



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y
MATEMATICAS

Clasificación de imágenes de arrecifes de coral: Análisis teórico y práctico del aprendizaje profundo y enfoque Multiview

Autor

Iván Sevillano García

Directores

Francisco Herrera Trigueros
Siham Tabik



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Junio de 2019

**Clasificación de imágenes de arrecifes de coral:
Análisis teórico y práctico del aprendizaje
profundo y enfoque Multiview**

Autor

Iván Sevillano García

Directores

Francisco Herrera Triguero
Siham Tabik

Clasificación de imágenes de arrecifes de coral: Análisis teórico y práctico del aprendizaje profundo y enfoque Multiview

Iván Sevillano García

Palabras clave: Aprendizaje automático, aprendizaje automático multi vista, redes neuronales, back-propagation

Resumen

En nuestro tiempo, la cantidad de información está creciendo más y más. Extraer conocimiento de toda esa información es uno de los retos de nuestro tiempo. El **Machine Learning** es el estudio de algoritmos y modelos estadísticos que puedan extraer este conocimiento de manera autónoma.

En el presente trabajo, introduciremos los principales conceptos del *Machine Learning* y discutiremos la factibilidad del aprendizaje basado en la información que se posee. Estudiaremos cómo un algoritmo puede aprender una tarea sin haber sido programada explícitamente para ello. Este aprendizaje lo podrá conseguir gracias a la experiencia que le supplementemos.

Como segunda tarea, introduciremos el concepto de **Multiview Machine Learning**. En la actualidad, la información que se obtiene puede venir de distintas fuentes, las cuales no tienen por qué tener siempre la misma estructura. Desde un punto de vista clásico del aprendizaje automático, no es fácil construir un modelo que aprenda de esta información no estructurada. El *Multiview Machine Learning* propone modelos que pueden trabajar con esta información no estructurada. Aprenderemos los principios básicos de este nuevo concepto y propondremos algoritmos de aprendizaje que tengan en cuenta las características propias del enfoque *Multiview*.

Introduciremos también el concepto de **Deep Learning** dentro del *Machine Learning*, y su herramienta más importante, las **redes neuronales artificiales**. Los modelos que el *Deep Learning* propone son bien conocidos por su buen comportamiento en distintos problemas del *Machine Learning*. Estudiaremos la estructura propia de las redes neuronales y cómo son capaces de aprender de experiencia. Nos centraremos en las redes neuronales **pre alimentadas**, el tipo de red neuronal que se utiliza generalmente en el aprendizaje supervisado.

El problema del aprendizaje se puede reducir a un problema de optimización por lo que, para llegar a conseguir aprender, necesitamos estudiar y

aplicar algoritmos de optimización. Introducimos las bases del cálculo diferencial y el algoritmo de optimización por excelencia, el algoritmo del **Gradiente Descendente**.

Uno de los problemas del *Machine Learning* dentro de la visión por computador es la **Clasificación de Imágenes**. En este problema, se pretende extraer información de imágenes. Para este objetivo, se ha desarrollado una nueva estructura de redes neuronales: las redes neuronales **Convolucionales**. Aprenderemos cómo funcionan y las diferencias entre las redes neuronales convolucionales y las redes neuronales completamente conectadas.

Aunque las herramientas utilizadas para extraer conocimiento actuales han demostrado tener muy buenos resultados, una vez construido un modelo con este conocimiento, la información puede quedar obsoleta en poco tiempo. En este trabajo estudiamos el concepto de *Transfer Learning*. Los seres humanos son capaces de abstraer conocimiento y reutilizar ideas de un problema en otros problemas nuevos sin la necesidad de estudiarlos desde cero. Veremos cómo se puede utilizar esta misma técnica en el *Machine Learning*. Veremos también que, al igual que en el caso de los seres humanos, esto ahorra tiempo y puede generar buenos modelos igualmente.

Tras todo el estudio teórico realizado, se plantea el problema de identificación de especies de corales en imágenes submarinas. Los arrecifes de coral son estructuras muy importantes en los ecosistemas marinos ya que son el sustento de la mayoría de especies que viven en la zona. En la actualidad, estudios han demostrado que estos arrecifes están disminuyendo su volumen y expertos han insistido en la necesidad de un método eficiente que cuantifique esta pérdida e identifique qué especies de corales están más dañadas. En este ambiente, surge la necesidad de crear un modelo que identifique de forma automática la especie de un coral en base a una imagen subacuática.

Para el problema que se plantea existen multitud de bases de datos para el entrenamiento de los posibles modelos. Sin embargo la mayoría de ellas poseen imágenes que han sido muy preprocesadas o en las que expertos han eliminado partes de la imagen que corresponden al fondo marino y demás conocimiento experto que no es posible transmitir a los modelos. Por ello, para la automatización real del problema de identificación de corales se precisa de una base de datos que no contenga datos preprocesados ni preseleccionados por expertos. La base de datos *structureRSMAS*, desarrollada en [1], posee estas características. Sin embargo, por la eliminación del preprocesamiento y selección experta de características, las imágenes son mucho más difícil de tratar. Entre otras dificultades, las imágenes no tratadas tienen una gran parte de fondo marino en ellas y no poseen demasiada calidad por

ser imágenes tomadas bajo el agua. Además, al ser imágenes muy generales de la estructura del coral, puede que haya varias especies de corales en la misma imagen, provocando solapamiento de clases.

En este trabajo partiremos del trabajo realizado en [1], donde se plantea una red neuronal convolucional que hace uso de *Transfer Learning*. Utiliza una red pre entrenada para otro problema de identificación en imágenes cuya base de datos es imagenet [2]. En general, este modelo basado en *Transfer Learning* consigue unos resultados excelentes pero, dada la complejidad de la base de datos, es necesario el desarrollo de mejores modelos.

El modelo de aprendizaje que se propone en este trabajo incorpora a la red planteada un enfoque *Multiview*, el modelo *MVL*. También se plantean modelos estructuralmente parecidos para comprobar la utilidad de este enfoque. Se analizan distintas estructuras, etapas de aprendizaje de los modelos *MVL*. Además, los modelos más reseñables para la comparativa son los *Ensemble Stacking*. Estos modelos son los más interesantes a la hora de comparar con el modelo *MVL* propuesto en este trabajo por la semejanza en su planteamiento. Los modelos *Stacking* se basan en juntar el conocimiento adquirido por varios modelos. Se termina comprobando que el modelo principal *MVL* es el que mejor se comporta para este problema.

Image coral reef Classification: Theoretical and Practical Analisys of Machine Learning and Multiview Machine Learning

Iván Sevillano García

Keywords: Machine Learning, Multiview Machine Learning, Artificial Neural Networks, back-propagation

Abstract

In our times, the amount of information is growing more and more. Trying to extract knowledge from this information is one of the challenges of our era. **Machine Learning** is the study of algorithms and statistical models that can extract this information on his own.

In this project, we will introduce the formal concepts of Machine Learning and will discuss the feasibility of this information extraction. We will study how can a machine learn a task without being programmed to it. This will be possible thanks to the experience we suply to this machine.

As a second task in this project, we will introduce **Multiview Machine Learning**. Nowadays, the information we can obtain from multiple sources have not always the same structure. With a classical Machine Learning point of view is not so easy to build a model to learn from this kind of not structured data. However, the Multiview point of view presents a new approach that can work well with this information. We will learn the bases for this new concept and will propose algorithms that will work with this premise.

We will also introduce the concept of **Deep Learning** on Machine Learning, and the most important structure for it, the **artificial Neural Networks**. The models Deep Learning introduce are well-known for the good performance on different problems of Machine Learning. We will study the structure and the way neural networks learn information from experience. We will focus on the **feedforward** neural networks, the kind of neural networks we will use to the supervised learning task later.

The learning problem will be reduced to an optimization problem, so to aim the learning problem task we need to learn about optimization algorithms. We introduce the bases of differential analysis and the **Gradient Descent** algorithm to use it later.

One of the most important Machine Learning problems in **Computer Vision** is the **image Clasification**. In this problem, we try to obtain knowledge from images. A new neural network has been developed for this problem: **Convolutional Neural Networks**. We will learn how they work and which are the difference between convolutional and fully connected neural networks.

Another interesting topic in our times is the **Transfer Learning**. This concept refers to the possibility of taking advantage of the knowledge we have before we try to learn nothing from a new problem. Humans has the ability of abstract his knowledge of past problems to find a solution to new ones. This gives us a huge capacity to solve new problems without the need of learning from zero each problem. We will see how we can try the same strategy on Machine Learning. The benefits of this technique are the same as the benefits on human: less time needed to learn approaching maybe even better results.

After all this theoretical study, we propose a practical problem of image classification of coral species in subacuatic images. Coral reefs are important structures on see life. That is because they are the sustent of most of the species that lives in the zone. Nowadays, there are studies that have shown that coral reefs are decreesing in volume and experts insist on the need of having a method that quantifies this loss and identifies the coral species that are threatened or in danger. With this premises, emerge the need of building a model which task is the automatic coral species recognition based of submarines images. Morover, as the images of this data base shows the structure of the corals from a longer distance, there could be more than one species of coral on one image, provoking a class overlapping.

For this problem, there are lots of data bases for the training of models. However most of them have very preprocessed images or images where experts has identify which part of image is the coral, deleting the background of the image. This knwoledge can't be transfered to a Machine Learning model. Thus, for the real task of automatic image classification of caral species it is needed a data base that has not this expert knowledge based on experts selection or expert preprocessing of images. *StructureRSMAS*, developed in [1], is a data base with this characteristics. But, as the images has not been preprocessed and the background has not been removed, the develop of a good Machine Learning model is hard.

In this project we start from the work done in [1], where it is proposed a *Deep Learning* model based on convolutional artificial neural networks which uses Transfer Learning. It use a network trained for another images class classification problem with the imangenet database[2]. Generaly, this

model bassed on Transfer Learning is supossed to achive good results but, because of the complexity of tha data base, it is needed the develop of better and more complex models.

The Machine Learning model we propose in this job adds to the base model proposed before a Multiview approach, the *MV* model. We also propose three more models that are similar to the *MVL* model propose to probe his goodness. We propose different models based on the structure and the learning algorithms the *MVL* models has. Finally, we develop *Ensemble Stacking* models. Because of the similarity of their structure, the comparsion between *MVL* and *Stacking* models is the most interesting one. *Stacking* models are based of joining the knowledge of two or more models. At the end, we check that the first approach of *MVL* models has the best behavior on tha problem proposed.

Yo, **Iván Sevillano García**, alumno de la titulación DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77187364-P, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Iván Sevillano García

Granada a 15 de Junio de 2019 .

D. **Francisco Herrera Trigueros** , Profesor del Área de Ciencias de la Computación e Inteligencia Artificial el del Departamento de Ciencias de la Computación e Inteligencia Artificial el de la Universidad de Granada.

D. **Siham Tabik** , Profesor del Área de Ciencias de la Computación e Inteligencia Artificial el del Departamento de Ciencias de la Computación e Inteligencia Artificial el de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Clasificación de imágenes de arrecifes de coral, Análisis teórico y práctico del aprendizaje profundo y enfoque Multiview*, ha sido realizado bajo su supervisión por **Iván Sevillano García** , y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 17 de Junio de 2019 .

Los directores:

Francisco Herrera Trigueros Siham Tabik

Agradecimientos

Para empezar quiero pedir perdón a las personas importantes en mi vida que no mencione en estos agradecimiento. Los que me conocéis sabéis que esta cabeza loca es capaz de olvidar hasta de lo que está hablando.

Este proyecto ha sido desarrollado dentro de una beca de colaboración con el departamento de Ciencias de la Computación. Agradezco la confianza que se ha depositado en mi para esta tarea.

Quiero dar las gracias a Paco por confiar en mi y velar por mi futuro como un padre, tanto en sus mejores como en sus peores momentos.

A Siham, por las distintas consideraciones necesarias para el trabajo. Me hace falta todavía mucho para alcanzar el nivel de seriedad científica que me ha transmitido en este trabajo.

A Anabel, por ayudarme a desarrollar la parte práctica de este trabajo. Haber partido de un proyecto tan interesante me ha hecho aprender bastante y me ha permitido vislumbrar la vanguardia del conocimiento en este tema.

A mi primer profesor de matemáticas, Luis Bravo, que inculcó en una mente de primero de ESO una inquietud que me ha ido guiando desde entonces.

A Felix y Juan Luis, mis rivales y compañeros en esos primeros cursos del instituto. Sin la competitividad sana que desarrollé para intentar alcanzarlos no habría estado a la altura para años venideros.

A todo ESTALMAT. ¿Quién sabe si las matemáticas me habrían parecido tan divertidas como lo parecieron en estas clases? Esta comunidad es un entorno maravilloso para el desarrollo de estas inquietudes.

A las personas que conocí en ERASMUS y todo lo que cada una me enseñó. Creo que aunque nos separemos, hemos experimentado una vida entera en sólo un año de la que hemos aprendido más de la vida que en todos los años de carrera.

A Sergio y la alegría con que te acuerdas de mi. Cada vez que recibo noticias tuyas, se me viene una sonrisa a la cara.

A Guille. Casi parecieras el hermano mayor de todos en el ERASMUS, y como tal, mereces un hueco en este agradecimiento.

A los nuevos compañeros de clase que he conocido apenas este año.

Habéis hecho mi último periodo de la carrera un lugar al que ir a pasarlo bien.

A mi amigo Alberto. Una amistad así no cambia aunque estemos largos periodos sin contacto.

A mi buddy, Eleonora, por ese afán de querer descubrir la ciudad de Granada hasta llegar a conocerla mejor que yo.

A mi familia del Turismo Rural. Tener tantos hermanos pequeños de los que cuidar una vez al año durante toda la vida no pudo ser más entretenido.

A Carmen, mi scout de confianza. Sólo a ti se te podía ocurrir que, tras terminar la carrera de derecho, tu verdadero sueño era estudiar medicina. Y por supuesto que lo ibas a conseguir. Eres toda una heroína y un gran ejemplo de superación.

A mis perros. Puede que vosotros no entendáis estos agradecimientos pero espero transmitiros lo que significan cuando vuelva a veros en casa.

A mi Alemán preferido, Thorsten. Sé que siempre tendrá una casa donde tu estés al igual que tu tendrás una donde quiera que yo me encuentre.

A Hector, una nueva adquisición. Disfruta de Granada estos años y recuerda que esos ratitos, pues son para ti.

A Paco y Juan y nuestras conversaciones de media noche en las que solucionamos el mundo.

A Andrés. Siempre me fascino imaginándome tu futuro y sintiendo como propias las posibilidades que se te abren. Sigue así y recuerda no olvidarte de disfrutar.

A Dani.

A Nuria.

A Moya.

A Elena.

A David.

A Marta.

A María.

A vosotros os debo mi vida en Granada puesto que habéis estado presentes en los momentos más importantes. En mis recuerdos siempre quedarán las historias locas que nos han ocurrido: el paraguas del congelador, el regalo sorpresa de topología, audios de estopa que nos recuerdan quién se lo está pasando bien, el encanto de las cuevas de Guadix, viajes por toda europa, juegos de mesa que casi nos llevan a las manos o el aprender a ser malvado. Todos estos momentos son solo una pequeña parte de lo que os quiero agradecer. Sois los que me dais más ganas de volver a Granada, de esforzarme más y sobre todo, los que me habéis dado la ilusión para escribir estos agradecimientos.

A Jorge, el mejor amigo que uno puede tener. Si en todos estos años ha habido una constante, esa has sido tu. No me imagino un futuro sin el apoyo de alguien tan especial. Aunque sólo seas un doble ingeniero.

A mis abuelos Teresa, Falete y Maria Luisa por vuestro amor incondicional. Espero que estéis orgullosos de vuestro nieto.

A mi padre, por todos los sacrificios que has hecho y haces. Nunca te podré agradecer lo suficiente todos ellos.

A mi madre. Eres un gran ejemplo de superación en la vida. De mayor quiero ser como tu.

y por último, a mi hermano Pablo, la persona que más aprecio. Puede que no te lo diga tan a menudo como debería, pero te quiero. Deseo que encuentres el camino que te haga feliz.

Índice general

1. Introducción	1
2. Objetivos del trabajo	5
I Aproximación Matemática	7
3. Fundamentos del Machine Learning	9
3.1. Introducción al Machine Learning	9
3.2. Factibilidad del aprendizaje	10
4. Multiview Machine Learning	13
4.1. Introducción al Multiview Machine Learning.	13
4.2. Métodos de máxima separación(margin-consistency)	15
4.2.1. Maximun Entropy discriminant(MED)	15
4.2.2. Generalización a varias vistas.(MVMED)	16
4.3. Creación de vistas artificiales	17
5. Computación numérica y problemas de optimización.	19
5.1. Overflow y Underflow.	19
5.2. Condicionamiento	20
5.3. Optimización basada en el gradiente.	21
5.4. Uso de las segundas derivadas: Matriz Hessiana.	23
5.5. Optimización con restricciones.	25
6. Deep Feedfordward Networks	27

6.1.	Función de coste	29
6.2.	Unidades de salida.	30
6.2.1.	Unidades lineales	31
6.2.2.	Unidades sigmoidales	31
6.2.3.	Unidad softmax para distribuciones multinomiales. . .	32
6.2.4.	Otras unidades de salida	33
6.3.	Unidades ocultas	34
6.3.1.	Unidades lineales rectificadas y generalizaciones(ReLU)	34
6.3.2.	Unidades Sigmoide logistico y tangente hiperbólica. .	35
6.3.3.	Otras unidades ocultas.	36
6.4.	Diseño de la arquitectura de la red	36
6.4.1.	Teorema de aproximación universal.	37
6.4.2.	Otras consideraciones de la arquitectura.	38
6.5.	Redes Neuronales Convolucionales	38
6.6.	Introducción al Transfer Learning	41
6.7.	Transfer Learning en Imágenes: Problema imagenet	42
7.	Algoritmo de Back-propagation	43
7.1.	Grafos computacionales	44
7.2.	Regla de la cadena para cálculo de derivadas.	44
7.2.1.	Algoritmo <i>backpropagation</i>	46
7.2.2.	Back-propagation en redes neuronales completamente conectadas	48
7.2.3.	Derivadas símbolo a símbolo	48
7.2.4.	Algoritmo Back-propagation genérico.	49
II	Aproximación Informática	53
8.	Caso de estudio: Clasificación de corales. Base de datos <i>structureRSMAS</i>	55
8.1.	Clasificación de corales	55
8.2.	Data Augmentation	56
8.3.	Medida del error. Validación cruzada(<i>K-fcv</i>).	58

8.4. Clasificación de corales: Estado de arte	59
9. Propuestas de modelos	63
9.1. Creación de las vistas artificiales. Núcleos artificiales de aprendizaje basados en Transfer Learning.	63
9.2. Propuesta Multiview para 2 vistas(MVL)	64
9.3. Modelos Densos(MD)	66
9.4. Modelo One Stage Multiview(1-STA-MVL)	68
9.5. Modelos de Ensemble:Stacking(ST)	70
9.5.1. Stacking Res-Inc	70
9.5.2. Stacking Res-Inc-MVL	72
10. Análisis de resultados.	75
10.1. Software utilizado	77
III Conclusion	79
11. Conclusiones y vías futuras	81
Bibliografía	84

Capítulo 1

Introducción

El *Machine Learning* es una de las fronteras del conocimiento de la actualidad. En el presente trabajo se estudia formalmente los principios básicos de este campo. Utilizaremos herramientas estadísticas para asegurar la posibilidad de aprendizaje y así motivar su estudio y posterior aplicación.

Con la posibilidad de aprender asegurada, generar modelos que sean capaces de extraer conocimiento de la vida real eficientes y eficaces es cada vez más importante. El *Machine Learning* pretende estudiar distintos problemas de la vida real y generar modelos que sean capaces de extraer conocimiento de los mismos.

Uno de los grandes problemas que se presentan a la hora de obtener conocimiento es la falta de estructuración de la información, es decir, que no todos los datos tienen la misma forma. El *Multiview Machine Learning*, que también introducimos en este trabajo, pretende solucionar en parte este problema. En él, se plantea la posibilidad de acoplar conocimiento de distintas fuentes de información que no tengan necesariamente la misma estructura.

Para la extracción del conocimiento, uno de los campos más importantes es el *Deep Learning*, cuyas estructuras más características son las redes neuronales artificiales. Estas han demostrado tener muy buenos resultados en problemas de aprendizaje. En este trabajo se estudia la estructura y funcionamiento interno de las mismas. Además, se presenta tanto el algoritmo de gradiente descendente, el cuál es primordial para el cálculo de extremos relativos de problemas de minimización, como el algoritmo *backpropagation*, utilizado para el cálculo del gradiente. Este último algoritmo es sumamente importante en la aplicación de las redes neuronales artificiales ya que consigue que el cálculo del gradiente sea mucho más veloz.

Entre los problemas de aprendizaje que pueden abarcar las redes neuronales artificiales, se encuentra el que venimos a estudiar en este trabajo:

la clasificación de imágenes. Un nuevo modelo de red neuronal se plantea para ello y demuestra su buen comportamiento, alcanzando y superando algunos estados del arte de muchos problemas de este tipo: las redes neuronales convolucionales. A nivel intuitivo, estas redes neuronales interpretan la información de una imagen como un humano lo haría de forma natural. Estudiaremos la estructura y comportamiento de estas redes para su posterior uso en la parte práctica del trabajo.

Otro gran hito de nuestra época es el *Transfer Learning*. Los seres humanos somos capaces de adaptar el conocimiento que tenemos de problemas ya resueltos a nuevos problemas que nos encontramos. Esto nos hace tener una capacidad para resolver problemas mucho más amplia que si tuviésemos que aprender cada problema desde cero. La transferencia de conocimiento a nivel del *Machine Learning* también es posible y, al igual que como el ser humano, se ahorra mucho tiempo de aprendizaje utilizando esta técnica, por lo que quedan claros los beneficios de la aplicación del *Transfer Learning*.

Tras todo el estudio teórico realizado, se plantea el problema de identificación de especies de corales en imágenes submarinas. Los arrecifes de coral son estructuras muy importantes en los ecosistemas marinos ya que sustentan a la mayoría de especies que viven en la zona. En la actualidad, estudios han demostrado que estos arrecifes están disminuyendo su volumen y expertos han insistido en la necesidad de un método eficiente que cuantifique esta pérdida e identifique qué especies de corales están más dañadas. En este ambiente, surge la necesidad de crear un modelo que identifique de forma precisa la especie de un coral en base a una imagen subacuática.

Para el problema que se plantea existen multitud de bases de datos para el entrenamiento de los posibles modelos. Sin embargo la mayoría de ellas poseen imágenes que han sido muy preprocesadas, expertos han eliminado partes de las imágenes que corresponden al fondo y demás conocimiento experto que no es posible transmitir a los modelos. Por ello, para la automatización real del problema de identificación de corales se precisa de una base de datos que no contenga datos preprocesados ni preseleccionados por expertos. La base de datos *structureRSMAS*, desarrollada en [1], posee estas características. Sin embargo, por la eliminación del preprocesamiento y selección experta de características, las imágenes son mucho más difícil de tratar. Entre otras dificultades, las imágenes no tratadas tienen una gran parte de fondo marino en ellas y no poseen demasiada calidad por ser imágenes tomadas bajo el agua.

Para el desarrollo de un modelo competente para este problema, partiremos del trabajo realizado en [1], donde se plantea una red neuronal convolucional que hace uso de *Transfer Learning*. Utiliza una red pre entrenada para otro problema de identificación en imágenes cuya base de datos es *Imagenet* [2]. En general, este modelo basado en *Transfer Learning* unos

resultados excelentes pero, dada la complejidad de la base de datos, es necesario el desarrollo de mejores modelos.

El modelo de aprendizaje que se propone en este trabajo incorpora a la red planteada un enfoque *Multiview*, el modelo *MVL*. También se plantean modelos estructuralmente parecidos para comprobar la utilidad de este enfoque. Proponemos para ello distintos modelos que difieren con el principal modelo *MVL* en su estructura y etapa de aprendizaje. Además se estudian modelos basados en *Ensemble Stacking*. Estos modelos son los más importantes a la hora de comparar con el propuesto en este trabajo por la semejanza en su planteamiento. Este tipo de modelos se basa en reunir el conocimiento adquirido por varios modelos. Se termina comprobando que el modelo principal *MVL* es el que mejor se comporta para este problema.

Capítulo 2

Objetivos del trabajo

Al comienzo de este trabajo, los objetivos que se tenían eran los siguientes:

- A nivel teórico:

- Aprender los fundamentos formales del *Machine Learning* y la factibilidad del mismo.
- Estudiar el enfoque *Multiview* del *Machine Learning* para su posible aplicación a problemas de *Machine Learning*.
- Analizar los modelos de redes neuronales prealimentadas y, en concreto, el modelo más eficaz para reconocimiento de imágenes, las redes neuronales convolucionales, tanto su estructura como su funcionamiento.
- Exponer la teoría desarrollada del cálculo diferencial para optimización, necesaria para el estudio de las redes neuronales. Se pretende entender el algoritmo de gradiente descendente para cálculo de mínimos. Con objetivo de hacer más eficiente este algoritmo, presentamos el algoritmo *backpropagation* para el rápido cálculo del gradiente.
- Entender el concepto de *Transfer Learning*.

- A nivel práctico:

- Comprender y replicar los modelos estudiados en [1] basados en la teoría que se pretende estudiar a nivel teórico.
- Aplicar *Data Augmentation*, una técnica que aumenta el volumen de datos en nuestro caso.
- Proponer un modelo *Multiview* para el problema de clasificación de imágenes. Se pretende construir vistas artificiales y aplicar

algoritmos propios del *Multiview Machine Learning* considerando estas vistas.

- Construir otros modelos con los que poder comparar el modelo *Multiview*. No solo queremos construir los modelos replicados sino además otros modelos que tengan estructuras parecidas.
- Finalmente, queremos construir un tipo de modelo ampliamente probado y que ha demostrado su eficacia: los ensambles *Stacking*. Estos modelos acoplan otros modelos y aprovechan la información de todos ellos. Por la característica de acoplamiento, se pretende construir además un modelo *Stacking* que acople el modelo *Multiview* propuesto y los modelos replicados.

Parte I

Aproximación Matemática

Capítulo 3

Fundamentos del Machine Learning

El **Machine Learning**, o **aprendizaje automático**, es una rama de la computación que estudia la forma de obtener conocimiento a través del aprendizaje. En la primera subsección se define el concepto de aprendizaje y se planteará el problema de la obtención de conocimiento. En la segunda, se plantea si la obtención de este conocimiento es factible.

3.1. Introducción al Machine Learning

En 1997, Tom Mitchell[3] propone la siguiente definición de aprendizaje: Se dice que un algoritmo aprende en base a la experiencia E con respecto a una tarea T y una medida P si el rendimiento del algoritmo para cumplir la tarea T mejora con respecto a P gracias a la experiencia E.

Una tarea T será entendida como cualquier objetivo que pretendemos que nuestro algoritmo aprenda. Algunos ejemplos de tareas son clasificación, regresión, detección de anomalías, eliminación de ruido de instancias corruptas o la predicción de función de densidad de una variable aleatoria.

Una medida P debe medir, de alguna forma, la bondad de nuestro algoritmo. En el ejemplo de clasificación, una medida usual es la **precisión**. Esta medida no es más que el número de instancias que han sido clasificadas correctamente dividido por el número total de instancias clasificadas. En otros ejemplos de tarea esta medida no tiene sentido. Si nuestro algoritmo tiene como tarea la regresión de una función específica no tiene sentido el hablar de aciertos o errores en la clasificación. Un ejemplo de medida para este problema puede ser el **Error cuadrático medio(ECM)**.

Cuando hablamos de experiencia E solemos hablar de una base de datos

en los que cada instancia es un ejemplo. Los algoritmos de Machine Learning se pueden clasificar en dos categorías dependiendo del tipo de ejemplos que le suministremos:

- **Aprendizaje no supervisado.** En este tipo de aprendizaje se presta a estudiar la composición de la base de datos así como las características más importantes de la misma.

La tarea que nuestro algoritmo será, por tanto, aprender la distribución de probabilidad $p(x)$ del vector aleatorio x o de algún estadístico basado en el mismo. Esto es:

- **Aprendizaje supervisado.** Cada ejemplo tiene asociado una etiqueta u objetivo.

La tarea que nuestro algoritmo será en este caso estimar la probabilidad condicionada $p(y|x)$ de la etiqueta y conociendo el valor del vector aleatorio x .

Podemos generalizar estos conceptos de aprendizajes al de la búsqueda de una función objetivo $f: \mathcal{X} \rightarrow \mathcal{Y}$ donde \mathcal{X} es el dominio de los posibles inputs e \mathcal{Y} es el codominio o posibles outputs de la función objetivo. Nuestra experiencia E se puede formalizar como un conjunto $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$, $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$ en el que $f(x_i) = y_i$ para $i = 1, \dots, N$. Finalmente, el algoritmo de aprendizaje utilizará el conjunto \mathcal{D} para encontrar una función $g: \mathcal{X} \rightarrow \mathcal{Y}$ que aproxime a f . La función g será buscada por el algoritmo de aprendizaje dentro de un conjunto de funciones al que llamaremos hipótesis y denotaremos \mathcal{H} .

Para medir la proximidad de la función g a la función objetivo f contaremos con otro subconjunto de $\mathcal{X} \times \mathcal{Y}$ al que llamaremos **conjunto de test**. Atendiendo a los resultados de la función g en el conjunto de test, definiremos una **función de coste**, expresada usualmente en términos del error cometido.

El problema de aprendizaje puede enfocarse como un problema de minimización de la función de coste asociada a nuestro conjunto de test.

3.2. Factibilidad del aprendizaje

En el apartado anterior hemos dado por supuesto que el conocimiento del conjunto \mathcal{D} , el cual contiene un conjunto finito de datos, nos puede dar información suficiente como para conocer la función f en todo su conjunto de definición. Recordemos que el conjunto \mathcal{X} puede ser el espacio de definición de un vector aleatorio. Así pues, no podremos decir qué ocurre fuera de

nuestro conjunto \mathcal{D} . Pese a esto, veamos que al menos estadísticamente si podremos inferir el valor de f , es decir, es posible el aprendizaje.

Para resolver esta incógnita, vamos a utilizar la desigualdad de **Hoeffding**[4]:

Lema 3.1 Desigualdad de Hoeffding. *Sea x_1, \dots, x_N una muestra aleatoria simple(m.a.s.), donde cada x_i sigue una distribución de Bernouilli con media μ . Sea $\tilde{x} = \frac{1}{N} \sum_{i=1}^N x_i$ la media muestral asociada a la m.a.s.. Entonces*

$$\mathcal{P}[|\mu - \tilde{x}| > \epsilon] \leq 2e^{-2\epsilon^2 N}$$

Esta desigualdad nos da una cota para el error cometido al intentar estimar la media de una distribución de Bernouilli por la media muestral de una m.a.s. pero ¿Qué tiene que ver esto con el aprendizaje?

Volviendo al problema del aprendizaje de una función objetivo $f: \mathcal{X} \rightarrow \mathcal{Y}$, sea $h \in \mathcal{H}$ una hipótesis en concreto. Definimos ahora una nueva variable aleatoria $\phi: \mathcal{X} \rightarrow \{0, 1\}$, $\phi(x) = 1$ si $h(x) = f(x)$. $\phi(x) = 0$ si $h(x) \neq f(x)$. La función ϕ seguirá entonces una distribución de Bernouilli con una media μ determinada. Las instancias x_i hacen a su vez de m.a.s., en caso de haber sido escogidas de forma independiente. El problema del aprendizaje se ha reducido a estimar entonces la media de una distribución de Bernouilli con un error pequeño. Si estimamos ϕ y nuestro su media es cercana a cero podremos decir que la función g aproxima bien a nuestra función objetivo. En caso contrario, diremos que no lo hace.

En un problema real, tendremos un espacio de búsqueda \mathcal{H} de donde extraer una h cuya función ϕ asociada tenga una media lo más cercana a 0 posible. Veamos ahora cómo establecer una búsqueda de solución óptima al considerar varias hipótesis.

Definición 3.1 *Sea $h \in \mathcal{H}$. Llamamos $E_{in}(h)$ a la fracción del conjunto de prueba \mathcal{D} en el cual h y la función objetivo f difieren, es decir:*

$$E_{in}(h) = \frac{1}{N} \sum_{i=0}^N [[h(x) \neq f(x)]]$$

A si mismo, llamamos $E_{out}(h)$ a la probabilidad de que h y f sean distintos, esto es:

$$E_{out}(h) = \mathcal{P}[h(x) \neq f(x)]$$

Nota: $[[sentencia]]$ vale 1 cuando 'sentencia' es cierta y 0 cuando es falsa.

Haciendo uso de nuestra notación anterior, $E_{in}(h)$ hace el papel de la media muestral. $E_{out}(h)$, por otra parte, es la esperanza de nuestra distribución de Bernouilli. Así pues, sustituyendo en la desigualdad de *Hoeffding*:

$$\mathcal{P}[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N}, \epsilon > 0$$

Esto es, para cada h hipótesis, su ratio de acierto se aproxima con un error ϵ a la esperanza de acierto en el conjunto total \mathcal{X} con una probabilidad menor a $2e^{-2\epsilon^2 N}$, siendo N el tamaño de la muestra.

Ahora bien, nuestro problema no solo se basa en encontrar el ratio de fallo de una de las funciones $h \in \mathcal{H}$, si no de encontrar la que minimice la media de la variable aleatoria ϕ asociada. Sea g la función de hipótesis mínima, es decir, la que tiene menor media muestral de todas las hipótesis en \mathcal{H} . Supongamos que el número de hipótesis es M finito. Puesto que g es una de las funciones de hipótesis, se verifica que:

$$'|E_{in}(g) - E_{out}(g)| > \epsilon' \Rightarrow' |E_{in}(h_1) - E_{out}(h_1)| > \epsilon'$$

$$\textbf{OR}'|E_{in}(h_2) - E_{out}(h_2)| > \epsilon'$$

OR...

$$\textbf{OR}'|E_{in}(h_M) - E_{out}(h_M)| > \epsilon'$$

Y si llamamos \mathcal{B}_i a cada uno de estos sucesos, \mathcal{B}_g al primer suceso, podemos aplicar la propiedad subaditiva de la probabilidad y obtener la siguiente desigualdad:

$$\mathcal{P}[\mathcal{B}_g] \leq \mathcal{P}\left[\bigcup_{i=1}^N \mathcal{B}_i\right] \leq \sum_{i=0}^M \mathcal{P}[\mathcal{B}_i] \leq 2Me^{-2\epsilon^2 N}$$

A esta desigualdad la llamaremos desigualdad uniforme de *Hoeffding* y nos da una cota para una hipótesis que hayamos escogido de forma no aleatoria del conjunto \mathcal{H} (como es el caso de nuestro algoritmo de aprendizaje al escoger g como el que tenga la menor esperanza).

Capítulo 4

Multiview Machine Learning

En muchas tareas de análisis de datos, la forma de obtener información es a través de distintas medidas. Cada una de estas medidas por separado no es capaz de describir por completo el caso de estudio en la mayoría de los casos. En estos casos, las medidas tomadas pueden ser separadas en lotes o vistas. En esta sección se presenta la rama del aprendizaje automático que estudia la separación en vistas de la información, el *Multiview Machine Learning*, desarrollada. Se presenta también el problema de minimización asociado al aprendizaje supervisado *Multiview* para distintas funciones de coste. Por último, plantearemos la posibilidad de crear vistas artificiales para que este estudio no sea exclusivo de los casos en los que existan a priori dichas vistas.

4.1. Introducción al Multiview Machine Learning.

El *Multiview Machine Learning* es una rama importante del *Machine Learning*, presentada en 2013 por Sun[5] y desarrollada más ampliamente en 2019[6].

Durante el proceso de aprendizaje, el *Multiview Machine Learning* utiliza explícitamente distintas representaciones de la información y modela la relación entre ellas. Llamaremos **vista** a cada uno de estas representaciones. Aquí se listan algunos ejemplos de situaciones en las que se utilizan distintas representaciones:

- Datos sobre un paciente de hospital entre los cuales se encuentran imágenes espectrales(1) por un lado y datos sanguíneos(2) por otro. Las distintas representaciones son imágenes por un lado y valores numéricos por otro.
- Imágenes desde distintas perspectivas de un coche(frontal(1), lateral(2,

trasera(3)...). Las vistas atienden a la posición relativa de la cámara con respecto al coche.

- Características de imágenes separadas artificialmente. Se puede tratar una imagen de maneras distintas para extraer información de colores(1) y texturas(2).

Una primera idea para aprovechar la característica de múltiples vistas podría ser la concatenación de las mismas vistas. Es seguro que nos dará más información que cualquiera de las posibles vistas por separado. Sin embargo, esto elimina la posibilidad de obtener información de la estructura completa de las vistas. Una buena estructuración de la información puede darnos más conocimiento que la simple concatenación de información.

En general, se puede diferenciar los algoritmos *Multiview* en tres grandes grupos:

- **Co-training.** Este tipo de algoritmo se utiliza en aprendizaje semi-supervisado. Aprende distintos clasificadores y son entrenados para llenar los campos no etiquetados.
- **Co-Regularization.** También se utiliza en problemas de aprendizaje semi supervisado. Este tipo de algoritmos pretenden enfatizar que la probabilidad conjunta de todas las vistas debe seguir un patrón común, y penaliza la discordancia con este patrón.
- **Margin-Consistency o de máxima separación.** Al contrario que los otros dos tipos de algoritmos, este tipo pretende enfatizar que vistas distintas deben de tener comportamientos estadísticos distintos y, por tanto, cuanto más diferentes sean, más información podremos aprender de todas las vistas de forma conjunta. Se utiliza en problemas de aprendizaje supervisado.

El potencial que tiene el enfoque de múltiples vistas dentro del *Machine Learning* es enorme. Cada vez se incrementa más la cantidad de datos no estructurados de forma uniforme en el mundo real. Entre otros ejemplos, para analizar páginas webs podemos obtener datos de lo que hay escrito en ellas o sobre las imágenes que muestran. Así, cada vista ofrece información distinta y complementaria a las demás.

En este trabajo nos centraremos en el aprendizaje supervisado desde el enfoque *Multiview* aplicando el tercer tipo de algoritmos(Margin-consistency).

4.2. Métodos de máxima separación(margin-consistency)

Los métodos de máxima separación son muy útiles para la creación de clasificadores. Entre ellos, el algoritmo SVM(Support Vector Machine) es uno de los más importantes, el cuál propone un algoritmo que separa con una afinidad elementos de distinta clase, siendo además esta afinidad la más óptima posible. El enunciado del problema de optimización para clasificación binaria es el siguiente:

Ejemplo 4.1 Dado un conjunto de datos $\mathcal{L} = \{(x_l, y_l), l \in \{1, \dots, L\}\}$, se busca una función $f(x) = w^\top x + b$ que sea la solución de

$$\arg \min_w \sum_i w_i^2 = \|w\|^2$$

Donde se cumple la restricción siguiente:

$$y_l f(x_l) \geq 1, \forall l \in \{1, \dots, L\}$$

esto es, clasifica bien los datos del conjunto \mathcal{L} . En caso de que no se pueda cumplir esta restricción se puede reformular el problema introduciendo nuevos parámetros $\{\psi_i\}_{i \in \{1, \dots, L\}}$ como sigue:

$$\arg \min_w \sum_i w_i^2 + c \sum_i \psi_i = \|w\|^2 + c \sum_i \psi_i$$

Donde se cumple la restricción siguiente:

$$y_l f(x_l) \geq 1 - \psi_i, \forall l \in \{1, \dots, L\}, \psi_i \geq 0$$

Se puede generalizar este método a varias vistas, obteniendo así un primer método *Multiview*. Sin embargo, vamos a estudiar un ejemplo más interesante, el cuál finalmente utilizaremos en la parte práctica del trabajo.

4.2.1. Maximun Entropy discriminant(MED)

El método *Maximun Entropy Discriminant*(MED) es una combinación de métodos de máxima separación y métodos basados en aprendizaje estadístico. El método MED busca una función discriminante $L(x, \theta)$ parametrizada en θ que clasifica correctamente los casos del conjunto de datos. Esto lo consigue aprendiendo la distribución de los datos. Este problema se puede enunciar como

Ejemplo 4.2 Dado un conjunto de datos $\mathcal{L} = \{(x_l, y_l), l \in \{1, \dots, L\}\}$, se busca una función discriminante $L(x; \theta)$ que sea la solución de

$$\arg \min_{p(\theta, \gamma)} KL(p(\theta, \gamma) || p_0(\theta, \gamma))$$

Donde se cumple la restricción siguiente:

$$\int p(\theta, \gamma) [y_l L(x_l, \theta) - \gamma_l] d\theta d\gamma \geq 0$$

esto es, clasifica bien los datos del conjunto \mathcal{L} con un margen gamma_l. KL es la divergencia de Kulback-Leiber.

La divergencia de Kullback-Leiber[7] es una forma de medir lo parecida que es una distribución a otra. No debe confundirse con una medida de verdad, puesto que esta operación no es simétrica entre otras características. Se define como:

$$KL(P || Q) = - \int_{-\infty}^{\infty} p(x) \log \left(\frac{q(x)}{p(x)} \right) dx$$

donde P, Q son variables aleatorias y p, q son las respectivas distribuciones de probabilidad de cada una de ellas.

4.2.2. Generalización a varias vistas.(MVMED)

Se puede generalizar este problema cuando tenemos un problema donde hay dos o más vistas, es decir, el conjunto $\mathcal{L} = \{(x_l^v, y_l), l \in \{1, \dots, L\}, v \in \{1, 2\}\}$ para dos vistas. En este caso, tendremos que aprender dos funciones discriminantes, una por vista. Los parámetros γ también se pueden tratar como variables dependientes de la vista.

El problema para dos vistas es el siguiente:

Ejemplo 4.3 Dado un conjunto de datos $\mathcal{L} = \{(x_l, y_l), l \in \{1, \dots, L\}\}$, con $x_l = (x_l^1, x_l^2)$, se busca una función discriminante $L(x; \theta)$ que sea la solución de

$$\arg \min_{p(\theta, \gamma)} KL(p(\theta, \gamma) || p_0(\theta, \gamma))$$

Donde se cumple la restricción siguiente:

$$\int p(\theta^1, \gamma) [y_l L(x_l^1, \theta^1) - \gamma_l] d\theta^2 d\gamma \geq 0$$

$$\int p(\theta^2, \gamma) [y_l L(x_l^2, \theta^2) - \gamma_l] d\theta^2 d\gamma \geq 0$$

donde $(\theta^1, \theta^2) = \theta \in \Theta$ el conjunto de búsqueda de los parámetros.

El problema 4.3, al igual que 4.2, puede solucionarse con el método de los multiplicadores de Lagrange y, por tanto, por su generalización de *Karush–Kuhn–Tucker*, los cuales nos describirán una distribución de probabilidad $p(\theta) = \int p(\theta, \gamma) d\gamma$. Se puede entonces crear un clasificador conjunto como:

$$\hat{y} = \text{sign} \left(\frac{1}{2} \sum_{v=1}^2 \int p(\theta) L^v(x^v | \theta^v) d\theta \right)$$

También podríamos obtener un clasificador distinto por cada vista con la siguiente regla:

$$\hat{y}^v = \text{sign} \left(\int p(\theta) L^v(x^v | \theta^v) d\theta \right)$$

4.3. Creación de vistas artificiales

Cuando hablamos de vistas distintas en *Multiview Learning* parece muy natural separar en vistas distintas información que nos llega de distintas fuentes. Un ejemplo muy representativo es un documento audio visual, en el cuál podríamos separar por video(1) y sonido(2). Sin embargo, ¿Es esta la única separación válida? ¿Podríamos crear vistas que no sean tan naturales?

Sea $x \in \mathcal{X}$ con $x = \{x_1, x_2, \dots, x_N\}$. Una primera posibilidad para considerar vistas distintas es separar x en subconjuntos disjuntos, esto es, $x = (x^1, x^2)$ con $x^1 = (x_1, x_2, \dots, x_m), x^2 = (x_{m+1}, \dots, x_N)$ (cualquier reordenación de los elementos sería válida, al igual que considerar más vistas y no solo 2). Sin embargo esta separación no nos ofrece ninguna ventaja a considerar el punto x completo, no hay relación entre los datos de cada vista.

Otra posibilidad es usar **Núcleos de Aprendizaje**. Esta nueva técnica es equivalente a la creación de un set de características basadas en el input. Esta creación se realiza gracias a una función de núcleo especializada. El escoger una buena función de núcleo que separe el input en un set de características apropiado no es un problema trivial y será uno de los problemas que intentaremos resolver más adelante.

La creación de varios núcleos de aprendizaje equivale a extraer varias vistas distintas de nuestros datos sin necesidad de que sean tan naturales como los primeros ejemplos de la sección.

Capítulo 5

Computación numérica y problemas de optimización.

Usualmente, cuando trabajamos con algoritmos de Machine Learning estos requieren gran cantidad de cálculos numéricos. Esto se traduce en encontrar sucesivas aproximaciones las cuales se acercarán de forma iterativa a nuestra función de búsqueda. Este proceso es muy diferente a la visión analítica del problema, la cual busca la expresión simbólica de dicha función solución. Algunos algoritmos típicos son los de optimización o resolver sistemas de ecuaciones lineales, los cuales requieren gran cantidad de computo. Incluso evaluar una función de búsqueda en una computadora digital puede ser un problema si la función está evaluada en números reales, ya que no pueden ser expresados en una máquina con cantidad finita de información. Estudiaremos los problemas de Underflow y Overflow, asociados a la cuantificación de la información, y el problema más importante a la hora de encontrar solución a ecuaciones lineales, el condicionamiento de matrices. Se expondrá también las principales herramientas del análisis numérico de cálculo de mínimos, el gradiente descendente y algunas modificaciones más complejas. Por último, se plantea el problema de minimización con restricciones.

5.1. Overflow y Underflow.

Algunos de los problemas de que nos podemos encontrar son los asociados a la cuantificación de los números reales. Puesto que en casi ningún numero real su representación digital coincide con su verdadero valor, estaremos cometiendo errores de redondeo continuos. Si un algoritmo no está preparado para minimizar el posible error de redondeo asociado a esto, puede que el resultado esté lejos de ser una buena solución del problema.

Uno de los problemas de redondeo especialmente problemático es el conocido como **Underflow**. Este se produce cuando estimamos por cero números cercanos a cero. En este caso, si pretendemos dividir por esta estimación obtendremos un error, pese a que nuestro número original era muy pequeño, pero no nulo.

Otro error que cabe destacar es el **Overflow**. Este error aparece cuando números especialmente grandes en valor absoluto los sustituimos por falta de espacio por infinito.

Pongamos un ejemplo. Veamos que la función **softmax**, definida como:

$$\text{softmax} : \mathbb{R}^n \rightarrow (0, 1)^n$$

$$\text{softmax}(x_1, \dots, x_n)_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}$$

puede tener problemas de los tipos nombrados anteriormente si no se crea una rutina específica para ella. Supongamos que cada coordenada del vector X es igual a una constante c . En tal caso, analíticamente sabemos que el valor de la función softmax es $1/n$ para cada coordenada fina. Sin embargo, si evaluamos por partes cada una de las exponenciales y la constante c es suficientemente grande, cada término será sustituido por infinito, estaremos experimentando Overflow. Esto se puede solucionar si en vez de calcular la función softmax en el vector X lo evaluamos en $X - \{\max(X)\}^n$. Esto lo podemos hacer ya que la función softmax es invariante bajo esta transformación. La demostración se realiza dividiendo numerador y denominador por la exponencial del máximo de X . Tras ver que cada exponente de las exponenciales es cero a lo sumo, las exponenciales serán a lo sumo uno, no tendremos problemas de overflow.

Analíticamente, considerar el logaritmo de la función softmax no plantea ningún problema ya que por muy pequeño que sea el resultado, éste será positivo. Sin embargo, si ejecutamos la rutina de softmax y este nos devuelve un valor nulo, el logaritmo nos devolverá un error. Para aplicar el logaritmo de la función softmax habrá que definir una subrutina distinta para que esto no ocurra.

5.2. Condicionamiento

Cuando hablamos de condicionamiento, hablamos de cuánto puede cambiar el resultado de una función si modificamos ligeramente los valores iniciales. Un mal condicionamiento es un problema para el cálculo científico ya que los errores de redondeo considerados anteriormente pueden eliminar la validez de un cálculo posterior.

Sea $A \in \mathcal{M}_{n \times n}$ invertible. Se define el condicionamiento de una matriz bajo una norma matricial $\|\cdot\|_p$ asociada como:

$$\text{cond}(A) := \|A\|_p \|A^{-1}\|_p$$

que en caso de utilizar la norma matricial $\|\cdot\|_2$ tenemos que es igual a $\max\left\{\frac{|\lambda_i|}{|\lambda_j|}, \lambda_i, \lambda_j\right\}$ valores propios de la matriz A}. Cuanto más grande sea este número, más sensible es la inversión de matrices a cambios en la entrada. Esta característica de sensibilidad a cambios es intrínseca a la matriz.

5.3. Optimización basada en el gradiente.

Muchos problemas del Deep Learning están asociados a optimización de funciones. Esto es, encontrar una entrada x para la cual $f(x)$ sea máximo o mínimo global. En esta sección consideraremos el problema de minimización ya que el problema de maximización puede ser resuelto con la minimización de la función $-f(x)$.

A la función que querremos minimizar la llamaremos función objetivo. También se le llama función de pérdida, función de error o función de coste, dependiendo de las implicaciones que se tengan en cada contexto. Son términos prácticamente intercambiables.

Notaremos, en caso de existir, como x^* al elemento del dominio de definición de la función objetivo que minimiza a la misma, esto es $x^* = \arg \min(f(x))$. Supongamos que la función f es derivable y con derivada continua. El significado geométrico de la función derivada de f es la pendiente que tiene f en el punto en el que se evalúa. Una buena aproximación de la función f en un punto cercano a x , digamos $x + \varepsilon$ es $f(x + \varepsilon) \approx f(x) + \varepsilon f'(x)$. Esta aproximación se puede deducir de la definición de derivada para un ε lo suficientemente pequeño o a partir del desarrollo de Taylor a ambos lados de la igualdad.

De esta última igualdad aproximada, deducimos que el conocimiento que nos ofrece la derivada es útil para ver en qué dirección crece o decrece la función objetivo. Por ejemplo, para un ε suficientemente pequeño, $f(x - \varepsilon \text{sign}(f'(x)))$ es menor que $f(x)$. Un primer algoritmo iterativo de minimización puede ser considerar los sucesivos puntos $x_{n+1} = x_n - \varepsilon \text{sign}(f'(x_n))$, esto es, avanzar con un paso ε en la dirección de la derivada. A este primer procedimiento de la llama **gradiente descendente**.

Nótese que en caso de que la derivada se anule, ésta no nos da información de hacia donde avanzar. A estos puntos se les llama puntos críticos de la función. Estos puntos pueden ser mínimos relativos, máximos relativos o puntos de silla.

Un punto que obtiene el menor valor de f es llamado **mínimo global**. Puede haber uno o varios mínimos globales. Además, puede haber un mínimo local el cuál no sea mínimo global. Este último caso es con el que trabajaremos con Deep Learning. Por esta razón, aunque busquemos el mínimo absoluto de la función, nos conformaremos con un punto con un valor lo suficientemente cercano al mínimo absoluto.

La función que pretendemos minimizar suele utilizar varias entradas, aunque para que el concepto de minimización tenga sentido, debe haber una única salida, esto es, f está definida sobre los siguientes dominio y codominio:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

Para funciones con dominio de varias variables, el concepto de derivada se puede generalizar. Para ello, utilizamos el concepto de **derivada parcial**. Notaremos por $\frac{\partial f}{\partial x_i}(x)$ a la derivada parcial de f respecto a la variable x_i en el punto x . Esta función nos dice cuánto cambia la función si modificamos sólo la variable x_i . La forma de generalizar la derivada a dimensiones superiores es el concepto de **gradiente**. El gradiente es la función que contiene cada derivada parcial y lo denotamos como $\nabla_x f(x)$. El elemento i -ésimo del vector es la derivada parcial con respecto a la variable i -ésima con respecto a f en el punto x . Los puntos críticos en varias dimensiones están caracterizados como los puntos donde cada una de las componentes del gradiente son cero.

Se define la **derivada direccional** de f en la dirección u vector unitario como la derivada con respecto a α de la función $f(x + \alpha u)$ evaluada en $\alpha = 0$. Utilizando la regla de la cadena, podemos ver que $\frac{\partial f}{\partial \alpha}(x + \alpha u) = u^\top \nabla_x f(x)$.

Para minimizar nuestra función objetivo, querremos encontrar la dirección en la que ésta decrece más rápido. Es decir, encontrar la solución al siguiente problema de minimización:

$$\min_{\|u\|=1} u^\top \nabla_x f(x) \quad (5.1)$$

$$= \min_{\|u\|=1} \|u\| \cdot \|\nabla_x f(x)\| \cdot \cos(\theta) \quad (5.2)$$

Donde el ángulo θ es el ángulo en el que incide el vector unitario u y el gradiente de f . Puesto que la norma de u es 1 y la norma del gradiente es constante con respecto a u , el término que nos determina el valor de la función a minimizar es $\cos(\theta)$. Se puede deducir que minimizamos la función cuando la dirección de u es contraria al gradiente. A los métodos que proponen minimizar la función con pasos hacia la dirección que más decrece la función se conocen como basados en el **gradiente descendente**. En concreto, proponen un algoritmo iterativo de esta forma:

$$x_{n+1} = x_n - \varepsilon \nabla_x f(x_n)$$

donde ε es el ratio de aprendizaje.

5.4. Uso de las segundas derivadas: Matriz Hessiana.

El método que hemos utilizado hasta ahora se basaba en conocer hacia donde decrecía la función objetivo. Utilizamos la aproximación lineal de la función objetivo en un entorno del punto y avanzamos hacia la dirección que más descendía. En esta sección vamos a utilizar una aproximación cuadrática de la función. Para ello, haremos uso de la **matriz Hessiana** de la función.

Dada $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, llamamos **segunda derivada** de f con respecto a las variables x_i, x_j , y lo denotamos como $\frac{\partial^2 f}{\partial x_i \partial x_j}$, como la derivada parcial con respecto a x_j de la derivada parcial de f con respecto a x_i . En una sola dimensión, denotamos a la segunda derivada con respecto a la única variable x_i como $f''(x)$. La interpretación geométrica de esta segunda derivada nos describe cuál es la **curvatura** de la función.

El estudio de la curvatura nos puede ayudar para escoger un ratio de aprendizaje adecuado en cada paso y no arbitrario, ya que hasta ahora no teníamos ninguna restricción con respecto a este parámetro.

Para funciones de varias variables, el papel de la segunda derivada para una sola variable la hace la **matriz Hessiana**. La matriz Hessiana de una función $f : \mathbb{R}^n \rightarrow \mathbb{R}$ dos veces derivable en un punto x , notada por $H(f)(x) \in \mathcal{M}_{n \times n}$, se define como:

$$H(f)(x)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}(x)$$

En caso de que las derivadas parciales sean continuas, se verifica que podemos intercambiar el orden de derivación, esto es:

$$\frac{\partial^2 f}{\partial x_j \partial x_i}(x) = \frac{\partial^2 f}{\partial x_i \partial x_j}(x)$$

Esto implica que la matriz Hessiana sea simétrica. En el ámbito del Deep Learning, la mayoría de funciones tienen matriz Hessiana simétrica y real-evaluada, por lo que podemos sustraer un conjunto de valores propios reales con sus respectivos vectores propios ortogonales entre si.

Para cada vector unitario d , la segunda derivada direccional sobre esa dirección está dada por $d^\top H d$. Obtenemos así una medida de la curvatura en la dirección d , lo cual nos da una idea de la bondad con la que podemos

esperar que el gradiente descendente funcione. Así, podemos extender la función f con la serie de Taylor de segundo orden:

$$f(x) \approx f(x_{(0)}) + (x - x_{(0)})^\top g + \frac{1}{2}(x - x_{(0)})^\top H(x - x_{(0)})$$

donde H es la matriz Hessiana y g el gradiente. Evaluando en $x_{(0)} - \varepsilon g$:

$$f(x_{(0)} - \varepsilon g) \approx f(x_{(0)}) - \varepsilon g^\top g + \frac{1}{2}\varepsilon^2 g^\top H g$$

Los tres términos que aparecen en esta aproximación se pueden interpretar como el valor inicial de la función f en nuestro punto de origen, la mejora esperada al avanzar una distancia ε en la dirección g y un último término de corrección basado en la curvatura de la función. Cuando el último término es demasiado grande, el método del gradiente descendente puede incluso terminar incrementando nuestra función objetivo, lo cual no nos conviene. En la práctica y de forma heurística, puesto que la aproximación con la serie de Taylor pierde validez cuanto más lejos del punto $x(0)$ nos encontramos, utilizaremos $\varepsilon = \frac{g^\top g}{g^\top H g}$. Esto nos asegura que, en el peor de los casos, cuando la dirección de g pertenezca al subespacio propio del valor propio mayor, el tamaño de paso estará dado por $\frac{1}{\lambda_{max}}$.

Volviendo al punto de vista analítico, la segunda derivada de una función real de variable real nos puede dar información para determinar si un punto con derivada nula es un mínimo o máximo relativo. Si la segunda derivada es positiva en un extremo relativo, podemos deducir que a la izquierda de dicho punto la función decrece y que a la derecha, crece:

$$0 < f''(x) = \lim_{h \rightarrow 0} \frac{f'(x+h) - f'(x)}{h} = \lim_{h \rightarrow 0} \frac{f'(x+h)}{h}$$

Hemos deducido que nuestro punto es un mínimo relativo. Se razona de manera análoga para la segunda derivada con valor negativo y se concluye con que nuestro punto es un máximo relativo. Si la segunda derivada es nula, este test no nos puede dar información sobre qué tipo de punto es el que estamos analizando, siquiera un punto de silla.

Este test de la segunda derivada se puede generalizar con la matriz Hessiana. Dependiendo de los valores propios de la matriz, podremos inferir si un punto es máximo o mínimo. En caso de que la matriz Hessiana sea definida positiva(resp, negativa), el punto será un mínimo(resp. máximo) relativo. En dimensiones superiores, se pueden encontrar evidencias de que el punto escogido es un punto de silla. Por ejemplo, si la matriz Hessiana tiene al menos un valor propio negativo y otro positivo, las derivadas direccionales en tales direcciones definen una función creciente y otra decreciente en cada una de ellas.

En base a todo este conocimiento, se propone el siguiente **método de Newton** basado en el desarrollo en serie de Taylor de segundo orden:

$$f(x) \approx f(x_{(0)}) + (x - x_{(0)})^\top \nabla_x f(x_{(0)}) + \frac{1}{2}(x - x_{(0)})^\top H(f)(x_{(0)})(x - x_{(0)})$$

Si resolvemos para el punto crítico de esta expresión obtenemos:

$$x^* = x_{(0)} - H(f)(x_{(0)})^{-1} \nabla_x f(x_{(0)})$$

Si la función f es cuadrática, nuestro método avanzará en un único paso al extremo de la función. En caso de ser aproximable localmente por una función con Hessiana definida positiva, el método consistirá en iterar la ecuación anterior, acercándonos al mínimo de la función.

El tipo de métodos que utilizan exclusivamente el gradiente para la optimización de funciones, como el gradiente descendente, son llamados métodos de **primer orden**, mientras que los métodos que utilizan también información de la matriz Hessiana, como el método de Newton, son llamados de **segundo orden**, ya que basan su estrategia en la aproximación de Taylor de primer y segundo orden.

5.5. Optimización con restricciones.

En determinadas ocasiones, la búsqueda del extremo absoluto de nuestra función f la realizaremos en un espacio más restringido que el de la definición de f . Por ejemplo, encontrar el máximo de f en la esfera unidad debe ser posible ya que esta esfera es compacta. Sin embargo, puede que f no tenga ningún punto crítico en ella. El método que utilizaremos es la aproximación de **Karush–Kuhn–Tucker**, que supone una generalización del método de los multiplicadores de Lagrange.

Definición 5.1 Sean \mathbb{S} un conjunto de puntos verificando p inecuaciones y k ecuaciones, es decir, $\mathbb{S} = \{x \in \mathbb{R}^n : h_i(x) \leq 0, g_j(x) = 0, \forall i \in 1, 2, \dots, p, \forall j \in 1, 2, \dots, k\}$. Definimos el **Lagrangiano Generalizado** como:

$$\begin{aligned} L : \mathbb{R}^n \times (\mathbb{R}_0^+)^p \times \mathbb{R}^k &\rightarrow \mathbb{R} \\ (x, \lambda, \alpha) &\rightarrow f(x) + \sum_i \lambda_i h_i(x) + \sum_j \alpha_j g_j(x) \end{aligned}$$

Teorema 5.1 En las condiciones anteriores y suponiendo que el conjunto \mathbb{S} no es vacío, los puntos óptimos de la función f en el conjunto \mathbb{S} son los mismos puntos óptimos de la siguiente expresión:

$$\min_x \max_{\lambda} \max_{\alpha} L(x, \lambda, \alpha) = \min_{x \in \mathbb{S}} f(x)$$

Demostración. Podemos comprobar que si x no verifica alguna de las restricciones impuestas, el valor máximo que puede tomar $L(x, \lambda, \alpha) = \infty$ ya que si no cumple la igualdad(resp. desigualdad) i -ésima, haciendo crecer la componente i -ésima del vector λ (resp. α) tenemos que crece indefinidamente. En caso de que las restricciones se cumplan, se tiene que :

$$L(x, \lambda, \alpha) = f(x) + \sum_i \lambda_i h_i(x) \leq f(x)$$

ya que $g_i(x) = 0 \forall i = 1, 2, \dots, n$. Además, vemos que el máximo en λ se alcanza para $\lambda = 0$, entendiendo 0 como el vector 0. Y por tanto, los mínimos coinciden.

Este teorema nos da, por tanto, un método para calcular mínimos en conjuntos con restricciones.

Capítulo 6

Deep Feedfordward Networks

Los modelos Deep Feedfordwars Networks, también llamados redes neuronales prealimentadas o perceptrón multicapa, son de los modelos por excelencia en el ámbito del Deep Learning. Su objetivo es aprender una función objetivo f^* por medio de funciones parametrizadas en un conjunto Θ . Por ejemplo, para un clasificador que separe datos $y = f^*(x)$, una red neuronal prealimentada aprenderá una función $f(x; \theta), \theta \in \Theta$, escogiendo la mejor parametrización.

Estos modelos se llaman **prealimentados** porque la información fluye desde el input, x , hasta el output, y , sin que dentro de la función pueda haber posibles bucles en los que la información pase varias veces, no hay **recurrencia**. A las redes neuronales en las que pueda haber recurrencias las llamaremos **redes neuronales recurrentes**.

A las redes neuronales se las llama **redes** porque suelen ser representadas como una composición de funciones. Este tipo de modelos se suelen representar como un grafo acíclico, describiendo cómo se componen estas funciones. Por ejemplo, nuestra función f podría estar representada como la composición de otras tres funciones, es decir $f(x) = f^3(f^2(f^1(x)))$.

A cada función la llamaremos capa de la red neuronal, distinguiéndolas entre ellas por un cardinal asociado a lo profunda que esté la capa. Esto es, a la función f^1 la llamaremos primera capa, a la f^2 , segunda capa... El cardinal de la última capa determina la **profundidad** de la red neuronal, siendo en este caso una red neuronal de 3 capas de profundidad. A si mismo, a la última capa la llamaremos capa de salida. A las capas que no son ni la primera ni la última, las llamaremos capas ocultas.

Finalmente, el nombre de **neuronal** viene dado porque es un método inspirado en la neurociencia. Cada una de las capas son $f^i : \mathbb{R}^n \rightarrow \mathbb{R}^m$, con

una salida de dimensión m . A este m lo llamaremos *anchura* de la capa. A cada una de las componentes del vector se la puede interpretar como una neurona, entendiendo ésta como una unidad que procesa las n entradas y devuelve una única salida. Las neuronas reales tienen un procedimiento parecido.

Normalmente trataremos con redes neuronales *totalmente conectadas*, esto es, cada neurona de una capa recibe como entrada toda la salida de la capa anterior, como vemos en la figura 6.1. Esta red neuronal tiene nivel de profundidad dos y tiene anchura 3.

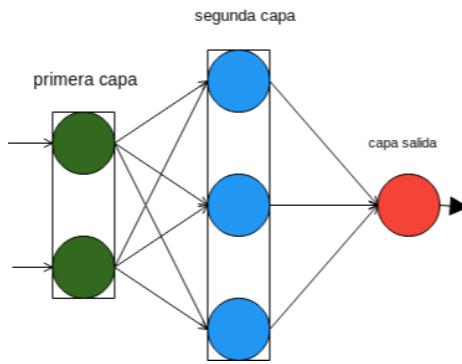


Figura 6.1: Ejemplo de arquitectura de una red neuronal

Para la neurona j -ésima de la capa i -ésima tenemos determinada una función $f^{i,j} : \mathbb{R}^{n_{i-1}} \rightarrow \mathbb{R}$ asociada, donde n_{i-1} es la anchura de la capa anterior. Así, cada capa puede verse como una función vectorial $f^i = (f^{i,1}, f^{i,2}, \dots, f^{i,n_i})^\top$. De la misma forma, la función que realiza nuestra red neuronal f se puede ver como concatenación de estas funciones: $f : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_m}$, $f(x) = (f^m \circ f^{m-1} \circ \dots \circ f^1)(x)$.

Como todos los métodos de aprendizaje, la red neuronal aprenderá una serie de pesos de entre un conjunto Θ que aproxime a nuestra función objetivo, esto es, una $f(x; \theta) \approx y$, donde nuestra función objetivo verifica $f^*(x) = y$. Para este cometido, utilizaremos el gradiente descendente, consiguiendo los pesos óptimos para nuestro problema.

Para entender la necesidad de crear un modelo complejo como el actual pasa por comprender los problemas de modelos más simples. Un ejemplo son los modelos lineales, tales como la regresión lineal o la regresión logística. Son muy sencillos de optimizar, sin embargo su capacidad está limitada a funciones lineales.

Para suplir la falta de capacidad de nuestro modelo lineal, se propone aplicar una transformación ϕ no lineal para luego poder separar, ahora si, de forma lineal nuestro espacio transformado. La interpretación más directa de esta transformación es que estamos separando en un conjunto de carac-

terísticas la instancia x , $\phi(x)$, las cuales esperamos que representen mejor nuestra instancia x .

Así pues, nuestro trabajo consistirá en encontrar una función $\phi(x; \theta)$ que separe en características adecuadas nuestras instancias.

Como en otros modelos de aprendizaje, este modelo necesita una función de coste, el método de optimización y la salida de la red. Además tendremos que tomar decisiones como la anchura y profundidad de la red, así como las **funciones de activación** de cada neurona. El método de optimización más utilizado para las redes neuronales es el algoritmo llamado **back-propagation**, el cuál explicaremos más adelante. Este algoritmo utiliza de forma eficiente el algoritmo del gradiente descendente explicado anteriormente.

6.1. Función de coste

Recordando que un problema de optimización podía interpretarse como la búsqueda de distribución de probabilidad $p(y|x; \theta)$ o un estadístico de la misma. Para obtener una buena aproximación de la misma, utilizaremos el principio de máxima verosimilitud.

La función de verosimilitud se define en base a una muestra aleatoria simple o conjunto de datos $(x_1, \dots, x_n) = x$ de una distribución con parámetro θ_0 . Si los posibles valores de este parámetro están en Θ , se define la función de **verosimilitud** como:

$$\mathcal{L} : \Theta \rightarrow [0, 1], \mathcal{L}(\theta; x) = \prod_{i=1}^n f(x_i | \theta)$$

El parámetro $\hat{\theta} \in \Theta$ que maximice esta función es el parámetro que hace más probable nuestra muestra. Para encontrarlo, en vez de derivar directamente nuestra función de verosimilitud, derivaremos otra función que tendrá máximo en el mismo punto. Gracias a la concavidad del logaritmo, podemos garantizar que el máximo de $\mathcal{L}(\theta)$ y la de su logaritmo es el mismo. Así pues, la función que maximizaremos será:

$$\log(\mathcal{L}(\theta; x)) = \log\left(\prod_{i=1}^n f(x_i | \theta)\right) = \sum_{i=0}^n \log(f(x_i; \theta))$$

Esta definición nos proporciona una función de coste genérica, ya que no depende del modelo que estemos tratando.

El objetivo que nos planteamos en este momento es el de minimizar la diferencia entre el valor real de cada muestra (x_i, y_i) , y el valor que le da nuestro modelo, esto es, minimizar la siguiente expresión:

$$\mathbb{E}_{x,y \sim \hat{P}_{data}} [|\log(\hat{P}_{data}(y|x)) - \log(P_{model}(y|x))|]$$

Puesto que hay términos que no intervienen en el algoritmo de aprendizaje, podemos definir la siguiente función de coste:

$$J(\theta) = -\mathbb{E}_{x,y \sim \hat{P}_{data}} [\log(P_{model}(y|x))]$$

Donde $\mathbb{E}_{x,y \sim \hat{P}_{data}}$ denota la esperanza muestral que obtenemos empíricamente. Es decir, utilizaremos la entropía cruzada de los datos de muestra y las predicciones que el modelo asigna como función de coste.

Un problema con el que nos podremos encontrar a la hora del diseño de redes neuronales es que el módulo del gradiente no sea lo grande que deseáramos o no indique con demasiada bondad el lugar hacia el que mejorar. Funciones que se saturan, es decir, toman valores muy próximos a cero, pueden dar lugar a gradientes con modulo casi cero. Esto ocurre porque algunas funciones de activación utilizan la exponencial, la cual se vuelve prácticamente cero cuando el valor introducido es muy negativo. El logaritmo de la función de máxima verosimilitud solucionaría este problema.

Un problema que nos podemos encontrar al utilizar la entropía cruzada para calcular la estimación de máxima verosimilitud es que puede no existir un valor mínimo de entrada, haciendo decrecer hasta el infinito algunos valores, haciendo que nuestro modelo no pueda trabajar bien. Algunas técnicas de regularización limitarán este posible decrecimiento.

6.2. Unidades de salida.

La decisión de escoger una función de coste está íntimamente relacionada con la decisión de tomar una u otra capa de salida. Utilizaremos la función de entropía cruzada entre la distribución de nuestro modelo y la de los datos de prueba, así que la configuración de nuestro modelo determinará la función de coste asociada.

Cualquier tipo de unidad de salida puede ser utilizada como unidad oculta, pero en esta sección nos vamos a centrar en estas unidades como de salida, por su interpretabilidad.

Supondremos que, al ser una red neuronal prealimentada, las capas anteriores nos ofrecen características ocultas $h = f(x; \theta)$ que nos servirán de entrada para la neurona de salida, y nuestra unidad tendrá que procesar de alguna manera la entrada h para completar la tarea que le encargaremos realizar.

6.2.1. Unidades lineales

Las unidades lineales computan una aplicación lineal sobre $h \in \mathbb{R}^m$, asociando a h la transformación $Wh + b$, donde $W \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^n$. Los parámetros asociados a esta capa serán los valores W y b .

Esta unidad es lineal y no satura, por lo que no tiene problemas con distintos métodos de aprendizaje basados en el gradiente.

6.2.2. Unidades sigmoidales

Para casos en los que queremos inferir una variable binaria querremos definir una distribución de probabilidad de y condicionada a x . Puesto que la variable es binaria, sólo tendremos que calcular la probabilidad de que y alcance uno de los valores.

Podemos definir, en base a una unidad lineal, una distribución de Bernouilli como sigue:

$$P(y = 1|x) = \max(0, \min(1, Wh + b))$$

Efectivamente, lo que hemos definido arriba es una probabilidad, ya que es creciente, continua a la derecha y toma valores entre 0 y 1. Sin embargo, puesto que tiene derivada nula cuando el cálculo $Wh + b$ queda fuera del intervalo $[0,1]$, no nos sirve para nuestro objetivo de aprendizaje basado en el gradiente. Una alternativa para solventar este problema es utilizar la función sigmoide definida a continuación:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

La salida de nuestra unidad sigmoide sería como sigue:

$$\hat{y} = \sigma(Wh + b)$$

Llamemos z al resultado de aplicar la transformación lineal a nuestro vector oculto h , $z = Wh + b$. Supongamos que tenemos una medida de probabilidad no normalizada \hat{P} en y , esto es, que no suma uno. Podemos dividir por la cantidad adecuada a posteriori para conseguir una probabilidad real. Suponiendo que las probabilidades aplicando el logaritmo son lineales en y y en z , lo cual nos permite exponenciar y luego dividir por una constante adecuada para obtener una distribución de Bernouilli controlada por el sigmoide:

$$\log \hat{P}(y) = yz$$

$$\hat{P}(y) = e^{yz}$$

$$P(y|z) = \frac{e^{yz}}{\sum_{i=0}^1 e^{iz}} = \frac{e^{yz}}{e^{0z} + e^{1z}} = \frac{1}{e^{-yz} + e^{z(1-y)}}$$

o lo que es lo mismo,

$$P(y|z) = \begin{cases} \frac{1}{1+e^z} = \sigma(-z) & \text{si } y = 0 \\ \frac{1}{e^{-z}+1} = \sigma(z) & \text{si } y = 1 \end{cases} = \sigma((2y - 1)z)$$

La función de coste asociada a esta configuración de salida siguiendo el enfoque de máxima verosimilitud quedaría definida como:

$$J(\theta) = -\log(P(y|x)) = -\log(\sigma((2y - 1)z)) = \xi((1 - 2y)z)$$

Donde ξ es la función conocida como *softplus* y es definida como $\xi(z) = \log(1 + e^z)$.

Esta función de perdida nos vuelve a plantear el problema de la saturación. Nuestra función satura únicamente cuando $(2y - 1)z$ es muy negativa, es decir, cuando $y = 1$ y z tiene valores muy positivo, o cuando $y = 0$ y z es muy negativo. Estos casos son los casos en los cuales nuestro modelo ya tiene la respuesta correcta, por lo que la falta de un gradiente no nulo no es perjudicial. Con otras funciones de pérdida, la función satura tanto si está en lo correcto como si no. Es el caso de considerar la función de coste del error cuadrático medio.

6.2.3. Unidad softmax para distribuciones multinomiales.

Cuando queramos extender la unidad anterior a distribuciones con n valores discretos, utilizaremos la función *softmax*.

En caso de la unidad sigmoidal, era claro que sólo necesitábamos calcular la probabilidad de $P(y = i|x)$ para un único i entre 1 y 0, ya que el otro valor se calcula como $1 - P(y = i|x)$. Para que la unidad softmax tenga sentido, debemos calcular cada una de las probabilidades condicionadas, es decir, queremos producir con la última salida un vector \hat{y} que cada componente esté entre 0 y 1 y que la suma de todos ellos sea 1, para que represente una probabilidad.

Al igual que en la unidad anterior, llamaremos $z = Wh + b$ al resultado de aplicar una transformación lineal determinada. Cada componente de z será interpretada como $z_i = \log \hat{P}(y = i|x)$, y la correspondiente salida será:

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=0}^n \exp(z_j)}$$

Como con la unidad logarítmica sigmoidal, el uso de la exponencial funciona bien cuando entrenamos una salida softmax con la función de coste máxima verosimilitud logarítmica evitando así la posible saturación. Calculando este coste de forma explícita, obtenemos

$$\log(\text{softmax}(z)_i) = z_i - \log \left(\sum_{j=1}^n e^{z_j} \right)$$

Cuando se intenta maximizar esta función de coste, el término z_i debe ser empujado a ser positivo y el resto de términos son empujados por el segundo sumando a ser negativos. En cuanto a este segundo sumando, cabe destacar que es aproximable por la función $\max_j(z_1, \dots, z_n)$. Esto se debe a que la exponencial de cualquier término z_k menor que el máximo es insignificante respecto al valor $e^{\max_j(x_j)}$. Así, se estará penalizando a la predicción incorrecta más inactiva. Por el contrario, si nuestra predicción ya es la correcta, tenemos que el logaritmo de la similitud será nulo y obtendremos saturación sólo en ese caso.

Hasta ahora sólo hemos visto la función aplicada a una sola instancia. La forma de generalizar la función de coste para un conjunto de datos o instancias es aproximar a través de unos parámetros θ el porcentaje de veces que se ha observado la salida i para un dato x de entrenamiento, es decir:

$$\text{softmax}(z(x; \theta))_i = \frac{\sum_{j=1}^m 1_{y^j=1; x^j} x^j}{\sum_{j=1}^m 1_{x^i=x}}$$

Las posibles limitaciones que nos encontraremos para encontrar la distribución correcta vendrá por problemas en el algoritmo de aprendizaje, el cual puede no encontrar el parámetro θ_0 máximo, y por parte de la limitación al considerar funciones dentro de un subconjunto de dimensión finita $\mathcal{H} \subset \mathcal{F}$ de todas las funciones posibles.

6.2.4. Otras unidades de salida

Hemos hecho una revisión de las unidades de salida más utilizadas. Se pueden construir otras unidades más complejas o considerar nuevas unidades de salida. Para la mayoría de éstas, el método derivado del principio de máxima verosimilitud nos ofrece una función de coste adecuada. Para una posible distribución $P(y|x; \theta)$, nos ofrece la función de coste $-\log P(y|x; \theta)$.

En algunos casos, puede no interesarnos la distribución de nuestra variable, sino un estadístico de la misma. Si $f(x; \theta) = w$ es dicho estadístico, podemos ver la función de coste como $-\log(P(y; w(x)))$.

6.3. Unidades ocultas

En esta sección hablaremos de una de las partes más características de las redes neuronales: las capas ocultas. En esta sección veremos algunas unidades ocultas y el cómo escogerlas para nuestro modelo. En este área de las redes neuronales todavía no hay una guía bien definida ni un sustento teórico robusto, por lo que es un campo abierto al estudio.

Como adelantamos en la sección anterior, las unidades de salida pueden ser utilizadas para servir de entrada para una próxima capa, convirtiéndolas en capas ocultas. Sin embargo, éstas no tendrán asociadas una función de coste asociada, por lo que no podremos utilizar directamente el principio de máxima verosimilitud. Veremos en la próxima sección un algoritmo para propagar el error para modificar los pesos de estas capas ocultas.

A no ser que se diga expresamente lo contrario, una unidad oculta aceptará un vector $x \in \mathbb{R}^n$ como entrada y le aplicará una transformación lineal $z = Wx + b$, $W \in \mathcal{M}_{n \times m}(\mathbb{R})$, $b \in \mathbb{R}^m$, siendo esta matriz W y este vector b los parámetros entrenables. A este vector $z \in \mathbb{R}^n$ le aplicará una función no lineal $g : \mathbb{R} \rightarrow \mathbb{R}$. La mayoría de capas ocultas se distinguen casi únicamente por la función no lineal que se haya escogido para definirla.

6.3.1. Unidades lineales rectificadas y generalizaciones(ReLU)

La función de activación escogida para esta capa será la función $g(z) = \max\{0, z\}$. Esta función de activación es fácil de optimizar ya que es muy parecida a una aplicación lineal. Además las posibles segundas derivadas son nulas, y los efectos de rectificación que las segundas derivadas tendrían en el gradiente no se aplican.

Usualmente, la función ReLU de activación se aplica después de una transformación lineal. La composición de estas funciones quedaría es la función $g(Wx + b)$. Una buena práctica para que los primeros valores en nuestro entrenamiento comiencen no siendo idénticamente nulos podría ser iniciar el vector b con elementos pequeños pero positivos.

Algunas generalizaciones a esta función de activación pasan por incorporar un parámetro α como pendiente de nuestra unidad cuando el valor de z es menor que cero, esto es:

$$g(x; \alpha) = \max\{0, z\} + \alpha \min\{0, z\}$$

Tres de las posibles funciones de activación basados en escoger este parámetro distinto de cero pueden ser:

- $\alpha = -1$. Esta modificación convierte $g(x)$ en el valor absoluto de x .
- $0 < \alpha << 1$. Esta función es llamada Leaky ReLU y se basa en escoger un valor de α muy pequeño, como 0,01.
- **α paramétrico.** A esta función la llamaremos parametric-ReLU(PReLU), ya que toma α como parámetro entrenable.

Otras unidades que generalizan la función de activación ReLU es la función **maxout**, que en vez de ser una función que se aplica elemento a elemento, esta función divide los elementos del vector z en grupos de k elementos y devuelve el máximo.

6.3.2. Unidades Sigmoide logistico y tangente hiperbólica.

Además de las introducidas unidades lineales rectificadas, otras funciones de activación de interés son las que tienen como función de activación la función sigmoide:

$$\sigma(z) = \frac{e^z}{1 + e^z}$$

o la la función tangete hiperbólica:

$$\begin{aligned} \tanh(z) &= \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z}}{e^{2z} + 1} - \frac{1}{e^{2z} + 1} = \\ &= \frac{e^{2z}}{e^{2z} + 1} + \frac{e^{2z}}{e^{2z} + 1} - \frac{e^{2z} + 1}{e^{2z} + 1} = 2\sigma(2z) - 1 \end{aligned}$$

Ya hemos visto la función de activación sigmoidal como unidad de salida y hemos visto que satura en casi todo el dominio y es sensible únicamente en un entorno cercano al cero. Esto causará problemas en el cálculo del gradiente ya que lo hará prácticamente nulo. Anteriormente, una buena elección de función de coste puede solucionar este inconveniente para neuronas de salida, sin embargo para neuronas ocultas no tenemos esta posibilidad.

Usualmente, la función de activación sigmoidal funciona peor que la tangente hiperbólica. Esto se debe a que cerca de 0, la tangente hiperbólica es similar a la identidad. Por esta similaridad, será más facil entrenar esta unidad.

6.3.3. Otras unidades ocultas.

En la literatura, existen muchas unidades ocultas, pero son mucho menos frecuentes. Además, rara vez funcionan mejor que las visto hasta ahora. Veamos entonces qué tipo de unidades ocultas se trabajan actualmente.

Un tipo de unidades ocultas considera no utilizar ninguna función de activación y trabajar con funciones lineales. Esto imposibilita el aprendizaje de funciones no lineales, ya que es una composición de aplicaciones lineales. Aun así, esta concatenación de aplicaciones lineales puede ser útil para considerar menos parámetros entrenables en la red.

Otra opción es utilizar la función de activación *softmax*, explicada anteriormente. Como representa una distribución multinomial, se puede interpretar como un commutador dentro de la red. Las unidades ocultas que consideran esta función de activación suelen utilizarse en modelos complejos.

Otras funciones de activación relevantes son:

- Funciones de base radial, de la forma $g(x) = \exp\left(-\frac{1}{\sigma_i^2} \|W_{:,i} - x\|^2\right)$. Puesto que la función satura en casi todo x lejano al parámetro $W_{:,i}$, es difícil de optimizar.
- La función $\text{softplus}(x) = \log(1+e^x)$. Empíricamente se ha demostrado que, aunque estas funciones son diferenciables, funcionan peor que las funciones lineales o derivados de estas.
- Hard-tanh. esta función tiene el mismo codominio que la función tangente hiperbólica y se comporta de forma similar a la misma. Está definida como $\text{hard-tanh}(x) = \max(-1, \min(1, x))$.

Actualmente, el campo de la elección de funciones de activación es un campo abierto al estudio y con posibilidades de encontrar nuevas funciones de activación útiles.

6.4. Diseño de la arquitectura de la red

Otra de las características únicas de el modelo de red neuronal es la arquitectura de la red: la decisión de cuantas capas debe de tener la red, que anchura debe tener cada capa y cómo deben conectarse entre ellas.

Muchas de las redes neuronales estructuran la red de forma encadenada, es decir, cada salida de una capa sirve como entrada de la siguiente. Así, una configuración genérica de esta arquitectura constaría de una primera capa alimentada con la entrada de la red:

$$h_1 = g_1(W_1x + b_1)$$

y las siguientes capas trabajarían como sigue:

$$h_n = g_n(W_n h_{n-1} + b_n)$$

para cada $n \in \{2, 3, \dots, N\}$, donde N es la profundidad de la red.

Existen otras configuraciones de redes que no están completamente encadenadas. Un ejemplo de ello será la aplicación del Multiview Learning que veremos en el trabajo experimental.

En el siguiente apartado veremos que una red neuronal con una única capa oculta es capaz de aprender cualquier función, con la suficiente anchura. Aun así, las configuraciones con varias capas y menos parámetros tienen más capacidad de generalización al conjunto de test, aunque son más difíciles de optimizar.

6.4.1. Teorema de aproximación universal.

Puesto que la composición de aplicaciones lineales resulta en una aplicación lineal, las capas lineales sólo pueden llegar a representar funciones lineales, nuestro objetivo es que nuestra red aprenda cualquier tipo de función. Se podría pensar que para aprender este tipo de funciones no lineales habría que diseñar un sistema de funciones no lineales de la misma forma. Sin embargo, el siguiente teorema nos asegura que una red neuronal con una capa oculta con una función de activación 'buena' y una capa de salida lineal puede representar cualquier función medible con el error que queramos:

Teorema 6.1 *Sea $\phi : \mathbb{R} \rightarrow \mathbb{R}$ una función continua, creciente y acotada. Sea $f \in \mathcal{C}([0, 1]^n, R)$ una función medible. Entonces, dado un $\varepsilon > 0$, existen un número natural N , constantes $\alpha_i, b_i \in \mathbb{R}$, y vectores $w_i \in \mathbb{R}^n$ tal que, si definimos*

$$F(x) = \sum_{i=0}^N \alpha_i \phi(w_i^\top x + b_i)$$

entonces la diferencia entre la función F y f está uniformemente acotada por ε , es decir:

$$\|F(x) - f(x)\| < \varepsilon \quad \forall x \in [0, 1]^n$$

en otras palabras, las funciones de la forma de F son densas en el espacio de funciones continuas evaluadas en $[0, 1]^n$.

Gracias a este teorema podemos afirmar que al menos existe una configuración de alguna red neuronal que puede representar infinitamente bien la función que queramos. De hecho se puede demostrar que las funciones de activación ReLU, explicados en apartados posteriores, generan también funciones densas. La demostración se basa en construir una función acotada, continua y creciente basada en dos funciones ReLU.

Sin embargo, esto no nos asegura que el método de aprendizaje vaya a aprender dicha configuración, aunque la estructura lo permita. Tampoco nos asegura la capacidad de generalización al conjunto de test.

6.4.2. Otras consideraciones de la arquitectura.

Algunas redes neuronales han sido desarrolladas para eliminar la cantidad de parámetros que tiene una red neuronal o para una tarea en específico. Por ejemplo, las redes neuronales convolucionales son un ejemplo de ello. En estas, cada unidad está conectada con un subconjunto de unidades de salida de la capa anterior, teniendo menos pesos que entrenar. Otro ejemplo son las redes neuronales retroalimentadas, las cuales crean bucles de flujo de información, eliminando la prealimentación de este modelo.

Otro método para utilizar menos parámetros y eliminar complejidad del modelo es eliminar algunas conexiones entre capas. En este caso, nuestras redes no estarán completamente conectadas, como si lo estaban anteriormente.

6.5. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son un tipo especializado de redes neuronales para procesar datos distribuidos con una topología de malla. Esto implica que no solo se podrá extraer información de algunos de los datos por separado si no también de la disposición de los mismos. El nombre **convolucional** viene dado por la operación que realiza entre capas.

Las redes convolucionales se utilizan actualmente para analizar series temporales, con una estructura de mallado 1-D, o para analizar información de imágenes, con una estructura 2-D. Estas redes han tenido una gran repercusión y han obtenido muy buenos resultados cuando tenemos este tipo de información que depende del contexto.

Una de las características principales de estas redes en el caso de dos dimensiones es que no están conectadas de la manera usual, hacen uso de

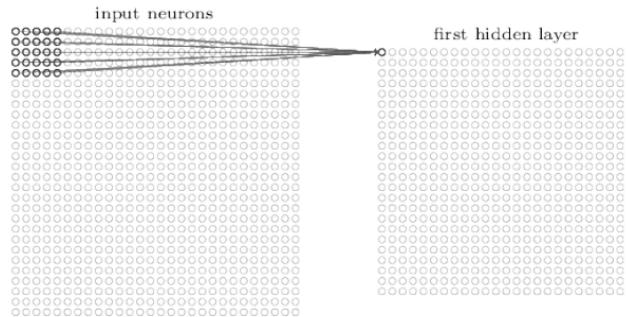


Figura 6.2: Aplicación de una operación de convolución a la primera ventana. Las dimensiones de la ventana son de 5×5 . Imagen obtenida de [8]

los conceptos de *localidad espacial*, *pesos compartidos* y *pooling*.

Cuando se dice de que una característica de estas redes es su localidad espacial, quiere decir que no se conecta todo el input a cada neurona de salida, sino sólo a pequeñas partes del mismo. Más concretamente, cada neurona tendrá una ventana de tamaños predefinidos n_1 y n_2 que aplicará a cada marco o submatriz del input. En 6.2 y en 6.3 se aplica la convolución a distintos marcos deslizando la ventana.

Cuando se habla de que estas redes comparten pesos se quiere decir que a cada ventana de aplicación se le somete a la misma operación, con los inputs respectivos. Esto se puede interpretar como una detección de un único patrón para cada marco de imagen. Puesto que se aplica la misma ventana a cada marco, esta operación es invariantes por traslaciones, dando más generalidad a la operación. Al aplicar esta operación, queda como resultado una matriz a la que llamaremos *mapa de características*.

Para cada capa se puede considerar varias ventanas distintas, utilizando distintos pesos y consiguiendo varios mapas de características. Esto producirá una red neuronal más compleja.

Otra capa importante para las redes convolucionales es la capa de *pooling*. En esencia, esta capa transcribe el mapa de características creado por la capa de convolución y la condensa. Por ejemplo, una capa de pooling podría utilizar marcos 2×2 de la imagen y devolver el mayor de los valores. A esta capa de pooling en concreto se le llama *max-pooling*.

Se pueden encadenar varias capas de convolución y max-pooling para crear una red todavía más compleja, haciendo también más complejo el patrón que la red puede identificar. Tras esta concatenación, se podrán encadenar capas de redes neuronales completamente conectadas de las que ya hemos estudiado y acomodarlas a la salida adecuada según la teoría tratada anteriormente.

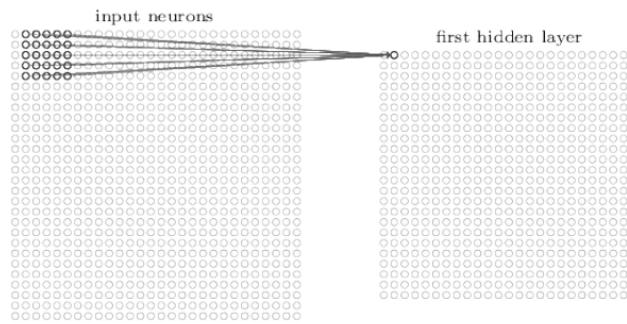


Figura 6.3: Aplicación de una operación de convolución a la segunda ventana.
Imagen obtenida de [8]

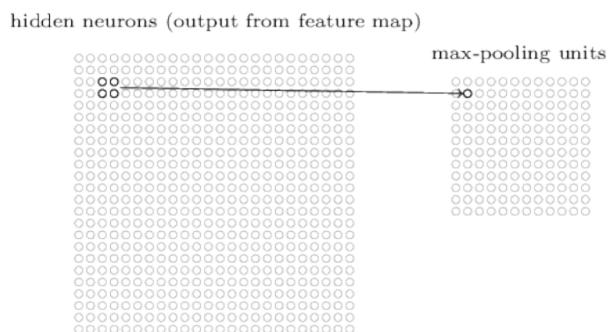


Figura 6.4: Capa de Max-Pooling de 2x2 aplicado a un mapa de características. Imagen obtenida de [8]

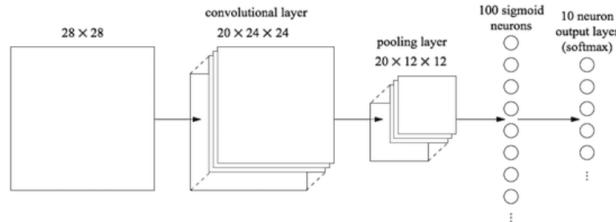


Figura 6.5: Red neuronal convolucional con una capa de convolución, una de max-pooling, una capa oculta completamente conectada y una capa softmax de salida. Imagen obtenida de

En 6.5 se observa un ejemplo de una red neuronal completamente configurada, con un input de una imagen de tamaño 28x28. Se aplica una primera capa de convolución con 20 ventanas de tamaño 5x5, tras la cual se aplica una capa de *max-pooling* de tamaño 2x2. Tras esto, la salida es conectada a 100 neuronas con función de activación sigmoidal y, finalmente, se conecta con la salida softmax de 10 neuronas.

6.6. Introducción al Transfer Learning

Las herramientas utilizadas para extraer conocimiento actuales han demostrado tener muy buenos resultados. Sin embargo, una vez construido un modelo con este conocimiento válido, la información puede quedar obsoleta en poco tiempo porque la distribución de nuestro conocimiento cambie. El concepto de *Transfer Learning* abarca todo el estudio de el conocimiento que se puede reutilizar de un modelo obsoleto y qué no. En esta sección se introduce este concepto y se presenta un caso práctico del que se puede extraer información: las redes pre entrenadas para el problema de imagenet.

En [9] se hace un estudio detallado sobre los conceptos básicos del *transfer learning*, aunque nosotros sólo vamos a utilizar la transferencia de la **representación de características**.

El objetivo de esta transferencia de información es crear un espacio de características que sean ‘buenas’ en el sentido de que diferentes problemas tengan una representación parecida en este espacio de características. Esto dependerá mucho del dominio donde esté definido el conjunto de datos.

En el caso del aprendizaje supervisado, el objetivo es encontrar un dominio con menos dimensiones en el que nuestro conjunto de datos esté bien representado y pueda ser separable en dicha representación. Hasta aquí no se diferencia en nada con el aprendizaje usual para extracción de característi-

Nombre de la red	Top 1 accuracy
ResNet50	0.749
InceptionV3	0.779
VGG16	0.713

Tabla 6.1: Nombre y accuracy de redes para el problema de Imagenet

cas. Lo novedoso es que si este espacio de menores dimensiones representa bien la información, este dominio de representación se puede utilizar para otro conjunto de datos distinto.

6.7. Transfer Learning en Imágenes: Problema imagenet

Imagenet [2] es un problema de clasificación de imágenes, con un conjunto de datos de más de 14 millones de imágenes pertenecientes a mil clases distintas, con más subclases dentro de ellas. Se ha trabajado mucho en la obtención de buenos modelos para la clasificación de este problema, obteniendo redes neuronales convolucionales muy complejas con muy buenos resultados. Estos resultados han sido propiciados por la competición asociada a este problema. Algunas redes famosas y sus resultados se detallan en la tabla 6.1.

Según el enfoque del *Transfer Learning*, los subespacios aprendidos por estos modelos deben representar bien otros problemas en los que se pretenda clasificar imágenes.

El *Transfer Learning* también estudia la información que no es transferible. Para otro problema de clasificación de imágenes se debe desechar la información que clasifica cada imagen en cada una de las 1000 clases del problema de imagenet y acomodarlo al nuevo problema.

Capítulo 7

Algoritmo de Back-propagation

Cuando utilizamos un modelo de red neuronal prealimentada, la entrada x es computado primero por la primera capa de la red. Tras esto, el resultado se ofrece a la siguiente como entrada, la cual será computada y ofrecida como próxima entrada para la siguiente capa hasta llegar a la última, la cuál nos devuelve la salida de la red \hat{y} . Esto se puede interpretar como una propagación de información **hacia adelante**, o **forward propagation**. En el proceso de entrenamiento, este resultado tendrá una función de coste asociada. El algoritmo **backpropagation** propaga este error desde el final hacia el inicio, viniendo de aquí su nombre. Se propuso este tipo de algoritmos por primera vez en [10]. Este método surge de la necesidad de calcular el gradiente de forma más eficiente, ya que su cálculo y posterior evaluación es bastante costoso si no se considera este algoritmo.

El término *backpropagation* se suele malentender. No es un algoritmo de optimización como tal si no un método de cálculo del gradiente de forma eficiente para su posterior aplicación en el método del gradiente descendente o similares. Otra interpretación que no coincide con la realidad de este método es que solo se utiliza para calcular el gradiente de redes neuronales. En realidad, el método permite calcular el gradiente $\Delta_x f(x, y)$ para cualquier función f , donde x es el conjunto de variables sobre las cuales querremos derivar e y un conjunto de variables de las cuales depende la función pero no querremos derivar respecto a ellas.

El modelo más genérico de *backpropagation* puede generalizarse para funciones con salidas múltiples, pero para nuestro caso supondremos que f solo tiene una salida.

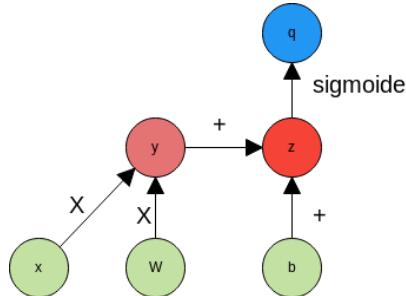


Figura 7.1: Grafo que representa la operación $\sigma(Wx + b)$. La arista con una X representa la operación binaria de multiplicar matricialmente el vector x y la matriz W , dando como resultado una nueva variable y . De igual forma, las aristas con un signo de suma representan la suma vectorial entre la variable b y la variable y resultante de la operación anterior, que también es binaria. Da como resultado la variable z . La última operación representa el aplicar la operación de activación sigmoide a z .

7.1. Grafos computacionales

En esta sección vamos a formalizar el lenguaje de grafos que hemos utilizado intuitivamente para hablar de redes neuronales.

- Para empezar, cada nodo de nuestro grafo será la representación de una variable de nuestro sistema.
- Las conexiones o aristas de nuestro grafo representarán operaciones. Estas pueden ser de una o varias variables del sistema(nodos) y deben ir a parar a otra de las variables. Distinguir entre las variables de entrada y las de salida lo haremos por la dirección que tenga la arista correspondiente.

Un ejemplo de estos grafos es el siguiente, en el que representamos el trabajo de una de las unidades básicas de una red neuronal:

7.2. Regla de la cadena para cálculo de derivadas.

La regla de la cadena en cálculo es utilizada para derivar composición de funciones. El algoritmo de **Back-propagation** utiliza esta regla para obtener el gradiente en un cierto orden y de forma eficiente.

Teorema 7.1 *Regla de la cadena(caso unidimensional)* Sea $\Omega_1, \Omega_2 \subset \mathbb{R}$ conjuntos abiertos y sean $f : \Omega_1 \rightarrow \mathbb{R}, g : \Omega_2 \rightarrow \mathbb{R}$, ambas funciones

derivables en los puntos x_0 y $f(x_0)$ respectivamente verificando $f(\Omega_1) \subset \Omega_2$, es decir, la composición $g \circ f$ está bien definida. Entonces la composición $g \circ f$ es derivable en x_0 y se verifica que

$$(g(f(x)))' = g'(f(x_0))f'(x_0)$$

Esto se puede reescribir en su forma diferencial, tomando $x \in \Omega_1, y = f(x), z = g(y) = g(f(x))$ se tiene

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

Teorema 7.2 Regla de la cadena (Caso n-dimensional) Sea $\Omega_1 \subset \mathbb{R}^n, \Omega_2 \subset \mathbb{R}^m$ conjuntos abiertos y sean $f : \Omega_1 \rightarrow \mathbb{R}^m, g : \Omega_2 \rightarrow \mathbb{R}$, ambas funciones diferenciables en un el punto x y $f(x)$ respectivamente verificando $f(\Omega_1) \subset \Omega_2$, es decir, la imagen de f cae en el conjunto de definición de g . Llamamos $y = f(x), z = g(y)$. Entonces la composición $g \circ f$ es diferenciable en x_0 y cada derivada parcial verifica

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^m \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

o, equivalentemente y expresado en forma vectorial:

$$\nabla_x z = \left(\frac{\partial y}{\partial x} \right)^\top \nabla_y z$$

El algoritmo *backpropagation* se aplica no solo a vectores de valores, sino a tensores con dimensionalidad superior. De forma conceptual, estamos realizando el mismo razonamiento que con vectores, con la salvedad de que los índices están ordenados en mallas tensoriales. Aun así, podemos reducir el caso tensorial al vectorial si reordenamos los índices en un vector, donde el cálculo de las derivadas parciales se realiza como en el caso vectorial. Ahora los índices que denotan a las derivadas parciales son tuplas de enteros $i = (i_1, i_2, \dots, i_k)$, siendo k la dimensionalidad del vector cuyos índices estemos considerando.

Teorema 7.3 Sean \mathbf{X}, \mathbf{Y} tensores de manera que $\mathbf{Y} = f(\mathbf{X})$ y tenemos $z = g(\mathbf{Y})$. Entonces se verifica

$$\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} \mathbf{Y}_j) \frac{\partial z}{\partial \mathbf{Y}_j}$$



Figura 7.2: Grafo que representa el cálculo de la función $f(f(f(w)))$. Con este ejemplo se puede poner de manifiesto el problema de la repetición de operaciones en el cálculo del gradiente. Imagen obtenida de [11]

7.2.1. Algoritmo *backpropagation*

El algoritmo *backpropagation* consistirá en utilizar la regla de la cadena para calcular el gradiente en cada nodo del grafo computacional de manera eficiente. La eficiencia la logrará por medio de una reordenación de los cálculos a realizar y por la no repetición de operaciones ya hechas.

Para empezar, consideremos un primer algoritmo que compute el gradiente de una función g que calcula un único escalar en base a distintos valores de entrada o nodos del grafo. Las sucesivas derivadas las querremos calcular con respecto a cada uno de los valores de entrada. Llamemos u^n al escalar que calcula nuestra función y llamemos $u^i, i \in \{1, 2, \dots, n_g\}$ a cada uno de los valores de entrada de la función. Expresado analíticamente, $u^n = g(u^1, u^2, \dots, u^{n_f})$. Supongamos que el grafo computacional es el de la figura 7.2. Cada nodo es una variable de entrada u^i , es decir, $u^1 = w, u^2 = x = f(w), u^3 = y = f(x)$, y $z = f(y)$ es la variable de salida.

Para calcular la derivada parcial de z respecto de w y aplicando la regla de la cadena, tenemos que :

$$\begin{aligned} \frac{\partial z}{\partial w} &= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} = \\ &= f'(y) f'(x) f'(w) = f'(f(f(w))) f'(f(w)) f'(w) \end{aligned}$$

donde podemos ver que se calcula dos veces la expresión $f(x)$. En grafos más complejos, esta repetición puede ralentizar nuestro algoritmo y hacerlo muy ineficiente, así que para hacer más eficiente este cómputo, podemos considerar guardar el cálculo de esta operaciones y utilizarlo para futuros cálculos. Dependiendo de la prioridad que tenga el tiempo de cómputo y la memoria que tengamos, podremos utilizar una u otra expresión.

A partir de ahora, denotaremos por \mathbb{A}^i a todos los nodos que intervienen en el cálculo de la expresión u^i , llamados nodos padre de u^i . Denotaremos por $Pa(u^i) = \{j : u^j \in \mathbb{A}^i\}$ el conjunto de los índices de los nodos padre de u^i . De esta forma, el cálculo de u^n se puede expresar como $u^n = g(\mathbb{A}^n)$.

Sin pérdida de generalidad, suponemos que los términos de $Pa(u^i)$ están ordenados por orden de aparición en el grafo de cómputo al aplicar la función que calcula la salida u^i . Como antes, podemos escribir la expresión como $u^i = f^i(\mathbb{A}^i)$.

Para el cálculo de las distintas diferenciales, utilizaremos la regla de la cadena:

$$\frac{\partial u^l}{\partial u^j} = \sum_{i:j \in Pa(u^i)} \frac{\partial u^l}{\partial u^i} \frac{\partial u^i}{\partial u^j}$$

Si llamamos \mathcal{G} al grafo de cómputo de la función f , podemos crear un grafo \mathcal{B} con la misma configuración de \mathcal{G} pero añadiendo algunos nodos. Por cada derivada parcial de una variable u^i respecto a cualquiera de sus padre u^j crearemos un nodo que computará tal diferencial. Esto se realizará con el producto vectorial de la regla de la cadena. Nótese que cada una de las expresiones $\frac{\partial u^i}{\partial u^j}$ no introducirán cálculos recursivos si hemos guardado los datos en una estructura de datos apropiada.

El algoritmo de *backpropagation* visita una única vez los nodos necesarios para el cálculo de las sucesivas derivadas, cosa que no ocurría si utilizásemos la expresión recursiva. Si nuestra prioridad en memoria fuese superior a la prioridad en tiempo, podríamos reducir el número de operaciones que guardamos en memoria y calcular éstas con la expresión recursiva.

En 7.3 se ofrece un grafo ampliado para el cálculo del gradiente de 7.2. En él se pueden observar los nodos derivados de cada variable.

El algoritmo en pseudocódigo se encuentra en el algoritmo 1

Data: Ejecución del grado computacional para obtener la salida u^n

Result: Tabla `grad_table` en la que guardaremos las diferenciales de u^n respecto a las variables u^i

```

grad_table[u^n] ← 1
for j = n-i hasta j = 1 do
    Para el siguiente cálculo, se utilizarán los valores guardados de
     $\frac{\partial u^i}{\partial u^j}$ 
    grad_table[u^j] ←  $\sum_{i:j \in Pa(u^i)} grad\_table[u^i] \frac{\partial u^i}{\partial u^j}$ 
end

```

Algorithm 1: Algoritmo Back propagation simplificado para cálculo de funciones genéricas.

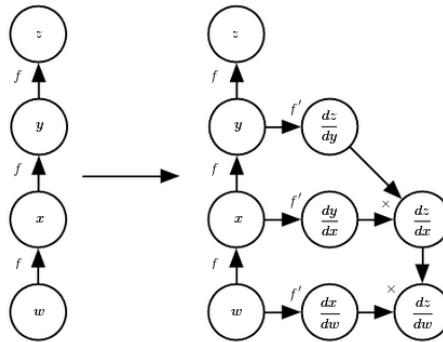


Figura 7.3: Grafo ampliado para el cálculo del gradiente del grafo 7.2. Imagen obtenida de [11]

7.2.2. Back-propagation en redes neuronales completamente conectadas

En este apartado veremos las particularidades del algoritmo **Back-propagation** para una red neuronal completamente conectada MLP(Multi Layer Perceptron). Recordemos que la variable de salida u^n de nuestro algoritmo será la función de pérdida $L(\hat{y}, y)$ para un solo ejemplo (x, y) , donde \hat{y} es la salida de la red neuronal con entrada x .

Para las capas ocultas, la red neuronal calcula su salida como $h^i = f^i(W^i h^{i-1} + b^i)$, siendo f^i la función de activación, W^i la matriz de pesos y b^i el vector de sesgos de la capa i . El término h^{i-1} representa la salida de la capa anterior.

El algoritmo 2 muestra el cálculo del gradiente con el algoritmo *backpropagation* pero ajustado a la estructura de red neuronal. Hemos introducido una función de regularización que explicaremos más adelante.

7.2.3. Derivadas símbolo a símbolo

Las expresiones algebraicas y los grafos que estamos considerando utilizan símbolos en lugar de valores numéricos, aunque luego para computar los valores reales, los sustituiremos. Estamos utilizando **representación simbólica**.

A la hora de diferenciar, simplemente sustituiremos los valores simbólicos asociados al input por valores numéricos y obtendremos unos valores numéricos describiendo el gradiente asociado. Llamamos a esto **diferenciación símbolo-a-numero**. Bibliotecas como **Torch** y **Caffe** utilizan este punto de vista.

Por otro lado, podríamos almacenar en cada nodo la expresión obtenida

Data: l profundidad de la red
Data: W^i matriz de pesos de la capa i -ésima
Data: b^i vector de sesgos de la capa i -ésima
Data: f^i función de activación de la capa i -ésima
Data: x input de la red
Data: y el valor de la función objetivo
Data: L función de coste
Data: Ω función de regularización

Una vez realizado los cálculos para calcular la función de coste:

$$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y)$$

for $k=l, l-1, \dots, 1$ **do**

$g \leftarrow \nabla_{a^k} J = g \odot f'(a^k)$	Cálculo de los gradientes referidos a este nivel de la red
	neuronal(W^i, b^i):
$\nabla_{b^k} J = g + \lambda \nabla_{b^k} \Omega(\theta)$	
$\nabla_{W^k} J = gh^{k-1} + \lambda \nabla_{W^k} \Omega(\theta)$	
	Gradiente que utilizará la capa anterior
$g \leftarrow \nabla_{h^{k-1}} J = W^k g$	

end

Algorithm 2: Algoritmo *backpropagation* para una estructura de red neuronal prealimentada (MLP)

para luego, sólo al final del proceso, completar el cálculo pertinente. En este caso, estaremos hablando de **diferenciación símbolo-a-símbolo** y es el enfoque que tienen **Theano** y **Tensor-flow**.

Por la complejidad que tienen intrínsecamente los enfoques **símbolo-a-símbolo**, muchos software utilizan el primer enfoque. Sin embargo, éste tiene la ventaja de que el grafo de cálculo se puede reorganizar para mayor eficiencia, pudiendo realizar los cálculos de cada nodo en cuanto los nodos padre estén calculados.

7.2.4. Algoritmo Back-propagation genérico.

El algoritmo de *backpropagation* es más sencillo de entender que de formalizar. Para computar el gradiente de un escalar z con respecto a alguno de sus nodos padre comenzamos obteniendo la derivada de z respecto a si mismo, 1. Podemos ahora obtener el gradiente de cada uno de los nodos padre de z multiplicando por el jacobiano de la operación que da lugar a z . Podemos continuar esta operación hasta que encontremos el nodo padre deseado, x . En caso de que algún nodo sea recorrido varias veces para el cálculo de varios gradientes simplemente sumamos los gradientes.

De manera formal, cada nodo en el grafo \mathcal{G} representa una variable,

que para hacerla lo más general posible, será un tensor V . Además, para representar cada una de las conexiones con las diferentes componentes del grafo, tendrá asociadas las siguientes operaciones:

- $\text{get_operation}(V)$. Esta subrutina nos devuelve la función que da lugar a nuestra variable.
- $\text{get_consumers}(V, \mathcal{G})$. Esta subrutina nos devuelve los nodos hijos de V en el grafo computacional \mathcal{G} .
- $\text{get_inputs}(V, \mathcal{G})$. Devuelve los nodos padre del nodo V en el grafo computacional \mathcal{G} .
- Además, para cada operación obtenida por get_operation denotada por op , debemos poder construir(u obtener) otra función $bprop$ que será necesaria para el cálculo del gradiente. Esta operación no necesita conocer ninguna regla de derivación, simplemente computar la siguiente expresión:

$$bprop(inputs, X, G) = \sum_i (\nabla_X op.f(inputs)_i) G_i$$

donde $inputs$ serán las variables de entrada de f , X será el tensor del cuál queremos calcular el gradiente y G es el gradiente de la salida.

A continuación se describe en 3 el algoritmo genérico.

Data: \mathbb{T} el conjunto de variables sobre las cuales queremos saber su gradiente

Data: \mathcal{G} el grafo computacional

Data: z la variable a diferenciar

Construimos un grafo \mathcal{G}' un subgrafo de \mathcal{G} que contiene a todos los nodos que son ancestros de z e hijos de algún nodo de \mathbb{T} .

```

grad_table[z] ← 1
for  $V \in \mathbb{T}$  do
    | build_grad(V,  $\mathcal{G}$ , grad_table)
end

```

return La tabla de gradientes $grad.table$ restringido a \mathbb{T}

Algorithm 3: Algoritmo genérico de Back Propagation. La mayoría del cálculo se hace en la subrutina 4

Input: V variable cuyo gradiente añadiremos a \mathcal{G} y $grad_table$.

Input: \mathcal{G} grafo a modificar

Input: \mathcal{G}' restricción del grafo a los que intervienen en el cálculo del gradiente

Input: $grad_table$ estructura de datos donde se guardan los gradientes de cada nodo

```

if  $V \in grad\_table$  then
|   return  $grad\_table$ 
end
 $i \leftarrow 1$ 
for  $hijo$  in  $get\_consumers(V, \mathcal{G}')$  do
|    $op \leftarrow get\_operation(hijo)$ 
|    $D \leftarrow build\_grad(C, \mathcal{G}, \mathcal{G}', grad\_table)$ 
|    $G^i = op.bprop(get\_inputs(C, \mathcal{G}'), V, D)$ 
|    $i \leftarrow i + 1$ 
end
 $G \leftarrow \sum_i G^i$ 
 $grad\_table][V] = G$ 
Insertamos  $G$  en como nodo y con sus consiguientes operaciones en el grafo  $\mathcal{G}$ 
return  $G$ 
```

Algorithm 4: Subrutina que es llamada por el algoritmo Back-propagation en 3. Calcula el gradiente de cada uno de los nodos de V de forma recursiva. Se utiliza programación dinámica.

Parte II

Aproximación Informática

Capítulo 8

Caso de estudio: Clasificación de corales. Base de datos *structureRSMAS*

Los corales son seres vivos muy importantes para los ecosistemas marinos. El reconocimiento automático de las distintas especies de corales basado en imágenes subacuáticas puede ayudar a expertos a detectar rápidamente especies de corales vulnerables o en peligro. Para ello, en la actualidad se cuenta con una gran cantidad de bases de datos con imágenes subacuáticas de distintas especies de corales. Sin embargo, la mayoría de ellas cuenta con imágenes tomadas por expertos a zonas muy específicas de los corales, las cuales se parecen muy poco a las que un vehículo subacuático autónomo podría tomar. La base de datos que se presenta en esta sección utiliza las imágenes proporcionadas por estos vehículos. También se propone un método de preprocesado para la base de datos llamado *Data Augmentation*. Con esta estrategia, se puede aumentar la cantidad de datos de forma artificial. Como medida del error de distintos modelos se ha propuesto la validación cruzada, un método que utiliza todos los datos para entrenar y para testear, aprovechando toda la información disponible. Por último, se estudia el estado del arte del problema.

8.1. Clasificación de corales

Para el problema de clasificación de especies de corales, se cuenta con muchas bases de datos en la actualidad, como [12]. Esta y otras bases de datos cuentan con imágenes de distintos corales clasificadas por la especie a la que pertenece. La obtención de estas imágenes las han realizado expertos especialistas que saben qué partes son las más características de cada coral.

Clases en RSMAS	Nº de imágenes en esta clase
Acropora Cervicornis (ACER)	44
Acropora Palmata (APAL)	41
Colpophyllia Natans (CNAT)	34
Diadema Antillarum (DANT)	20
Diploria Strigosa (DSTR)	16
Gorgonians (GORG)	18
Millepora Alcicornis (MALC).	33
Montastraea Cavernosa (MCAV).	38
Meandrina Meandrites (MMEA)	30
Montipora spp. (MONT)	21
Palythoas Palythoa (PALY)	32
Sponge Fungus (SPO)	23
Siderastrea Siderea (SSID)	36
Tunicates (TUNI)	23

Tabla 8.1: Descomposición por clases de RSMAS

Sin embargo, las imágenes obtenidas por medio de los vehículos subacuáticos automáticos estarán lejos de ser como estas imágenes tomadas por especialistas.

Con estas premisas, se crea *structureRSMAS*, que pretende ser una base de datos con imágenes más realistas. Se han combinado imágenes de distintas páginas web científicas, como [13],[14] o [15]. Las imágenes fueron tomadas por distintas cámaras, distintas resoluciones y tienen distintos tamaños, lo que las hace más generales. Por contra, al ser imágenes tomadas bajo el agua, tendrán peor calidad.

La base de datos *structureRSMAS*, disponible en [16], es una base de datos formada por 409 fotografías RGB de distintas especies de corales, cuya descomposición en las 14 especies es la que se describe en la tabla 8.1. En la imagen 8.1 podemos ver un ejemplo de cada clase.

Nuestra labor será, por tanto, encontrar una modelo de aprendizaje al problema de clasificación multi clase.

8.2. Data Augmentation

El **Data Augmentation**(DA) es una técnica de preprocesado de datos utilizada para aumentar el volumen de información que tenemos disponible. Esto se logrará generando nuevos datos válidos basados en los que ya conocemos.



Figura 8.1: Ejemplo de cada una de las especies de corales de la tabla 8.1, dispuestas en orden.

En nuestro caso, crearemos nuevas instancias de imágenes a través de distintas transformaciones afines. Algunos ejemplos de estas transformaciones son:

- **Rotaciones.** Se aplica un giro de ángulo θ con centro el centro de la imagen. La codificación de esta transformación se escribiría de la siguiente manera:

$$\psi(i, j) = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} i - \alpha \\ j - \beta \end{pmatrix} + \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

donde (i, j) son las coordenadas de cada pixel de la imagen.

- **Traslaciones.** Se mueve un número de píxeles concreto toda la imagen. Se define como:

$$\psi(i, j) = \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} n \\ m \end{pmatrix}$$

donde $n, m \in \mathbb{Z}$

- **Simetrias axiales.** Dada una recta $r : Ax + By + C = 0$, se considera la simetría axial respecto a r de cada punto (i, j) como:

$$\psi(i, j) = \begin{pmatrix} i \\ j \end{pmatrix} - 2R \begin{pmatrix} A \\ B \end{pmatrix}$$

donde R es la distancia del punto a la recta con signo, es decir, $R = \frac{Ai+Bj+C}{A^2+B^2}$.

Cabe destacar que puede que algunos puntos de la imagen transformada no tenga valor definido. Al trasladar una imagen en la dirección positiva del eje x, algunos puntos con coordenadas negativas pueden no tener imagen. Hay distintas técnicas para llenar estos vacíos de información, como asignarle el valor más cercano o asignar un valor fijo a todos los píxeles indefinidos.

Hay muchas técnicas para aumentar los datos ya obtenidas y no todas se adecuan a cualquier tipo de aprendizaje. Al añadir nuevos datos podemos estar incluyendo datos erroneos o inservibles. Por ejemplo, si queremos detectar el dígito que tenemos en una imagen, aplicar un giro de 180 grados para obtener nuevos datos nos haría confundir un 6 con un 9 y viceversa.

8.3. Medida del error. Validación cruzada($K-fcv$).

La validación cruzada es una técnica utilizada para medir una aproximación del error de un modelo. Al considerar varios modelos sobre un mismo problema, es necesario plantear una manera meticulosa para compararlos.

Una estrategia válida es la separación del conjunto de datos en dos conjuntos bien definidos, uno para entrenar nuestro modelo(train) y otro para comprobar su validez y capacidad de generalización(test). El error vendrá dado por el porcentaje de errores que se cometan en el conjunto de test.

Este método no es del todo fiable, ya que a la hora de configurar los hiper parámetros de nuestro modelo, podemos estar incurriendo en sobre aprendizaje sobre el conjunto de test, ya que nos fijaremos en este porcentaje para escoger entre varios hiper parámetros.

Una solución a esto es crear también un conjunto de validación, separado del conjunto de test y de train, con el cuál ajustaremos hiper parámetros, para luego sólo medir el porcentaje de acierto en el conjunto de test.

Todas estas estrategias tienen un déficit importante: no aprovechan toda la información para todo el estudio. Cada uno de los conjuntos considerados son válidos para cada una de las tareas, ya sea probar, entrenar o validar el modelo, y solo se utiliza para una de ellas. La estrategia de **validación cruzada** si lo hace.

Esta técnica separa el conjunto de datos en k subconjuntos disjuntos. Para cada subconjunto $X_i \ i \in \{1, \dots, k\}$ se plantea entrenar un modelo distinto. El conjunto sobre el que se entrenará será el conjunto completo

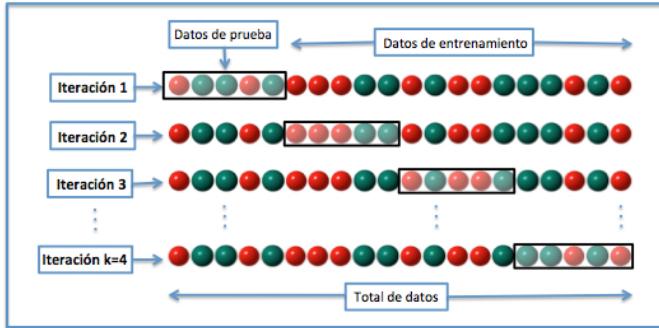


Figura 8.2: Imagen que expone el funcionamiento de la validación cruzada.
Imagen obtenida de [17]

pero sustrayendo X_i . Se calcula el porcentaje de acierto de cada modelo sobre X_i y se considera la media de estos porcentajes de acierto. En 8.2 se expone gráficamente su funcionamiento.

Con esta estrategia, todos los datos son utilizados para entrenar y testear el modelo, siendo accesible mucha más información. Como último apunte de este método, comprobar que la composición de cada una de las partes en las que dividimos la base de datos tiene una distribución similar ayuda a que esta estrategia sea más fiable.

Este método es el que se utiliza para evaluar la bondad de los modelos propuestos en este trabajo, en concreto, la validación cruzada con 5 subconjuntos(5-*fcv*).

8.4. Clasificación de corales: Estado de arte

En este trabajo, partimos de la investigación realizada por Anabel Gomez Rios[1] en el que se utiliza un clasificador a dos niveles basado en redes neuronales convolucionales para la clasificación de especies de corales, con una base de datos que une las ya nombradas *RSMAS* y *structureRSMAS*. El clasificador a dos niveles se basa en distinguir entre clases de una u otra base de datos para luego clasificarlas por separado. Nuestro objetivo en esta sección es estudiar el trabajo realizado para la base de datos *structureRSMAS*.

En el estudio de reconocimiento de patrones en imágenes, los modelos que mejor resultado obtienen son las redes neuronales convolucionales. Por ello, también se utilizan éstas para el desarrollo de un modelo competente para la base de datos considerada.

Puesto que la base de datos es tan pequeña en [16], se ha utilizado un método llamado *transfer learning*, en el cuál se aprovechan las confi-



Figura 8.3: Ejemplo del modelo Base propuesto en [1].

guraciones de otras redes neuronales entrenadas para problemas de mayor envergadura. Si se acomodan a un nuevo problema, se puede lograr la transferencia de conocimiento. Utilizando este tipo de clasificadores, se ha obtenido un porcentaje de acierto de hasta un 83,158 % para el método 5-*fcv* de evaluación del error.

También se utilizan técnicas DA. La mejor configuración que se ha considerado es la que aplica sólo una reflexión axial respecto a la recta vertical. Con esta transformación, se ha conseguido aumentar el porcentaje de acierto a un 84,211 %, alcanzando el estado de arte actual.

El modelo propuesto por Anabel Gomez[1] para el problema de clasificación de especies de corales tiene la siguiente configuración que aprovecha el conocimiento de las redes pre-entrenadas gracias al *tranfer learning*:

- Entrada de una imagen RGB para el modelo.
- Se introduce la imagen al modelo pre entrenado. Se elimina la capa de salida de 1000 neuronas y se utiliza la capa anterior de 2048 neuronas como entrada para las nuevas capas. Las capas anteriores a estas no serán entrenadas.
- Se le añade una capa de 512 neuronas con función de activación ReLu.
- Se acopla una última capa softmax de salida con 14 neuronas asociadas a las 14 especies de corales a clasificar.

A este modelo lo llamaremos **Modelo Base(MB)** a partir de ahora. La imagen 8.3 muestra gráficamente la configuración del modelo.

En la siguiente tabla se detallan los resultados obtenidos por los modelos base con las redes pre-entrenadas *Resnet50*(MB-Res) e *InceptionV3*(MB-Inc) sin considerar 8.2 y considerando *Data Augmentation*(prefijo DA-) 8.3. Para analizar la bondad de este método se ha creado una estructura de 5-*fcv*.

Set	MB-Res test	MB-Res train	MB-Inc test	MB-Inc train
1	0.8684	1.0	0.8552	1.0
2	0.8947	1.0	0.8289	1.0
3	0.7894	1.0	0.8552	1.0
4	0.8552	1.0	0.8157	1.0
5	0.8421	1.0	0.7631	1.0
5-fcv	0.8500	1.0	0.8236	1.0

Tabla 8.2: Estudio 5-fold cross validation de MB

Set	DA-MB-Res test	DA-MB-Res train	DA-MB-Inc test	DA-MB-Inc train
1	0.9210	1.0	0.8421	1.0
2	0.8552	1.0	0.8157	1.0
3	0.8026	1.0	0.8552	1.0
4	0.8421	1.0	0.8289	1.0
5	0.8552	1.0	0.8026	1.0
5-fcv	0.8552	1.0	0.8289	1.0

Tabla 8.3: Estudio 5-fold cross validation de DA-MB

Capítulo 9

Propuestas de modelos

Una vez estudiado cada uno de los aspectos teóricos que queremos utilizar, vamos a proponer un modelo práctico para la clasificación de imágenes de la base de datos *structureRSMAS*. Primero, veremos que la creación de vistas artificiales es factible para las imágenes del problema. Para ello, utilizaremos las redes pre-entrenadas para imagenet *Resnet50* e *InceptionV3* haciendo uso del conocimiento que podemos aprovechar de ellas gracias al *Transfer Learning*. Tras esto, utilizaremos el algoritmo *MVMED* para crear un buen modelo basado en las dos vistas creadas.

9.1. Creación de las vistas artificiales. Núcleos artificiales de aprendizaje basados en Transfer Learning.

Como ya comentamos en la sección 4.3, podemos crear vistas artificiales basándonos en buenas funciones de núcleo. Consideraremos como funciones de núcleo a las funciones que a cada imagen le aplican cada una de las redes pre-entrenadas hasta la penúltima capa, transformando una imagen RGB de 3x224x224 datos (o 3x299x299) en un vector de características de dimensión 2048, dependiendo de la red pre-entrenada utilizada. Es decir, para la red pre entrenada i -ésima, se considera la función de núcleo

$$\phi_i : \mathbb{R}^{N_i} \times \mathbb{R}^{N_i} \times \mathbb{R}^3 \rightarrow \mathbb{R}^{2048} , \phi_i(X) = f_i^{n-1}(X)$$

donde N_i es la dimensión de la imagen de entrada de la red i -ésima y donde f_i^j denota a la función de activación j -ésima de la i -ésima red pre entrenada. Cada instancia de la base de datos quedará definida como $X^* = (\phi_1(X), \dots, \phi_n(X))$, siendo X la imagen real de la base de datos.

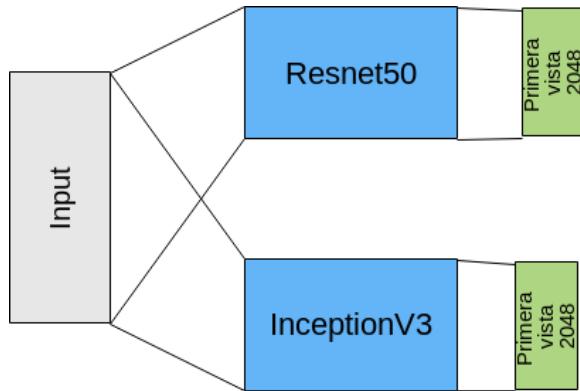


Figura 9.1: Creación de vistas artificiales basada en el conocimiento adquirido por cada una de las redes pre entrenadas *Resnet50* e *InceptionV3*

Por el estudio asociado al *Transfer Learning*, los subespacios de características creados y considerados núcleos deben ser representativos para nuestro problema. Puesto que los MB obtienen buenos resultados, se puede asumir que estos subespacios de características son buenos para la clasificación.

Para la creación de vistas artificiales hemos considerado dos de las redes neuronales pre entrenadas para imagenet. De esta forma, extraeremos dos conjuntos de 2048 características. En la figura 9.1 se muestra gráficamente cómo extraemos las vistas de forma artificial.

9.2. Propuesta Multiview para 2 vistas(MVL)

En el apartado 4.2.2 a los algoritmos de entrenamiento *Multiview*, estudiamos el algoritmo MVMED para 2 vistas. El objetivo del mismo era crear un buen clasificador entrenando por separado dos funciones discriminante que maximicen la entropía por separado. Esto lo traducimos a que cada vista es entrenable de forma aislada. Aun así, debemos entrenar cada vista con los mismos ejemplos de entrenamiento.

Hemos mantenido la configuración propuesta en el modelo MB para aprovechar cómputos, es decir:

- Cada vector de 2048 características es conectado completamente con una capa de 512 neuronas con función de activación ReLU. Cada una de estas capas será identificada por la vista de la que proviene($ReLU_i$).
- Acoplamos una capa de salida softmax a cada capa $ReLU_i$ por separado. Esta salida será característica a cada vista en concreto. A esta capa la llamaremos SV_i , donde i es el identificador de la vista.

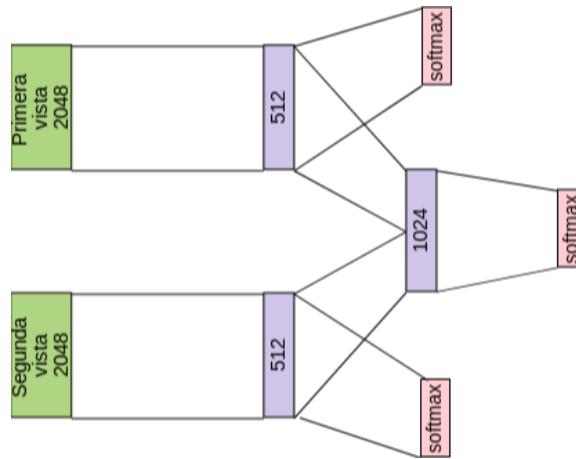


Figura 9.2: Modelo MVL propuesto.

- Concatenamos las anteriores capas ReLU de 512 características, dando como resultado una capa con 1024 características. Esta capa no aplica ninguna modificación a la entrada de la misma.
- Acoplamos una capa de salida softmax a la capa de concatenación de 1024 neuronas. Esta será la capa softmax final(SF).

Entrenaremos estos pesos con el algoritmo de optimización propuesto en[1], Stochastic Gradient Descent(SGD). Sin embargo, la etapa de entrenamiento será considerablemente más compleja:

- Primero, entrenaremos las capas SV_i y $ReLU_i$ como si no estuviesen conectadas. Esto entrenaría cada una de las funciones discriminante y así separar los subespacios de cada vista de forma óptima.
- Tras esto, fijamos estos nuevos subespacios y entrenamos la última capa SF .

A este nuevo modelo lo llamaremos a partir de ahora **modelo Multi-view Learning(MVL)**.

En la figura 9.2 se muestra gráficamente la configuración del modelo propuesto. Nótese que las capas softmax utilizadas para el entrenamiento de las capas ReLu SV_i no son utilizadas para la clasificación final, de esta tarea se encarga SF . Sin embargo, son cruciales para el entrenamiento de cada vista por separado. Además, sirven como clasificadores independientes.

En las siguientes tablas se detallan los resultados obtenidos en una estructura de 5-fcv considerando *Data Augmentation* 9.2 y sin considerar 9.1.

Set	MVL test	MVL train
1	0.9078	1.0
2	0.8947	1.0
3	0.8552	1.0
4	0.8684	1.0
5	0.8947	1.0
5 fcv	0.8842	1.0

Tabla 9.1: Estudio 5-fold cross validation de MVL

Set	DA-MVL test	DA-MVL train
1	0.9342	1.0
2	0.8947	1.0
3	0.8421	1.0
4	0.8815	1.0
5	0.8815	1.0
5 fcv	0.8868	1.0

Tabla 9.2: Estudio 5-fold cross validation de DA-MVL

Para hacer un estudio exhaustivo de la bondad del enfoque *Multiview*, hemos ideado distintos modelos con distintas estructuras y con diversas formas de entrenamiento. Para empezar, creamos un modelo que no tiene en cuenta la posible estructuración de la información, los modelos densos. Continuamos con la creación de un modelo que si estructura la información por separado pero no aprovecha la información asociada a cada distribución por separado. Por último, se construyen modelos que acoplan el conocimiento de otros modelos: Los modelos de ensamble *Stacking*.

9.3. Modelos Densos(MD)

Para empezar, consideraremos modelos que no tienen en cuenta la separación por vistas creadas artificialmente pero contando con la información de los núcleos de aprendizaje. Esto lo hacemos para considerar la primera aproximación del enfoque *Multiview*, en la que simplemente se concatenan las vistas obtenidas. Se utiliza la separación de vistas artificial considerada en la figura 9.1.

A continuación, se explica la configuración de los modelos densos:

- Concatenamos las dos vistas extraídas, obteniendo un vector de carac-

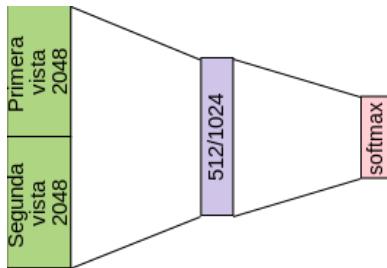


Figura 9.3: Modelo denso

terísticas de 4096 componentes.

- Conectamos a una capa de neuronas ReLU este vector. Esta capa la hemos considerado con 512 o con 1024 neuronas.
- Conectamos esta última capa a la salida softmax.

En la figura 9.3 se ofrece una descripción gráfica del modelo denso. Como vemos, esta configuración es una red neuronal completamente conectada. Será entrenada con el algoritmo SGD.

Dependiendo de el número de neuronas en la capa intermedia, nombraremos a los modelos concretos como *MD512* o *MD1024*. En las siguientes tablas se detallan los resultados obtenidos en una estructura de 5-*fcv* considerando *Data Augmentation* 9.3 y sin considerar 9.3.

Set	MD512 test	MD512 train	MD1024 test	MD1024 train
1	0.8815	1.0	0.9210	1.0
2	0.8684	1.0	0.8684	1.0
3	0.8157	1.0	0.8289	1.0
4	0.8684	1.0	0.8552	1.0
5	0.8815	1.0	0.8684	1.0
5-fcv	0.8631	1.0	0.8684	1.0

Tabla 9.3: Estudio 5-fold cross validation de MD

La elección de estos dos modelos tiene su justificación. Puesto que queremos compararlos con nuestro modelo MVL, tendremos que conseguir modelos con un número parecido de pesos a optimizar. Ya que el modelo optimiza por un lado 512 neuronas por vista, la consideración del primer modelo MD512 queda justificada.

Por otro lado, el modelo MVL completo cuenta con 1024 neuronas entrenadas en la penúltima capa, por lo que el segundo modelo MD1024 también tiene interés en la comparativa.

Set	DA-MD512 test	DA-MD512 train	DA-MD1024 test	DA-MD1024 train
1	0.9210	1.0	0.9210	1.0
2	0.8684	1.0	0.8684	1.0
3	0.8289	1.0	0.8157	1.0
4	0.8815	1.0	0.8815	1.0
5	0.8815	1.0	0.8815	1.0
5-fcv	0.8763	1.0	0.8736	1.0

Tabla 9.4: Estudio 5-fold cross validation de DA-MD

Se podría construir muchos mas modelos MD en esta linea pero la configuración óptima de estas redes neuronales queda fuera de los objetivos de este trabajo.

9.4. Modelo One Stage Multiview(1-STA-MVL)

El modelo *One Stage Multiview(1-STA-MVL)* considerado en esta sección enfoca la comparativa con el modelo MVL a la etapa de aprendizaje. Se parte de las mismas dos vistas extraídas en 9.1. Se detalla la estructura del modelo a continuación.

- Cada vista se conecta a una capa ReLU de 512 neuronas.
- Concatenamos ambas capas, obteniendo así una capa de 1024 neuronas. Esta capa no aplica ningún cambio a las entradas del modelo.
- Finalmente, se acopla una capa softmax de salida de 14 neuronas.

Nótese que la estructura del modelo 1-STA-MVL es casi idéntica al modelo MVL. La diferencia radica en el considerar el algoritmo de entrenamiento MED en lugar de MVMED. El algoritmo MED sólo aprende una distribución de probabilidades, que será la de el estadístico $(\phi_1(X), \phi_2(X))$. El nombre de *one stage* ha sido inspirado porque sólo tiene una etapa de entrenamiento.

En la figura 9.4 se ofrece una descripción gráfica del modelo. El modelo 1-STA-MVL se optimizará utilizando el algoritmo SGD.

En las siguientes tablas se detallan los resultados obtenidos en una estructura de 5-fcv considerando *Data Augmentation* 9.6 y sin considerar 9.5.

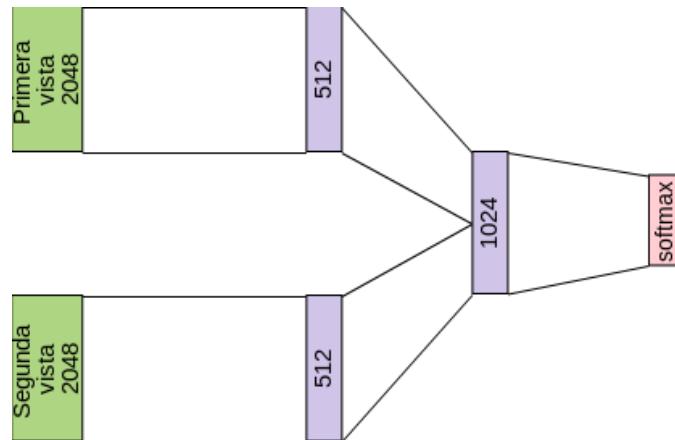


Figura 9.4: Modelo 1-STA-MVL

Set	1-STA-MVL test	1-STA-MVL train
1	0.9210	1.0
2	0.8552	1.0
3	0.8157	1.0
4	0.8684	1.0
5	0.8947	1.0
5-fcv	0.8710	1.0

Tabla 9.5: Estudio 5-fold cross validation de 1-STA-MVL

Set	DA-1-STA-MVL test	DA-1-STA-MVL train
1	0.9342	1.0
2	0.8421	1.0
3	0.8289	1.0
4	0.8684	1.0
5	0.8552	1.0
5-fcv	0.8657	1.0

Tabla 9.6: Estudio 5-fold cross validation de DA-1-STA-MVL

9.5. Modelos de Ensemble:Stacking(ST)

Estos modelos van a ser los más complejos con los que vamos a comparar el modelo MVL. Los modelos Ensembles pretenden mejorar la información obtenida por varios modelos. El principio con el que se trabaja se basa en que distintos modelos tendrán errores distintos, pudiendo suplir estos errores si se trabaja en conjunto.

Los modelos Stacking[18] son un tipo de ensamblaje de modelos que concatenan las salidas o predicciones de varios modelos. Estas predicciones se utilizan como entrada para un nuevo modelo de segundo nivel el cual podremos entrenar para obtener una predicción final.

9.5.1. Stacking Res-Inc

El primer modelo Stacking que vamos a considerar en esta sección acopla los modelos MRRes y MRInc. A continuación, pasamos a describir más detalladamente el modelo Stacking utilizado:

- Como dato de entrada, se utilizan las imágenes RGB de la base de datos.
- Se consideran los modelos MRRes y MRInc como dos modelos válidos y obtenemos las predicciones que cada modelo realiza. Cada predicción es un vector de probabilidades de 14 componentes.
- Estas predicciones son tratadas como características para la siguiente capa y son concatenadas, dando una capa de 28 componentes.
- Esta capa la utilizamos como entrada para una capa softmax nueva de otras 14 neuronas.

A este modelo lo llamaremos ST-Res-Inc porque es un modelo Stacking que acopla los modelos MRRes y MRInc. En la figura 9.5 se muestra gráficamente la composición del modelo Stacking.

En las siguientes tablas se detallan los resultados obtenidos en una estructura de 5-*fcv* considerando *Data Augmentation* 9.8 y sin considerar 9.7.

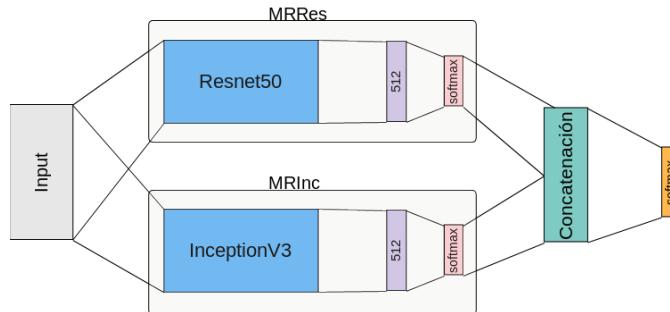


Figura 9.5: Modelo Stacking considerando los dos modelos de referencia MRRes y MRInc(ST-Res-Inc).

Set	ST-Res-Inc test
1	0.9078
2	0.8815
3	0.8552
4	0.8289
5	0.8289
5-fc	0.8605

Tabla 9.7: Estudio 5-fold cross validation de ST-Res-Inc

Set	DA-ST-Res-Inc test
1	0.9078
2	0.8815
3	0.8421
4	0.8947
5	0.8421
5-fc	0.8736

Tabla 9.8: Estudio 5-fold cross validation de DA-ST-Res-Inc

9.5.2. Stacking Res-Inc-MVL

El modelo Stacking que se propone en esta sección acopla los modelos MRRes, MRInc y MVL. A continuación se detalla su configuración:

- Como dato de entrada, se utilizan las imágenes RGB de la base de datos.
- Se consideran los modelos MRRes y MRInc. Obtenemos las predicciones que cada modelo realiza. Cada predicción es un vector de probabilidades de 14 componentes.
- Se considera el modelo MVL. Para el modelo MVL, sepáramos las vistas necesarias de la imagen como en 9.1. Obtenemos la predicción del modelo MVL, que es un vector de probabilidades de 14 componentes.
- Las predicciones de los modelos MRRes, MRInc y MVL son tratadas como características para la siguiente capa y son concatenadas, dando una capa de 42 componentes.
- Esta capa la utilizamos como entrada para una capa softmax nueva de otras 14 neuronas.

A este modelo lo llamaremos ST-Res-Inc-MVL porque es un modelo Stacking que acopla los modelos MRRes, MRInc y MVL. La última capa de este modelo será entrenada con el algoritmo de optimización SGD. En las siguientes tablas se detallan los resultados obtenidos en una estructura de 5-fcv considerando *Data Augmentation* 9.10 y sin considerar 9.9.

Set	ST-Res-Inc-MVL test	ST-Res-Inc-MVL train
1	0.9210	1.0
2	0.8815	1.0
3	0.8552	1.0
4	0.8684	1.0
5	0.8421	1.0
5-fcv	0.8736	1.0

Tabla 9.9: Estudio 5-fold cross validation de ST-Res-Inc-MVL

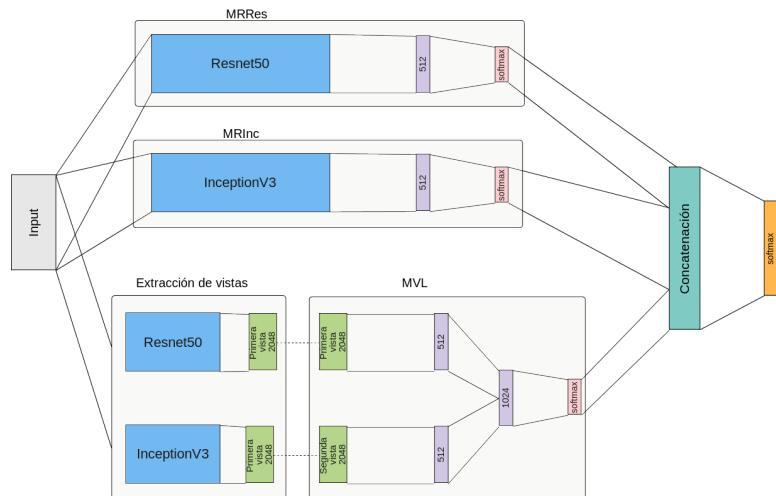


Figura 9.6: Modelo Stacking considerando los dos modelos de referencia MRRRes y MRInv y el modelo MLV

Set	DA-ST-Res-Inc-MVL test	DA-ST-Res-Inc-MVL train
1	0.9210	1.0
2	0.8947	1.0
3	0.8552	1.0
4	0.8947	1.0
5	0.8684	1.0
5-fcv	0.8868	1.0

Tabla 9.10: Estudio 5-fold cross validation de DA-ST-Res-Inc-MVL

Capítulo 10

Análisis de resultados.

En esta sección vamos a analizar los resultados de los modelos expuestos en este trabajo. Consideraremos los cinco tipos de modelos que hemos analizado, los cuales son:

- Modelos Base(MB), en los cuales consideraremos dos tipos según la red pre-entrenada de la cual herede información, las cuales serán Resnet50(MBRes) e InceptionV3(MBInc).
- Modelo Multiview(MVL).
- Modelos Stacking(ST), dentro de los cuales consideraremos dos tipos, el que acopla los dos modelos MBRes y MBInc(ST-Res-Inc) y el que acopla también el modelo MVL(ST-Res-Inc-MVL).
- Modelos densos(MD), en los que hemos configurado dos modelos distintos dependiendo de el número de neuronas intermedias. Hemos considerado de 512(MD512) y 1024(MD1024).
- Modelo One Stage Multiview(1-STA-MVL).

Además para cada modelo hemos considerado la aplicación de *Data Augmentation*. A continuación se presentan los resultados del estudio 5-fcv de los distintos modelos expuestos10.1. En ellos se resalta en rojo los dos mejores resultados para la base de datos.

Pasamos al análisis de los datos obtenidos.

- El peor modelo es MBInc, con un 82,36 % de acierto sin DA y un 82,89 % con DA.
- El modelo MBRes tiene un porcentaje de acierto de un 85 % sin DA y un 85,52 % con DA. La diferencia con los resultados obtenidos en

Modelo	Medida de acierto 5-fcv sin DA	Medida de acierto 5-fcv con DA
MBRes	0.8500	0.8552
MBInc	0.8236	0.8289
MVL	0.8842	0.8868
ST-Res-Inc	0.8605	0.8736
ST-Res-Inc-MVL	0.8736	0.8868
MD-Res-Inc512	0.8631	0.8763
MD-Res-Inc1024	0.8684	0.8736
1-STA-MVL	0.8710	0.8657

Tabla 10.1: Tabla comparativa de los distintos modelos expuestos en el trabajo.

[1] pueden ser debidos a factores aleatorios como la semilla aleatoria escogida. Por la poca cantidad de imágenes en el conjunto de datos, la mejora de este modelo se traduce en reconocer 3 imágenes más.

- La utilización de DA aumenta el porcentaje de acierto de casi todos los modelos. Sin embargo, esta mejora no es de mucho más de un 1 %.
- El modelo MVL ha alcanzado un 88,42 % de acierto sin DA y un 88,68 % con DA, obteniendo la mejor puntuación de todos los modelos.
- El modelo ST-Res-Inc obtienen buenos resultados. Obtiene 86,05 % sin DA y un 87,36 % con DA, superando los modelos MB en los que se basa.
- El modelo ST-Res-Inc-MVL llega a alcanzar al modelo MVL con un 88,68 % con DA. Sin aplicar DA, obtiene un 87,36 % de acierto.
- Los modelos MD obtienen resultados muy parecidos, siendo estos un 86,31 % para MD512 y un 86,84 % para MD1024 sin DA. Considerando DA, obtienen un 87,63 % y un 87,36 % respectivamente.
- El modelo MVL-MED obtiene un buen porcentaje de acierto para el caso sin DA, con un 87,10 %. Sin embargo, es el único modelo que empeora con DA, con un 86,57 %.

Se ha conseguido un 88,68 % de acierto con el modelo *MVL*, lo que es una mejora sustancial al 84,211 % obtenido por el mejor modelo MB[1], alcanzando así el estado de arte de este problema.

10.1. Software utilizado

Para los experimentos se ha utilizado Python[19]. Python es un lenguaje de programación de alto nivel caracterizado, entre otras cualidades, por su flexibilidad y comprensión. Además, se ha popularizado últimamente y se han creado muchas librerías adaptadas para *Machine Learning*.

Una de estas librerías es *Keras*[20]. Esta librería tiene muchas facilidades para la creación y entrenamiento de redes neuronales de distinta índole, no solo las prealimentadas. También puede crear procedimientos dinámicos de *Data Augmentation* configurables, lo cuál es muy útil cuando se utilizan bases de datos muy grandes. Además, hace uso de la GPU para los cálculos paralelos, lo que hace mucho más rápido el cómputo. Esto es posible gracias a que usa *TensorFlow*.

TensorFlow[21] es un software libre para el cálculo numérico de grafos de computación. Como vimos en el apartado de cálculo numérico de grafos de computación, el objetivo de esta librería es reordenar dinámicamente los cálculos de los grafos para conseguir más eficiencia. Para estos cálculos se puede utilizar CPU o, como en nuestro caso, GPU.

Las redes preentrenadas a las que hacemos mención se han obtenido de keras, su apartado de aplicaciones [22]. En ella, podemos obtener información sobre cómo utilizar estas redes para configurar nuevos modelos gracias al *Transfer Learning*.

Para la ejecución de los programas para el análisis de resultados se ha utilizado el servidor web Titan de la ETSIIT.

Parte III

Conclusion

Capítulo 11

Conclusiones y vías futuras

Este trabajo tenía como principal objetivo hacer una introducción al aprendizaje automático. También se pretendía hacer una introducción al nuevo enfoque *Multiview* y la posibilidad de obtener conocimiento en base a éste. Para ello, se han utilizado técnicas de minimización como es el gradiente descendente. Por la necesidad de un cálculo rápido del gradiente, se presentó también el algoritmo de *backpropagation*.

En la parte práctica, se quería proponer un modelo basado en el enfoque *Multiview* competente para un problema de clasificación. Se han presentado distintos modelos comparables al modelo propuesto y se ha analizado la bondad del mismo respecto a los demás. También se ha aplicado técnicas de *Data Augmentation* para este análisis. Con todas estas circunstancias, el modelo propuesto ha mejorado todas las propuestas similares, consiguiendo así nuestro objetivo.

Hemos alcanzado los objetivos propuestos para este proyecto. Sin embargo para seguir trabajando en el estudio del enfoque *Multiview*, se proponen nuevas posibles mejoras.

Por un lado, hemos utilizado sólo dos vistas para nuestro modelo. Un buen camino para mejorar este modelo es utilizar más redes pre-entrenadas u otras técnicas de extracción de características para acoplarlas al modelo *MVL*. También hay que investigar el potencial que pueden tener los clasificadores *Multiview* para problemas en los que no siempre contemos con todas las vistas.

Por otro lado, la técnica de entrenamiento *Multiview*, en concreto la etapa de aprendizaje, deja abierta la posibilidad de entrenar capas de forma independiente. Esto puede generar un nuevo modelo de entrenamiento de redes neuronales que puede ser mejor que el actual.

También se considera la aplicación de este enfoque a problemas en los

que las vistas sean naturales y no creadas artificialmente. Problemas en los que las imágenes atiendan a distintas características de una misma clase pueden ser problemas abarcables por este tipo de aprendizaje.

Este proyecto ha sido desarrollado dentro de una beca de colaboración con el departamento de Ciencias de la Computación con el objetivo de continuar con el desarrollo del conocimiento de las redes neuronales y su interpretabilidad.

Bibliografía

- [1] Anabel Gómez-Ríos, Siham Tabik, Julián Luengo, Francisco Herrera, ASM Shihavuddin, and Bartosz Krawczyk. Redes neuronales convolucionales para una clasificación precisa de imágenes de corales. pages 1171–1176, 2019.
- [2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
- [3] Tom Mitchell, Bruce Buchanan, Gerald DeJong, Thomas Dietterich, Paul Rosenbloom, and Alex Waibel. Machine learning. *Annual review of computer science*, 4(1):417–433, 1990.
- [4] WIKIPEDIA. Hoeffding’s inequality. wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Hoeffding%27s_inequality. Última fecha de acceso:16/06/2019.
- [5] Shiliang Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, 23(7-8):2031–2038, 2013.
- [6] Shiliang Sun, Liang Mao, Ziang Dong, and Lidan Wu. *Multiview Machine Learning*. Springer, 2019.
- [7] WIKIPEDIA. Kullback-Leibler divergence. wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence. Última fecha de acceso:16/06/2019.
- [8] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.
- [9] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.
- [10] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. *A theoretical framework for back-propagation*, volume 1. 1988.

- [11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [12] Coral reef dataset, v2. shihavuddin, asm,. <https://data.mendeley.com/datasets/86y667257h/2>, 2017. Última fecha de acceso:16/06/2019.
- [13] Encyclopedia of life. <https://eol.org/>. Última fecha de acceso:16/06/2019.
- [14] Iucn red list of threatened species. <http://www.iucnredlist.org/>. Última fecha de acceso:16/06/2019.
- [15] Coralpedia of the univerisity of warwick. <http://coralpedia.bio.warwick.ac.uk/en>. Última fecha de acceso:16/06/2019.
- [16] Coral species identification with texture or structure images using a two-level classifier based on convolutional neural networks.
- [17] WIKIPEDIA. Validación cruzada. wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada. Última fecha de acceso:16/06/2019.
- [18] Stacking models for improved predictions. <https://www.kdnuggets.com/2017/02/stacking-models-imropved-predictions.html>. Última fecha de acceso:16/06/2019.
- [19] Python. <https://www.python.org/>. Última fecha de acceso:16/06/2019.
- [20] Keras. <https://keras.io/>. Última fecha de acceso:16/06/2019.
- [21] TensorFlow. <https://www.tensorflow.org/>. Última fecha de acceso:16/06/2019.
- [22] Keras Applications. <https://keras.io/applications/>. Última fecha de acceso:16/06/2019.

