal.

```
In [4]: x_train = x_train.astype('float32')
        x_test = x_test.astype('float32')
        x_train /= 255
        x_test /= 255
        print('x_train shape:', x_train.shape)
        print(x_train.shape[0], 'train samples')
        print(x_test.shape[0], 'test samples')
```

```
('x_train shape:', (60000, 28, 28, 1))
(60000, 'train samples')
(10000, 'test samples')
```

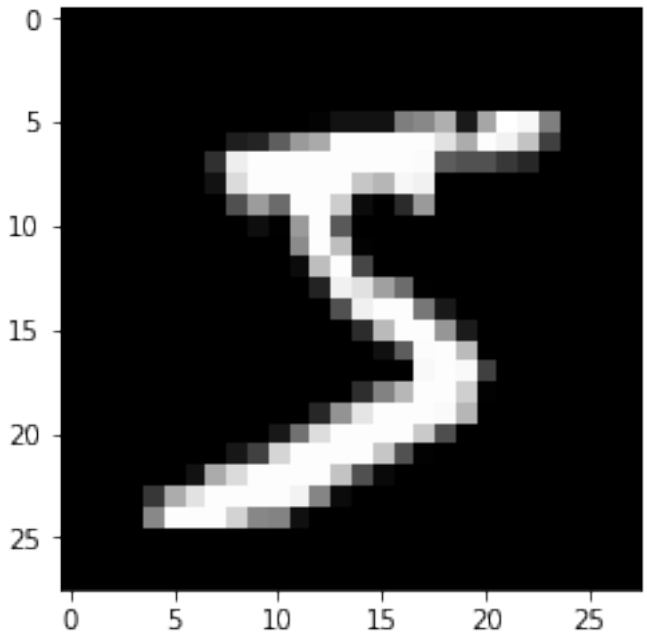
1.4.1 Rutina para dibujar un dígito

Vamos a definir una rutina para dibujar la imagen de un dígito. Se va a utilizar para explorar los datos y verificar los resultados de la inferencia realizada por el modelo.

```
In [5]: def plot_image(data):
            image = data
            image = np.array(image, dtype='float')
            pixels = image.reshape((28, 28))
            plt.imshow(pixels, cmap='gray')
            plt.show()
```

Vamos a mostrar la imagen del primer dígito de la muestra de entrenamiento.

```
In [6]: plot_image(x_train[0])
```



Ahora vamos a ver el valor del resultado observado en el primer dígito de la muestra de entrenamiento.

```
In [7]: y_train[0]
```

```
Out[7]: 5
```

Las clases se convierten en matrices binarias para utilizarlas con la función de pérdida `categorical_crossentropy`.

```
In [8]: y_train = keras.utils.to_categorical(y_train, num_classes)
        y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
In [9]: y_train[0]

Out[9]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

Podemos ver que el sexto valor es 1 indicando que la etiqueta de la primera imagen de la muestra de entrenamiento es un cinco. Recordemos que el vector es de 10 posiciones que inicia desde el 0.

Ahora vamos a crear el modelo con dos capas de convolución de 32 y 64 mapas de características respectivamente con kernel de 3×3 y ReLU como función de activación. Luego se aplica una capa de sub muestraje con ventana de 2×2 y finalmente una red reuronal conectada con 128 neuronas.

Para el modelo se utiliza la entropía, Adadelta como algoritmo de optimización y la métrica es la exactitud.

```
In [10]: model = Sequential()
         model.add(Conv2D(32, kernel_size=(3, 3),
                          activation='relu',
                          input_shape=input_shape))
         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))
         model.add(Flatten())
         model.add(Dense(128, activation='relu'))
         model.add(Dense(num_classes, activation='softmax'))

         model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adadelta(),
                       metrics=['accuracy'])
```

Ahora podemos inspeccionar el modelo y ver las características y parámetros.

```
In [11]: model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
```

```
Non-trainable params: 0
_________________________________________________________________
```

El modelo se ajusta a los datos de entrenamiento en lotes de 128 y se realizarán 12 ciclos, para luego validar con los datos de prueba.

```
In [12]: model.fit(x_train, y_train,
                   batch_size=batch_size,
                   epochs=epochs,
                   verbose=1,
                   validation_data=(x_test, y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 90s 1ms/step - loss: 0.2090 - acc: 0.9370 - val
Epoch 2/12
60000/60000 [==============================] - 92s 2ms/step - loss: 0.0495 - acc: 0.9851 - val
Epoch 3/12
60000/60000 [==============================] - 87s 1ms/step - loss: 0.0310 - acc: 0.9902 - val
Epoch 4/12
60000/60000 [==============================] - 91s 2ms/step - loss: 0.0210 - acc: 0.9933 - val
Epoch 5/12
60000/60000 [==============================] - 96s 2ms/step - loss: 0.0141 - acc: 0.9960 - val
Epoch 6/12
60000/60000 [==============================] - 97s 2ms/step - loss: 0.0099 - acc: 0.9972 - val
Epoch 7/12
60000/60000 [==============================] - 100s 2ms/step - loss: 0.0062 - acc: 0.9984 - va
Epoch 8/12
60000/60000 [==============================] - 100s 2ms/step - loss: 0.0044 - acc: 0.9988 - va
Epoch 9/12
60000/60000 [==============================] - 97s 2ms/step - loss: 0.0031 - acc: 0.9992 - val
Epoch 10/12
60000/60000 [==============================] - 94s 2ms/step - loss: 0.0023 - acc: 0.9994 - val
Epoch 11/12
60000/60000 [==============================] - 97s 2ms/step - loss: 0.0015 - acc: 0.9996 - val
Epoch 12/12
60000/60000 [==============================] - 95s 2ms/step - loss: 8.0593e-04 - acc: 0.9999 -


Out[12]: <keras.callbacks.History at 0x1a31eeeb90>
```

Luego del ajuste podemos salvar el modelo para utilizarlo sin realizar el ajuste de nuevo. Esta es una gran ventaja ya que se pueden compartir los resultados.

```
In [13]: model.save("MNIST-keras-no-drop.h5")
```

```
In [12]: from keras.models import load_model
         # Carga del modelo
```

```
model = load_model('MNIST-keras-no-drop.h5')
# Resumen de la estructura del modelo
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

Luego del ajuste que tomó alrededor de un minuto y medio por vuelta podemos ver el desempeño de 99%.

```
In [13]: score = model.evaluate(x_test, y_test, verbose=0)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
('Test loss:', 0.04357766299277946)
('Test accuracy:', 0.9902)
```

Si adicionalmente aplicamos `dropout` el modelo mejora ligeramente.

```
In [14]: model2 = Sequential()
         model2.add(Conv2D(32, kernel_size=(3, 3),
                           activation='relu',
                           input_shape=input_shape))
         model2.add(Conv2D(64, (3, 3), activation='relu'))
         model2.add(MaxPooling2D(pool_size=(2, 2)))
         model2.add(Dropout(0.25))
         model2.add(Flatten())
         model2.add(Dense(128, activation='relu'))
         model2.add(Dropout(0.5))
         model2.add(Dense(num_classes, activation='softmax'))
```

```
        model2.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adadelta(),
                       metrics=['accuracy'])
```

```
In [18]: model2.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 103s 2ms/step - loss: 0.2684 - acc: 0.9188 - val
Epoch 2/12
60000/60000 [==============================] - 107s 2ms/step - loss: 0.0896 - acc: 0.9742 - val
Epoch 3/12
60000/60000 [==============================] - 108s 2ms/step - loss: 0.0661 - acc: 0.9805 - val
Epoch 4/12
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0532 - acc: 0.9842 - val
Epoch 5/12
60000/60000 [==============================] - 106s 2ms/step - loss: 0.0485 - acc: 0.9857 - val
Epoch 6/12
60000/60000 [==============================] - 104s 2ms/step - loss: 0.0427 - acc: 0.9868 - val
Epoch 7/12
60000/60000 [==============================] - 106s 2ms/step - loss: 0.0380 - acc: 0.9887 - val
Epoch 8/12
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0332 - acc: 0.9895 - val
Epoch 9/12
60000/60000 [==============================] - 102s 2ms/step - loss: 0.0314 - acc: 0.9902 - val
Epoch 10/12
60000/60000 [==============================] - 98s 2ms/step - loss: 0.0302 - acc: 0.9909 - val_
Epoch 11/12
60000/60000 [==============================] - 99s 2ms/step - loss: 0.0270 - acc: 0.9919 - val_
Epoch 12/12
60000/60000 [==============================] - 102s 2ms/step - loss: 0.0257 - acc: 0.9918 - val
```

```
Out[18]: <keras.callbacks.History at 0x1a43d89b10>
```

```
In [20]: model2.save("MNIST-keras.h5")
```

Ahora podemos cargar el modelo salvado para realizar inferencias. Lo interesante es que una vez que el modelo está ajustado, lo podemos utilizar para inferir. De esta forma se pueden crear bases de datos de modelos pre entrenados que permiten realizar inferencias. Es como contar con los parámetros de un programa que ahora se puede ejecutar en cualquier lugar.

```
In [15]: from keras.models import load_model
         # Cargar modelo
```

```
model = load_model('MNIST-keras.h5')
# Resumen de la estructura del modelo
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               1179776
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

```
In [16]: score = model.evaluate(x_test, y_test, verbose=0)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
('Test loss:', 0.028851404254680527)
('Test accuracy:', 0.9907)
```

Vamos a almacenar en `y_pred` los valores inferidos de los datos de prueba `x_test`.

```
In [17]: y_pred = model.predict(x_test)
```

Con las inferencias ahora podemos hacer la matriz de confusión y a partir de esta generar métricas adicionales.

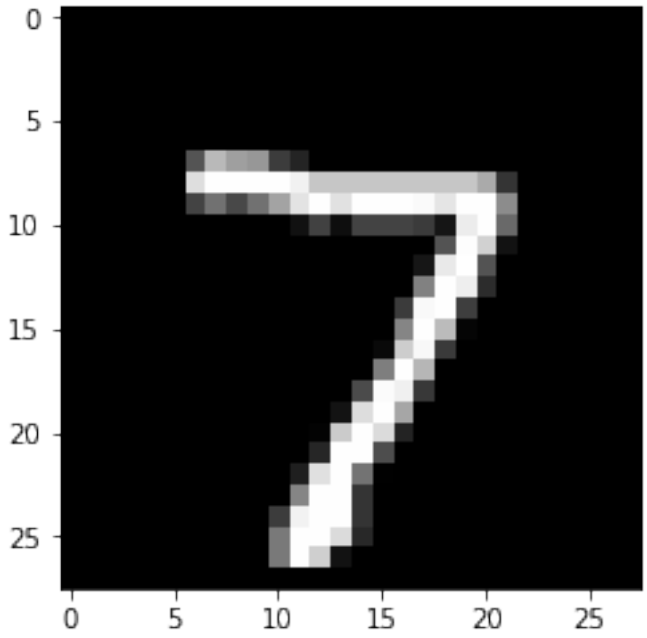
```
In [18]: matrix = metrics.confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
         matrix
```

```
Out[18]: array([[ 977,    0,    0,    0,    0,    0,    1,    1,    1,    0],
                [   0, 1131,    1,    1,    0,    0,    2,    0,    0,    0],
```

```
[   1,    2, 1023,    0,    1,    0,    1,    4,    0,    0],
[   0,    0,    2, 1004,    0,    3,    0,    0,    1,    0],
[   0,    0,    0,    0,  974,    0,    4,    0,    1,    3],
[   1,    0,    0,    5,    0,  884,    2,    0,    0,    0],
[   4,    2,    0,    0,    1,    3,  948,    0,    0,    0],
[   0,    1,    5,    1,    0,    0,    0, 1020,    1,    0],
[   2,    0,    3,    1,    1,    1,    1,    1,  963,    1],
[   3,    2,    0,    1,    5,    6,    1,    4,    4,  983]])
```

Vamos a probar con el primer dato de prueba.

```
In [19]: plot_image(x_test[0])
```



Observemos el resultado esperado.

```
In [20]: y_test[0]
```

```
Out[20]: array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

Ahora comparamos con el valor inferido.

```
In [21]: y_pred[0]
```

```
Out[21]: array([8.0566040e-12, 1.7861893e-10, 1.9713897e-09, 5.6523755e-09,
                1.5563544e-12, 4.1730020e-13, 4.4064523e-16, 1.0000000e+00,
                1.0673628e-12, 1.8242463e-09], dtype=float32)
```

Podemos corroborar que coincide.

```
In [22]: np.where(y_pred[0]==np.amax(y_pred[0]))[0][0]
```

```
Out[22]: 7
```

1.5 Matriz de confusión

Para dibujar la matriz de confusión se utiliza la rutina confusion_matrix de scikit-learn.

A esta rutina se le pasan los datos de prueba y las predicciones generadas por el modelo.

```
In [23]: matrix = metrics.confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
         matrix
```

```
Out[23]: array([[ 977,    0,    0,    0,    0,    0,    1,    1,    1,    0],
                [   0, 1131,    1,    1,    0,    0,    2,    0,    0,    0],
                [   1,    2, 1023,    0,    1,    0,    1,    4,    0,    0],
                [   0,    0,    2, 1004,    0,    3,    0,    0,    1,    0],
                [   0,    0,    0,    0,  974,    0,    4,    0,    1,    3],
                [   1,    0,    0,    5,    0,  884,    2,    0,    0,    0],
                [   4,    2,    0,    0,    1,    3,  948,    0,    0,    0],
                [   0,    1,    5,    1,    0,    0,    0, 1020,    1,    0],
                [   2,    0,    3,    1,    1,    1,    1,    1,  963,    1],
                [   3,    2,    0,    1,    5,    6,    1,    4,    4,  983]])
```

Para normalizar los datos de la matriz de confusión se normaliza cada fila.

```
In [24]: matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
```

```
Out[24]: array([[9.96938776e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
                 0.00000000e+00, 0.00000000e+00, 1.02040816e-03, 1.02040816e-03,
                 1.02040816e-03, 0.00000000e+00],
                [0.00000000e+00, 9.96475771e-01, 8.81057269e-04, 8.81057269e-04,
                 0.00000000e+00, 0.00000000e+00, 1.76211454e-03, 0.00000000e+00,
                 0.00000000e+00, 0.00000000e+00],
                [9.68992248e-04, 1.93798450e-03, 9.91279070e-01, 0.00000000e+00,
                 9.68992248e-04, 0.00000000e+00, 9.68992248e-04, 3.87596899e-03,
                 0.00000000e+00, 0.00000000e+00],
                [0.00000000e+00, 0.00000000e+00, 1.98019802e-03, 9.94059406e-01,
                 0.00000000e+00, 2.97029703e-03, 0.00000000e+00, 0.00000000e+00,
                 9.90099010e-04, 0.00000000e+00],
                [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
                 9.91853360e-01, 0.00000000e+00, 4.07331976e-03, 0.00000000e+00,
                 1.01832994e-03, 3.05498982e-03],
                [1.12107623e-03, 0.00000000e+00, 0.00000000e+00, 5.60538117e-03,
                 0.00000000e+00, 9.91031390e-01, 2.24215247e-03, 0.00000000e+00,
                 0.00000000e+00, 0.00000000e+00],
                [4.17536534e-03, 2.08768267e-03, 0.00000000e+00, 0.00000000e+00,
                 1.04384134e-03, 3.13152401e-03, 9.89561587e-01, 0.00000000e+00,
                 0.00000000e+00, 0.00000000e+00],
                [0.00000000e+00, 9.72762646e-04, 4.86381323e-03, 9.72762646e-04,
                 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 9.92217899e-01,
                 9.72762646e-04, 0.00000000e+00],
                [2.05338809e-03, 0.00000000e+00, 3.08008214e-03, 1.02669405e-03,
                 1.02669405e-03, 1.02669405e-03, 1.02669405e-03, 1.02669405e-03,
```

```
       9.88706366e-01, 1.02669405e-03],
      [2.97324083e-03, 1.98216056e-03, 0.00000000e+00, 9.91080278e-04,
       4.95540139e-03, 5.94648167e-03, 9.91080278e-04, 3.96432111e-03,
       3.96432111e-03, 9.74231913e-01]])
```

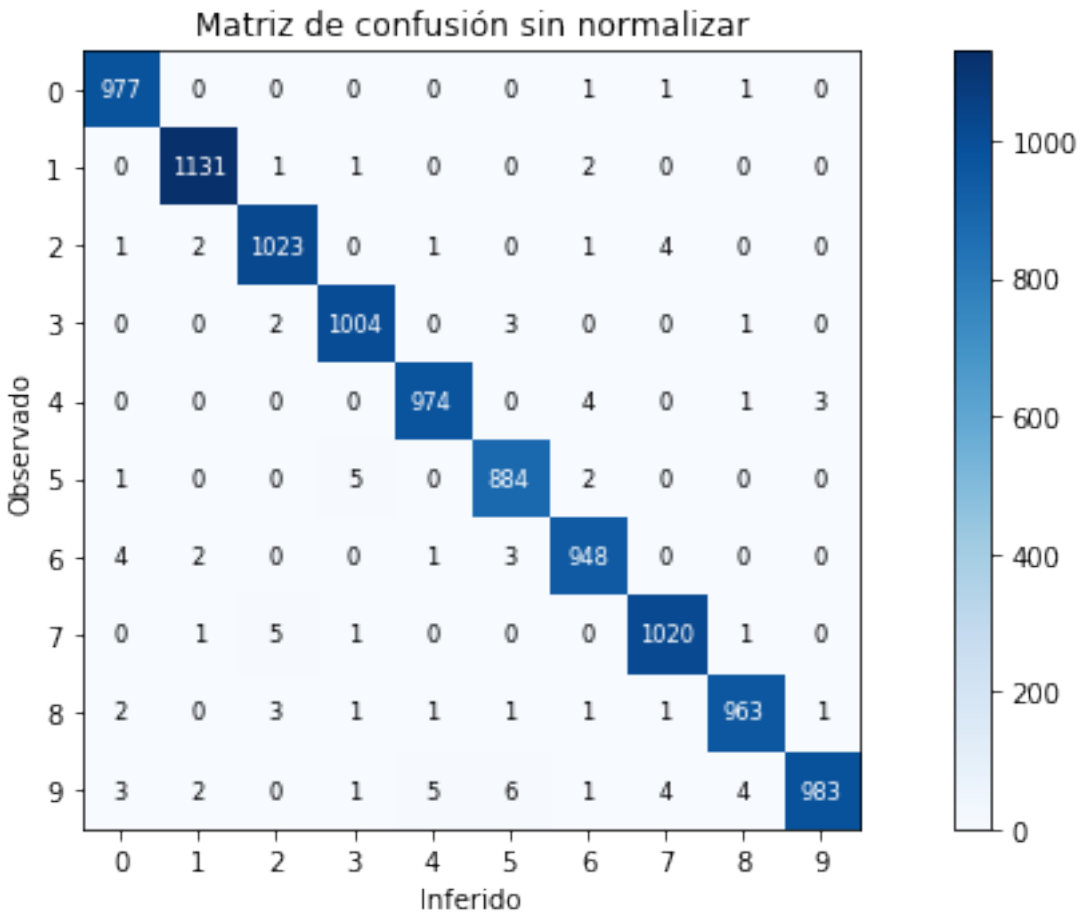
Hemos modificado el ejemplo para mostrar la matriz de confusión del ajuste con los datos MNIST.

```
In [25]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import confusion_matrix
         from sklearn.utils.multiclass import unique_labels

         class_names = np.array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'])

         def plot_confusion_matrix(y_true, y_pred, classes,
                                   normalize=False,
                                   title=None,
                                   cmap=plt.cm.Blues,
                                   printtext=False):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if not title:
                 if normalize:
                     title = 'Matriz de confusión normalizada'
                 else:
                     title = 'Matriz de confusión sin normalizar'

             # Compute confusion matrix
             cm = confusion_matrix(y_true.argmax(axis=1), y_pred.argmax(axis=1))
             # Only use the labels that appear in the data
             classes = classes[unique_labels(y_true)]

             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

             if printtext:
                 if normalize:
                     print("Matriz de confusión normalizada")
                 else:
                     print('Matriz de confusión sin normalizar')
                 print(cm)

             fig, ax = plt.subplots()
             im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
             ax.figure.colorbar(im, ax=ax)
             fig.set_size_inches(10, 5)
```

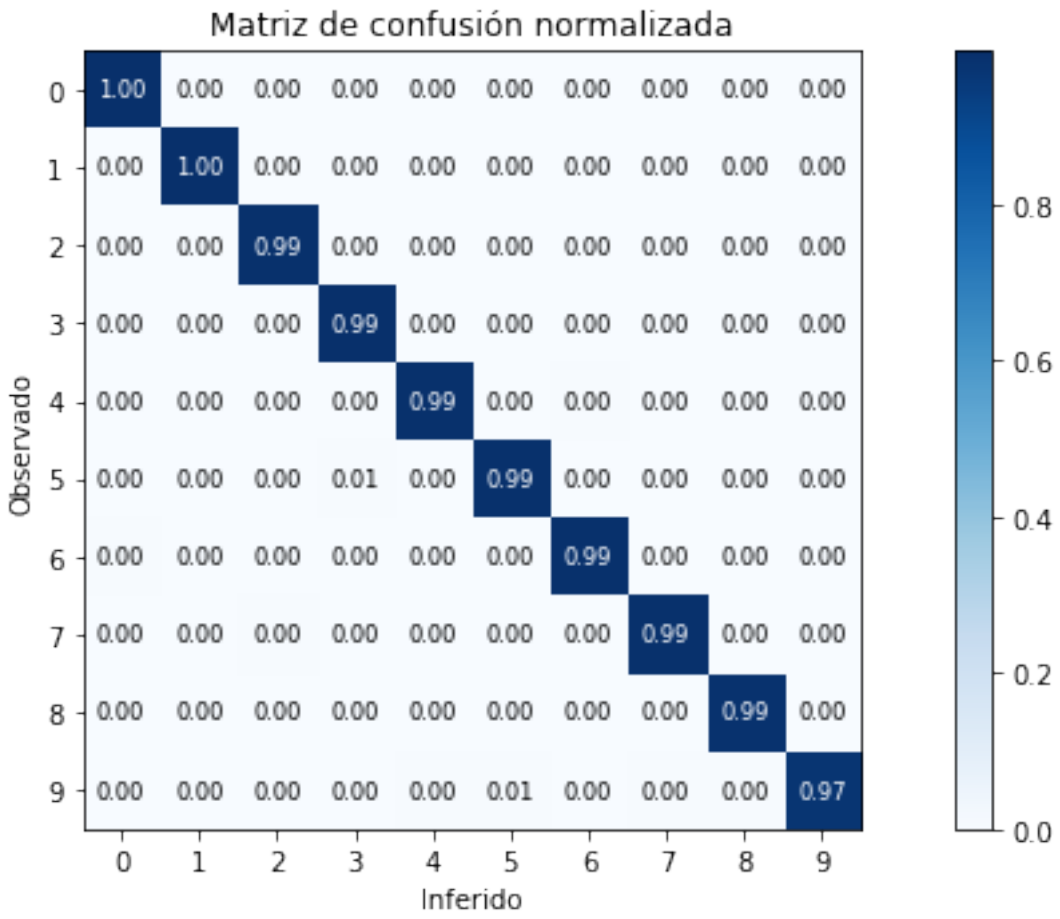
```
        # We want to show all ticks...
        ax.set(xticks=np.arange(cm.shape[1]),
               yticks=np.arange(cm.shape[0]),
               # ... and label them with the respective list entries
               xticklabels=classes, yticklabels=classes,
               title=title,
               ylabel='Observado',
               xlabel='Inferido')

        # Loop over data dimensions and create text annotations.
        fmt = '.2f' if normalize else 'd'
        thresh = cm.max() / 2.
        for i in range(cm.shape[0]):
            for j in range(cm.shape[1]):
                ax.text(j, i, format(cm[i, j], fmt),
                        fontsize='smaller',
                        ha="center", va="center",
                        color="white" if cm[i, j] > thresh else "black")
        fig.tight_layout()
        return ax
```

Ahora podemos ver de forma visual la matriz de confusión normalizada o sin normalizar.

```
In [26]: np.set_printoptions(precision=2)
         # Graficar la matriz de confusión sin normalizar
         plot_confusion_matrix(y_test, y_pred, classes=class_names,
                               title=u"Matriz de confusi\u00f3n sin normalizar")
         plt.show()
```



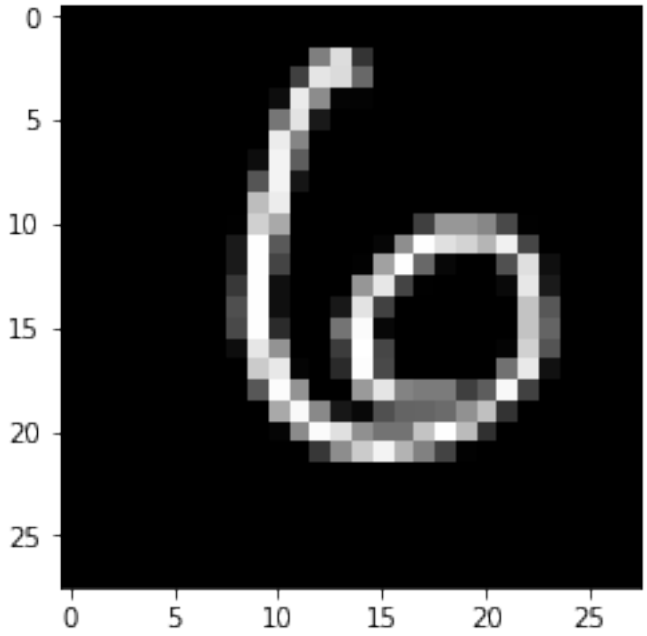
In [27]:
```
np.set_printoptions(precision=0)
# Matriz de confusión normalizada
plot_confusion_matrix(y_test, y_pred, classes=class_names, normalize=True,
                      title=u"Matriz de confusi\u00f3n normalizada")

plt.show()
```



1.6 Errores del modelo

Encontramos un ejemplo donde el modelo no genera el resultado esperado y en lugar de inferir un 6 infiere un 0.

```
In [28]: plot_image(x_test[100])
```




```
In [29]: y_test[100]
```

```
Out[29]: array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32)
```

```
In [30]: result=model.predict(np.expand_dims(x_test[100], axis=0))
         result
```

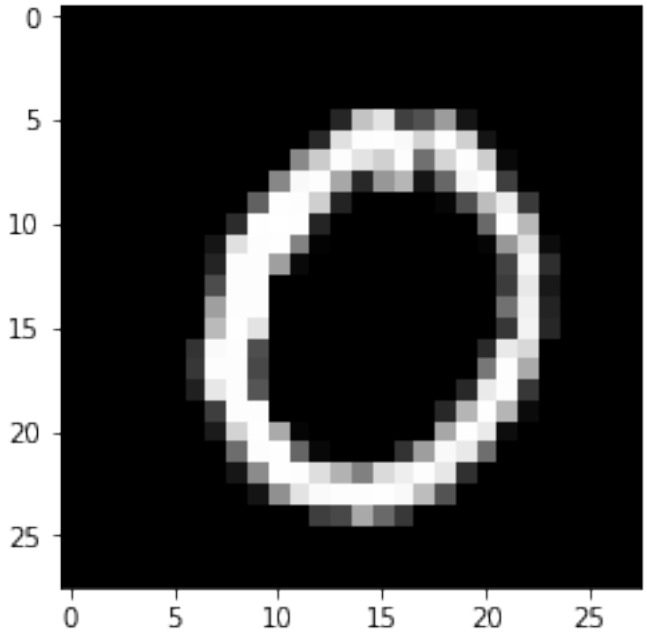
```
Out[30]: array([[4.e-11, 7.e-15, 6.e-17, 1.e-17, 1.e-13, 2.e-11, 1.e+00, 5.e-17,
                 7.e-13, 9.e-18]], dtype=float32)
```

```
In [31]: np.where(result==np.amax(result))[0][0]
```

```
Out[31]: 0
```

1.6.1 Otra prueba con un 0

```
In [32]: plot_image(x_test[101])
```




```
In [33]: y_test[101]
```

```
Out[33]: array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
In [34]: result=model.predict(np.expand_dims(x_test[101], axis=0))
         result
```

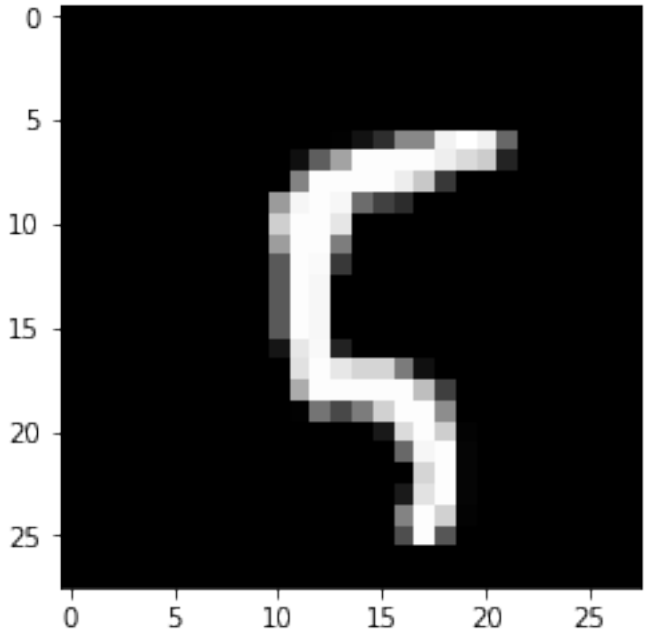
```
Out[34]: array([[1.e+00, 1.e-14, 3.e-13, 4.e-14, 9.e-16, 2.e-11, 3.e-07, 1.e-12,
                 7.e-11, 3.e-10]], dtype=float32)
```

```
In [35]: np.where(result==np.amax(result))[0][0]
```

```
Out[35]: 0
```

Ejemplo con figura incompleta

```
In [36]: plot_image(x_train[100])
```




```
In [37]: y_train[100]

Out[37]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)

In [38]: number=np.expand_dims(x_train[100], axis=0)

In [39]: arr=model.predict(number)[0]
         arr

Out[39]: array([8.e-08, 7.e-09, 2.e-11, 4.e-08, 9.e-09, 1.e+00, 5.e-05, 9.e-10,
                6.e-06, 6.e-06], dtype=float32)

In [40]: np.where(arr==np.amax(arr))[0][0]

Out[40]: 5
```

Apéndice C

Ajuste de una Red Neuronal Convolucional con CIFAR-10 en Keras

Ajuste de una Red Neuronal Convolucional con CIFAR10 en Keras

October 6, 2019

1 CIFAR10 en Keras

Vamos a realizar el ajuste de una Red Neuronal Convolucional utilizando la librería Keras. El proceso se realizará en varios pasos:

- Incluir librerías y definir parámetros.
- Cargar los datos.
- Ajustar los datos.
- Definir el modelo en Keras.
- Ajustar el modelo.
- Evaluar el modelo.

Adicionalmente vamos crear la matriz de confusión.
Finalmente vamos a revisar algunas particularidades con los datos.

1.1 Inclusión de librerías

Principalmente se incluye keras y desde keras se pueden cargar los datos CIFAR directamente. Se va a utilizar scikit learn para hacer la matriz de confusión y matplotlib para realizar los gráficos.

```
In [1]: from __future__ import print_function
        import keras
        from keras.datasets import cifar10
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten
        from keras.layers import Conv2D, MaxPooling2D
        import os

        from sklearn import metrics

        import numpy as np
        from matplotlib import pyplot as plt
        %matplotlib inline
```

```
Using TensorFlow backend.
```

1.2 Definición de parámetros

Se definen los parámetros para realizar el proceso de entrenamiento.

- `batch_size` es el tamaño del lote que se va a procesar.
- `num_classes` es el número de clases que existen en los datos.
- `epochs` es el número de ciclos de entrenamiento que se hace con todos los datos.
- `class_names` es el nombre de los objetos de cada clase.

```
In [2]: batch_size = 32
        num_classes = 10
        epochs = 50
        class_names = np.array(['airplane', 'automobile', 'bird', 'cat',
                                'deer', 'dog', 'frog', 'horse', 'ship', 'truck'])
```

Hemos creado la función `get_name` para obtener la etiqueta correspondiente a algún dato.

```
In [3]: def get_name(data, index):
            return class_names[data[index]][0]
```

1.3 Obtención de los datos

Se cargan con `cifar10.load_data()` y se dividen es conjunto de entrenamiento y prueba. Recordemos que son 50 mil muestras de entrenamiento y 10 mil muestras de prueba. Cada imagen es de 32×32 pixeles y hay tres canales ya que son a color.

```
In [4]: # División de los datos en datos de entrenamiento y prueba
        (x_train, y_train), (x_test, y_test) = cifar10.load_data()
        print('x_train shape:', x_train.shape)
        print(x_train.shape[0], 'train samples')
        print(x_test.shape[0], 'test samples')
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

1.3.1 Rutina para dibujar una imagen

Vamos a definir una rutina para dibujar una imagen del conjunto de datos. Se va a utilizar para explorar los datos y verificar los resultados de la inferencia realizada por el modelo.

```
In [5]: def plot_image(data, title):
            plt.axis('off')
            plt.title(title)
            plt.imshow(data)
            plt.show()
```

```
In [6]: plot_image(x_train[0], get_name(y_train, 0))
```


```
In [7]: y_train[0]
```

```
Out[7]: array([6], dtype=uint8)
```

Vamos a mostrar las primeras 9 imágenes del conjunto de datos de entrenamiento.

```
In [8]: fig, ax = plt.subplots()
        fig.set_size_inches(6, 6)

        for i in range(9):
            plt.subplot(330 + 1 + i)
            plt.axis('off')
            plt.imshow(x_train[i])
            plt.title(get_name(y_train, i))
        plt.show()
```

Las clases se convierten en matrices binarias para utilizarlas con la función de pérdida `categorical_crossentropy`.

```
In [9]: y_train = keras.utils.to_categorical(y_train, num_classes)
        y_test = keras.utils.to_categorical(y_test, num_classes)
```

1.4 Creación del modelo

Ahora vamos a crear el modelo con cuatro Capas de Convolución de 32, 32, 64 y 64 mapas de características respectivamente con kernel de 3×3 y ReLU como función de activación. Primero se aplican dos Capas de Convolución y se aplica una Capa de Sub muestraje con una ventana de 2×2 y finalmente una red reuronal conectada con 512 neuronas.

Para el modelo se utiliza la entropía, RMS como algoritmo de optimización y la métrica es la exactitud.

```
In [10]: model = Sequential()
         model.add(Conv2D(32, kernel_size=(3, 3),
                          padding='same',
```

```
                 activation='relu',
                 input_shape=x_train.shape[1:]))
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=(3, 3),
                 padding='same',
                 activation='relu'))
model.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

Ahora podemos corroborar el modelo.

```
In [11]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
```

```
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
```

Antes de pasar los datos al modelo se normalizan los valores.

```
In [12]: x_train = x_train.astype('float32')
         x_test = x_test.astype('float32')
         x_train /= 255
         x_test /= 255
```

1.5 Ajustar el modelo

Ahora se realiza el ajuste de modelo de acuerdo a los parámetros definidos previamente. En particular se van a realizar 10 pasadas a todos los datos para realizar el entrenamiento.

```
In [16]: model.fit(x_train, y_train,
                   batch_size=batch_size,
                   epochs=epochs,
                   validation_data=(x_test, y_test),
                   shuffle=True)

Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 199s 4ms/step - loss: 0.6370 - acc: 0.7866 - val
Epoch 2/50
50000/50000 [==============================] - 204s 4ms/step - loss: 0.6374 - acc: 0.7876 - val
Epoch 3/50
50000/50000 [==============================] - 200s 4ms/step - loss: 0.6361 - acc: 0.7885 - val
Epoch 4/50
50000/50000 [==============================] - 196s 4ms/step - loss: 0.6365 - acc: 0.7888 - val
Epoch 5/50
50000/50000 [==============================] - 202s 4ms/step - loss: 0.6314 - acc: 0.7878 - val
Epoch 6/50
50000/50000 [==============================] - 200s 4ms/step - loss: 0.6339 - acc: 0.7892 - val
Epoch 7/50
50000/50000 [==============================] - 201s 4ms/step - loss: 0.6359 - acc: 0.7890 - val
Epoch 8/50
```

```
50000/50000 [==============================] - 196s 4ms/step - loss: 0.6290 - acc: 0.7910 - val
Epoch 9/50
50000/50000 [==============================] - 204s 4ms/step - loss: 0.6313 - acc: 0.7891 - val
Epoch 10/50
50000/50000 [==============================] - 202s 4ms/step - loss: 0.6326 - acc: 0.7900 - val
Epoch 11/50
50000/50000 [==============================] - 196s 4ms/step - loss: 0.6286 - acc: 0.7905 - val
Epoch 12/50
50000/50000 [==============================] - 195s 4ms/step - loss: 0.6328 - acc: 0.7896 - val
Epoch 13/50
50000/50000 [==============================] - 203s 4ms/step - loss: 0.6264 - acc: 0.7911 - val
Epoch 14/50
50000/50000 [==============================] - 200s 4ms/step - loss: 0.6258 - acc: 0.7908 - val
Epoch 15/50
50000/50000 [==============================] - 198s 4ms/step - loss: 0.6308 - acc: 0.7918 - val
Epoch 16/50
50000/50000 [==============================] - 205s 4ms/step - loss: 0.6268 - acc: 0.7903 - val
Epoch 17/50
50000/50000 [==============================] - 196s 4ms/step - loss: 0.6252 - acc: 0.7936 - val
Epoch 18/50
50000/50000 [==============================] - 194s 4ms/step - loss: 0.6304 - acc: 0.7907 - val
Epoch 19/50
50000/50000 [==============================] - 198s 4ms/step - loss: 0.6293 - acc: 0.7910 - val
Epoch 20/50
50000/50000 [==============================] - 222s 4ms/step - loss: 0.6263 - acc: 0.7940 - val
Epoch 21/50
50000/50000 [==============================] - 203s 4ms/step - loss: 0.6279 - acc: 0.7922 - val
Epoch 22/50
50000/50000 [==============================] - 203s 4ms/step - loss: 0.6344 - acc: 0.7907 - val
Epoch 23/50
50000/50000 [==============================] - 201s 4ms/step - loss: 0.6333 - acc: 0.7894 - val
Epoch 24/50
50000/50000 [==============================] - 204s 4ms/step - loss: 0.6320 - acc: 0.7924 - val
Epoch 25/50
50000/50000 [==============================] - 202s 4ms/step - loss: 0.6322 - acc: 0.7931 - val
Epoch 26/50
50000/50000 [==============================] - 193s 4ms/step - loss: 0.6253 - acc: 0.7918 - val
Epoch 27/50
50000/50000 [==============================] - 200s 4ms/step - loss: 0.6312 - acc: 0.7898 - val
Epoch 28/50
50000/50000 [==============================] - 196s 4ms/step - loss: 0.6289 - acc: 0.7904 - val
Epoch 29/50
50000/50000 [==============================] - 202s 4ms/step - loss: 0.6348 - acc: 0.7891 - val
Epoch 30/50
50000/50000 [==============================] - 197s 4ms/step - loss: 0.6267 - acc: 0.7920 - val
Epoch 31/50
50000/50000 [==============================] - 196s 4ms/step - loss: 0.6337 - acc: 0.7914 - val
Epoch 32/50
```

```
50000/50000 [==============================] - 200s 4ms/step - loss: 0.6353 - acc: 0.7900 - val
Epoch 33/50
50000/50000 [==============================] - 200s 4ms/step - loss: 0.6340 - acc: 0.7893 - val
Epoch 34/50
50000/50000 [==============================] - 192s 4ms/step - loss: 0.6413 - acc: 0.7898 - val
Epoch 35/50
50000/50000 [==============================] - 194s 4ms/step - loss: 0.6380 - acc: 0.7904 - val
Epoch 36/50
50000/50000 [==============================] - 199s 4ms/step - loss: 0.6344 - acc: 0.7899 - val
Epoch 37/50
50000/50000 [==============================] - 199s 4ms/step - loss: 0.6357 - acc: 0.7910 - val
Epoch 38/50
50000/50000 [==============================] - 196s 4ms/step - loss: 0.6409 - acc: 0.7898 - val
Epoch 39/50
50000/50000 [==============================] - 199s 4ms/step - loss: 0.6418 - acc: 0.7872 - val
Epoch 40/50
50000/50000 [==============================] - 197s 4ms/step - loss: 0.6382 - acc: 0.7907 - val
Epoch 41/50
50000/50000 [==============================] - 201s 4ms/step - loss: 0.6419 - acc: 0.7909 - val
Epoch 42/50
50000/50000 [==============================] - 201s 4ms/step - loss: 0.6470 - acc: 0.7870 - val
Epoch 43/50
50000/50000 [==============================] - 197s 4ms/step - loss: 0.6428 - acc: 0.7876 - val
Epoch 44/50
50000/50000 [==============================] - 193s 4ms/step - loss: 0.6452 - acc: 0.7875 - val
Epoch 45/50
50000/50000 [==============================] - 511s 10ms/step - loss: 0.6403 - acc: 0.7892 - va
Epoch 46/50
50000/50000 [==============================] - 412s 8ms/step - loss: 0.6455 - acc: 0.7875 - val
Epoch 47/50
50000/50000 [==============================] - 199s 4ms/step - loss: 0.6485 - acc: 0.7877 - val
Epoch 48/50
50000/50000 [==============================] - 200s 4ms/step - loss: 0.6387 - acc: 0.7892 - val
Epoch 49/50
50000/50000 [==============================] - 196s 4ms/step - loss: 0.6436 - acc: 0.7880 - val
Epoch 50/50
50000/50000 [==============================] - 193s 4ms/step - loss: 0.6480 - acc: 0.7875 - val
```

```
Out[16]: <keras.callbacks.History at 0x1a2dfb3f90>
```

1.6 Prueba del modelo

Luego del ajuste ahora se prueba el modelo con los datos de prueba.

```
In [17]: scores = model.evaluate(x_test, y_test, verbose=1)
         print('Test loss:', scores[0])
         print('Test accuracy:', scores[1])
```

```
10000/10000 [==============================] - 8s 766us/step
Test loss: 0.6841973246574402
Test accuracy: 0.7784
```

Siempre es importante salvar el modelo para trabajar posteriormente y evitar realizar procesos de ajuste de nuevo. De esta forma se ahorra tiempo.

```
In [18]: model.save('keras_cifar10_trained_model-100.h5')
```

Se puede cargar el modelo salvado y se pueden corroborar las características de la arquitectura.

```
In [13]: from keras.models import load_model
         # Cargar el modelo del archivo salvado.
         # Hay un modelo entrenado con 10 iteraciones keras_cifar10_trained_model.h5
         # otro con 50 iteraciones keras_cifar10_trained_model-50.h5
         # y uno con 100 iteraciones keras_cifar10_trained_model-100.h5
         model = load_model('keras_cifar10_trained_model-50.h5')
         model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
```

```
Non-trainable params: 0
_________________________________________________________________
```

Podemos corroborar de nuevo los valores

```
In [14]: # Evaluación del modelo
         scores = model.evaluate(x_test, y_test, verbose=1)
         print('Test loss:', scores[0])
         print('Test accuracy:', scores[1])

10000/10000 [==============================] - 7s 674us/step
Test loss: 0.66253902053833
Test accuracy: 0.78


In [15]: from keras.models import load_model
         # Cargar el modelo del archivo salvado.
         # Hay un modelo entrenado con 10 iteraciones keras_cifar10_trained_model.h5
         # otro con 50 iteraciones keras_cifar10_trained_model-50.h5
         # y uno con 100 iteraciones keras_cifar10_trained_model-100.h5
         model = load_model('keras_cifar10_trained_model-100.h5')
         model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
```

```
In [16]: # Evaluación del modelo
         scores = model.evaluate(x_test, y_test, verbose=1)
         print('Test loss:', scores[0])
         print('Test accuracy:', scores[1])
```

```
10000/10000 [==============================] - 7s 677us/step
Test loss: 0.6841973246574402
Test accuracy: 0.7784
```

Vamos a almacenar en `y_pred` los valores inferidos de los datos de prueba `x_test`.

```
In [17]: y_pred = model.predict(x_test)
```

1.7 Matriz de confusión

Para dibujar la matriz de confusión se utiliza la rutina confusion_matrix de scikit-learn.

A esta rutina se le pasan los datos de prueba y las predicciones generadas por el modelo.

```
In [18]: matrix = metrics.confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
         matrix
```

```
Out[18]: array([[818,   9,  31,  14,  16,   2,  15,   6,  41,  48],
                [ 14, 866,   2,   3,   2,   2,   7,   1,  12,  91],
                [ 63,   4, 646,  30,  87,  41, 107,  14,   2,   6],
                [ 33,   2,  62, 530,  75, 113, 139,  10,  16,  20],
                [ 12,   0,  32,  24, 813,   8,  86,  16,   5,   4],
                [ 12,   1,  47, 135,  62, 662,  46,  28,   2,   5],
                [  7,   1,  22,  18,  21,   5, 923,   0,   2,   1],
                [ 12,   1,  35,  37,  87,  53,  19, 734,   1,  21],
                [ 61,  20,   4,   7,   3,   2,  13,   0, 866,  24],
                [ 12,  31,   2,   4,   2,   1,   9,   2,  11, 926]])
```

La matriz se puede generar normalizada.

```
In [19]: matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
```

```
Out[19]: array([[0.818, 0.009, 0.031, 0.014, 0.016, 0.002, 0.015, 0.006, 0.041,
                 0.048],
                [0.014, 0.866, 0.002, 0.003, 0.002, 0.002, 0.007, 0.001, 0.012,
                 0.091],
```

```
       [0.063, 0.004, 0.646, 0.03 , 0.087, 0.041, 0.107, 0.014, 0.002,
        0.006],
       [0.033, 0.002, 0.062, 0.53 , 0.075, 0.113, 0.139, 0.01 , 0.016,
        0.02 ],
       [0.012, 0.   , 0.032, 0.024, 0.813, 0.008, 0.086, 0.016, 0.005,
        0.004],
       [0.012, 0.001, 0.047, 0.135, 0.062, 0.662, 0.046, 0.028, 0.002,
        0.005],
       [0.007, 0.001, 0.022, 0.018, 0.021, 0.005, 0.923, 0.   , 0.002,
        0.001],
       [0.012, 0.001, 0.035, 0.037, 0.087, 0.053, 0.019, 0.734, 0.001,
        0.021],
       [0.061, 0.02 , 0.004, 0.007, 0.003, 0.002, 0.013, 0.   , 0.866,
        0.024],
       [0.012, 0.031, 0.002, 0.004, 0.002, 0.001, 0.009, 0.002, 0.011,
        0.926]])
```

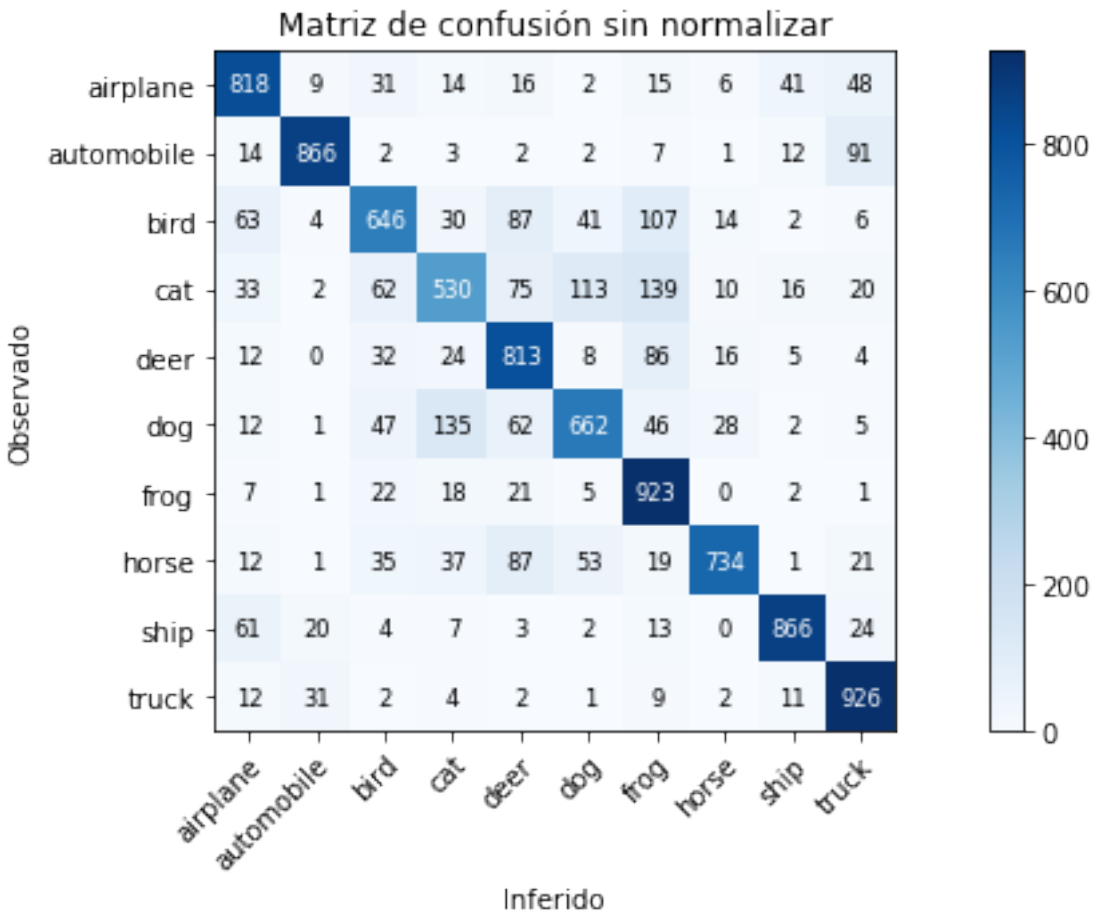
Hemos modificado el ejemplo para mostrar la matriz de confusión del ajuste con los datos CIFAR.

```
In [20]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import confusion_matrix
         from sklearn.utils.multiclass import unique_labels

         def plot_confusion_matrix(y_true, y_pred, classes,
                                   normalize=False,
                                   title=None,
                                   cmap=plt.cm.Blues,
                                   printtext=False):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if not title:
                 if normalize:
                     title = 'Matriz de confusión normalizada'
                 else:
                     title = 'Matriz de confusión sin normalizar'

             # Compute confusion matrix
             cm = confusion_matrix(y_true.argmax(axis=1), y_pred.argmax(axis=1))
             # Only use the labels that appear in the data
             classes = classes[unique_labels(y_true)]

             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

             if printtext:
```

```
        if normalize:
            print("Matriz de confusión normalizada")
        else:
            print('Matriz de confusión sin normalizar')
        print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    fig.set_size_inches(10, 5)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='Observado',
           xlabel='Inferido')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    fontsize='smaller',
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```

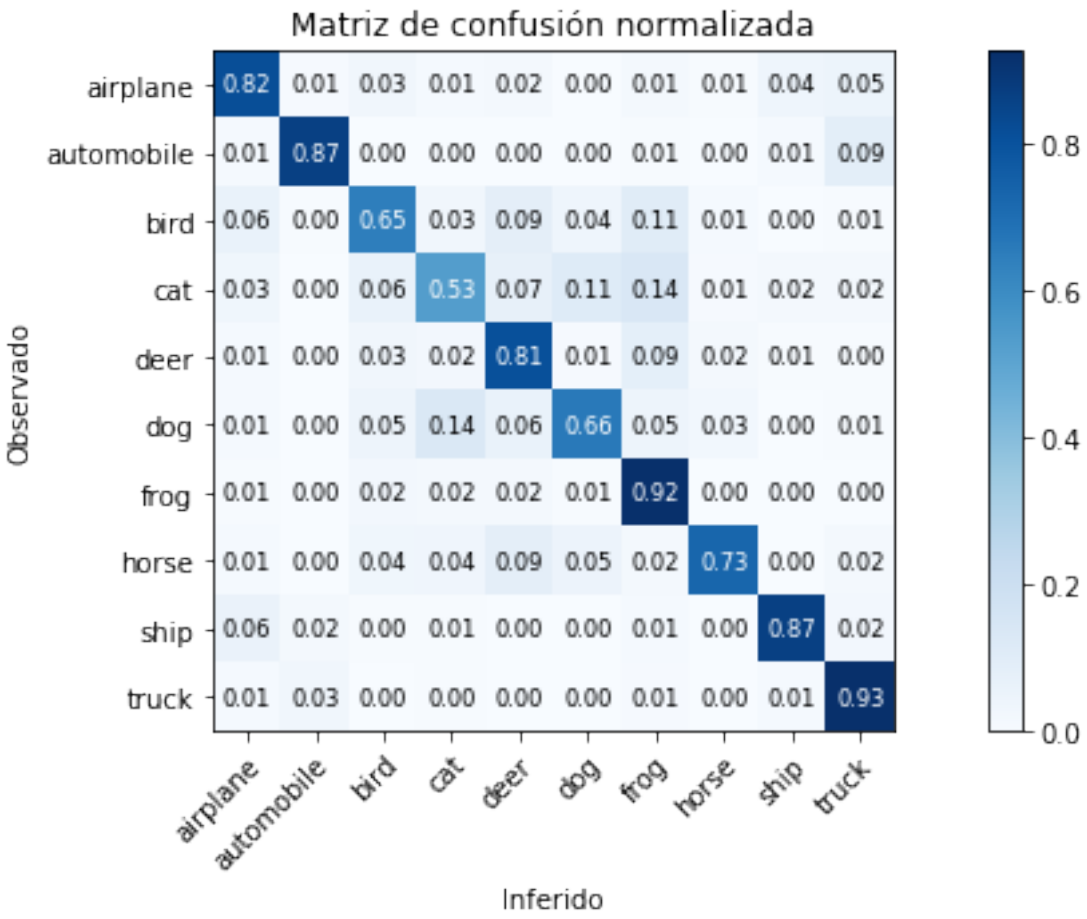
Ahora utilizamos la rutina recien creada para visualizar en una imagen la matriz de confusión.

```
In [21]: np.set_printoptions(precision=2)
         plot_confusion_matrix(y_test, y_pred, classes=class_names,
                               title=u"Matriz de confusi\u00f3n sin normalizar")
         plt.show()
```



Cambiando el parámetro `normalize` a `True` podemos ver la imagen con valores normalizados.

```
In [22]: np.set_printoptions(precision=0)
         plot_confusion_matrix(y_test, y_pred, classes=class_names, normalize=True,
                               title=u"Matriz de confusi\u00f3n normalizada")

         plt.show()
```


Matriz de confusión normalizada
airplane 0.82 0.01 0.03 0.01 0.02 0.00 0.01 0.01 0.04 0.05
automobile 0.01 0.87 0.00 0.00 0.00 0.00 0.01 0.00 0.01 0.09
bird 0.06 0.00 0.65 0.03 0.09 0.04 0.11 0.01 0.00 0.01
cat 0.03 0.00 0.06 0.53 0.07 0.11 0.14 0.01 0.02 0.02
deer 0.01 0.00 0.03 0.02 0.81 0.01 0.09 0.02 0.01 0.00
dog 0.01 0.00 0.05 0.14 0.06 0.66 0.05 0.03 0.00 0.01
frog 0.01 0.00 0.02 0.02 0.02 0.01 0.92 0.00 0.00 0.00
horse 0.01 0.00 0.04 0.04 0.09 0.05 0.02 0.73 0.00 0.02
ship 0.06 0.02 0.00 0.01 0.00 0.00 0.01 0.00 0.87 0.02
truck 0.01 0.03 0.00 0.00 0.00 0.00 0.01 0.00 0.01 0.93
Observado
airplane automobile bird cat deer dog frog horse ship truck
Inferido
0.8 0.6 0.4 0.2 0.0

Bibliografía

[1] Artificial Intelligence A Modern Approach, 3rd Edition, Stuart J. Russell y Peter Norvig, Pearson, 2016.

[2] A Logical Calculus of Ideas Immanent in Nervous Activity, Warren McCulloch y Walter Pitts.

[3] The Perceptron: A probabilistic model for information storage and organization in the brain. Frank Rosenblatt, Cornell Aeronautical Laboratory, Psychological Review, Vol. 65, No. 6, 1958.

[4] Scaling Learning Algorithms towards AI, Large-Scale Kernel Machines, Yoshua Bengio y Yann LeCun, MIT Press, 2007

[5] Receptive fields of single neurones in the Cat's striate cortex, D. H. Hubel y T. N. Wiesel Wilmer Institute, The Johns Hopkins Hospital and University, Baltimore, Maryland, U.S.A.

[6] Gradient-Based Learning Applied to Document Recognition, Yann LeCun, Léon Bottou, Yoshua Bengio y Patrick Haffner.

[7] Efficient Estimation of Word Representations in Vector Space, Tomas Mikolov, Kai Chen, Greg Corrado y Jeffrey Dean.

[8] DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks, David Salinas, Valentin Flunkert, Jan Gasthaus.

[9] Factorization Machines, Steffen Rendle, Department of Reasoning for Intelligence The Institute of Scientific and Industrial Research Osaka, University, Japan.

[10] Deep Residual Learning for Image Recognition, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.

[11] Neural Variational Inference for Text Processing, Yishu Miao, Lei Yu.

[12] Fully Convolutional Networks for Semantic Segmentation, Evan Shelhamer, Jonathan Long y Trevor Darrell.

[13] Pyramid Scene Parsing Network, Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang y Jiaya Jia.

[14] Rethinking Atrous Convolution for Semantic Image Segmentation Liang-Chieh Chen, George Papandreou, Florian Schroff y Hartwig Adam.

[15] The Elements of Statistical Learning, Data Mining, Inference and Prediction, 2nd Edition, Trevor Hastie, Robert Tibshirani y Jerome Friedman, Springer, 2008.

[16] Object Recognition with Gradient-Based Learning, Yann Lecun, Patrick Haffner, Léon Bottou and Yoshua Bengio, ATT Shannon Lab, 100 Schulz Drive, Red Bank, NJ, 07701, USA.

[17] Deep Learning for Computer Vision, Rajalingappa Shanmugamani, Packt Publishing Ltd, 2018.

[18] ImageNet: Classification with Deep Convolutional Neural Networks, Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton, University of Toronto.

[19] CS231n Convolutional Neural Networks for Visual Recognition, Andrej Karpathy, http://cs231n.github.io/convolutional-networks/.

[20] MNIST. http://yann.lecun.com/exdb/mnist/, Recolección realizada por Yann LeCun, Corinna Cortes y Christopher J.C. Burges.

[21] CIFAR-10 y CIFAR-100. https://www.cs.toronto.edu/~kriz/cifar.html, Recolección realizada por Alex Krizhevsky, Vinod Nair y Geoffrey Hinton.

[22] MXNet: A Flexible and Efficient Machine Learning, Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, Zheng Zhang, Library for Heterogeneous Distributed Systems.

[23] Amazon, *AWS DeepLens Developer Guide*. Seattle, USA, 2018.

[24] One-Shot Learning of Object Categories, Li Fei-Fei, Member IEEE, Rob Fergus, Student Member IEEE, and Pietro Perona, Member IEEE, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 28, NO. 4, APRIL 2006.

[25] Generative Adversarial Networks, Goodfellow Ian, Pouget-Abadie Jean, Mirza Mehdi, Xu Bing, Warde-Farley David, Ozair Sherjil, Courville Aaron, Bengio Yoshua (2014), Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.

[26] *Deep Learning*, Ian Goodfellow, Yoshua Bengio y Aaron Courville, MIT Press, 2016.

[27] Introduction to Convolutional Neural Networks, Jianxin Wu, LAMDA Group, National Key Lab for Novel Software Technology, Nanjing University, China.

[28] Derivation of Backpropagation in Convolutional Neural Network (CNN), Zhifei Zhang, University of Tennessee, Knoxvill, TN.

[29] Deep learning for pedestrians: backpropagation in CNNs, Laurent Boué, SAP Labs.

[30] Visualizing and Understanding Convolutional Networks, Matthew D. Zeiler y Rob Fergus, Dept. of Computer Science, New York University, USA.

[31] Evolutionary Design of Deep Neural Networks Author: Alejandro Baldominos Gómez, Supervisors: Dr. Pedro Isasi Viñuela, Dr. Yago Sáez Achaerandio, Computer Science Department, Ph.D. Programme in Computer Science and Technology, Universidad Carlos III de Madrid, Leganés.

[32] Introducción a las Técnicas de Computación Inteligente, Editores: José Aguilar Castro y Francklin Rivas Echeverría, Universidad de Los Andes, Medida, Venezuela, Junio, 2001.

[33] Amazon, *Amazon SageMaker*. Seattle, USA, 2018.

[34] Amazon, *Amazon SageMaker Documentation*. Seattle, USA, 2018.

[35] Amazon, *AWS DeepLens*. Seattle, USA, 2018.

[36] Amazon, *AWS DeepLens Developer Resources*. Seattle, USA, 2018.