Universidad Zaragoza
1542

Trabajo Fin de Grado

Reconstrucción 3D a partir de una imagen

Autor

Jesús Oliva Maza

Directores

Javier Civera Sancho
Luis Montesano del Campo

Escuela de Ingeniería y Arquitectura
2015

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Jesús Oliva Maza,

con nº de DNI 73130325P en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Reconstrucción 3D a partir de una imagen,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 5 de Septiembre de 2015

Fdo: Jesús Oliva Maza

A mi familia, que tanto apoyo me ha dado durante la elaboración de este trabajo

Reconstrucción 3D a partir de una imagen

Resumen

El objetivo de este Trabajo de Fin de Grado es la reconstrucción tridimensional de una escena a partir de una única vista. Este es uno de los retos claves en visión por computador, con potenciales aplicaciones en campos como robótica, modelado 3D, entendimiento de escenas e imágenes, etc.

Esta tarea se basa en el trabajo con "pistas" cuyos modelos matemáticos son incompletos y no paramétricos, como puntos y líneas de fuga, reconocimiento de objetos y escenas, etc. Es por ello que su resolución se basa en las técnicas de aprendizaje automático. En vez de plantear un modelo paramétrico, se suministran las imágenes de las escenas y sus correspondientes imágenes de profundidad y se extraen patrones de forma automática a través de un algoritmo. Recientemente, el algoritmo de las redes neuronales artificiales (RNA) de aprendizaje profundo está superando al estado del arte en varios retos de aprendizaje automático, por lo que ha sido el escogido para este trabajo.

Para la realización del trabajo, en primer lugar se ha realizado una revisión bibliográfica del estado del arte en este problema, en concreto en el uso de RNAs para su resolución. En segundo lugar, se han escogido imágenes de escenas con sus correspondientes profundidades, necesarias para el funcionamiento de la RNA, tras un estudio y comparación de las bases de datos (datasets) más empleadas por la comunidad científica. En tercer lugar, se ha hecho una revisión del software disponible y del hardware necesario para la implementación de una RNA, así como la instalación y familiarización con éste. Tras este trabajo previo para disponer de las herramientas con las que resolver el problema, se ha procedido al estudio de éste. Se han diseñado, implementado y analizado tres modelos de RNAs basados en una técnica en común aplicada de una forma innovadora, obteniendo resultados cercanos al estado del arte.

ÍNDICE

ANEXOS

1 INTRODUCCIÓN

El objetivo de la visión por computador es conseguir que un ordenador pueda "ver", es decir, que sea capaz de entender el contenido de las imágenes de manera automática. Algunos ejemplos de los problemas que suele tratar la visión por computador son los siguientes: reconstrucción tridimensional de una escena a partir de varias imágenes; detección, segmentación y localización de objetos, personas, caras; o reconocimiento de acciones en vídeo.

Uno de los aspectos más relevantes del entendimiento de una escena es su estructura 3D. Los algoritmos más usuales están basados en la reconstrucción a partir de varias imágenes (multivista). Dichos algoritmos se encuentran en una fase muy avanzada de desarrollo, siendo utilizados incluso en aplicaciones comerciales. Este Trabajo de Fin de Grado se centra en un problema mucho menos estudiado y mucho menos maduro desde un punto de vista investigador: la reconstrucción de una escena tridimensional a partir de una única imagen. Su relevancia radica en que hay situaciones en las cuales no se puede acceder a varias vistas de una escena. Por ejemplo, en robótica móvil, un robot que entra a una habitación necesita una reconstrucción inicial para navegar sin colisiones, y solo dispone de una imagen al entrar por la puerta. Como otro ejemplo, si se quisiera estimar la profundidad de una imagen descargada de internet, en muchas ocasiones no se dispone de otras imágenes de la misma escena.

La reconstrucción multivista se fundamenta en modelos matemáticos muy precisos, basados en correspondencias de puntos entre distintas imágenes. Con estos se computa la estructura tridimensional que mejor se ajusta a las imágenes, usando para ello relaciones geométricas (basadas en la triangulación de los puntos), modelos paramétricos y técnicas de optimización.

Sin embargo, en la reconstrucción a partir de una única imagen, estas técnicas no son aplicables ya que no se puede resolver por geometría. Para tratar este problema hay que trabajar con otras pistas cuyos modelos matemáticos son más ambiguos: líneas de fuga, perspectivas, reconocimiento de objetos, y con un sinfín de factores que nosotros mismos, aunque sea de forma inconsciente, tenemos en cuenta. Y es que las personas somos capaces de resolver este problema, aunque no sea con demasiada exactitud: cualquiera de nosotros puede intuir la profundidad de una escena a partir de una sola imagen o fotografía.

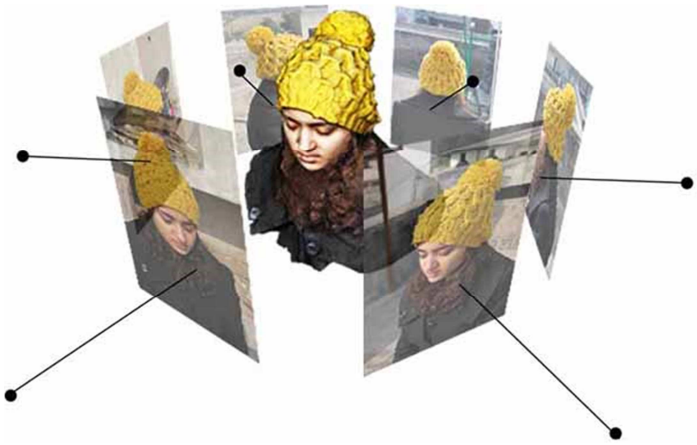


Figura 1.1: Reconstrucción 3D estéreo (multivista). [1]

Debido a este tratamiento de información de alto nivel y además de una manera intuitiva, la obtención de un modelo paramétrico del problema se vuelve extremadamente complicada. Es por ello que se recurre al aprendizaje automático para

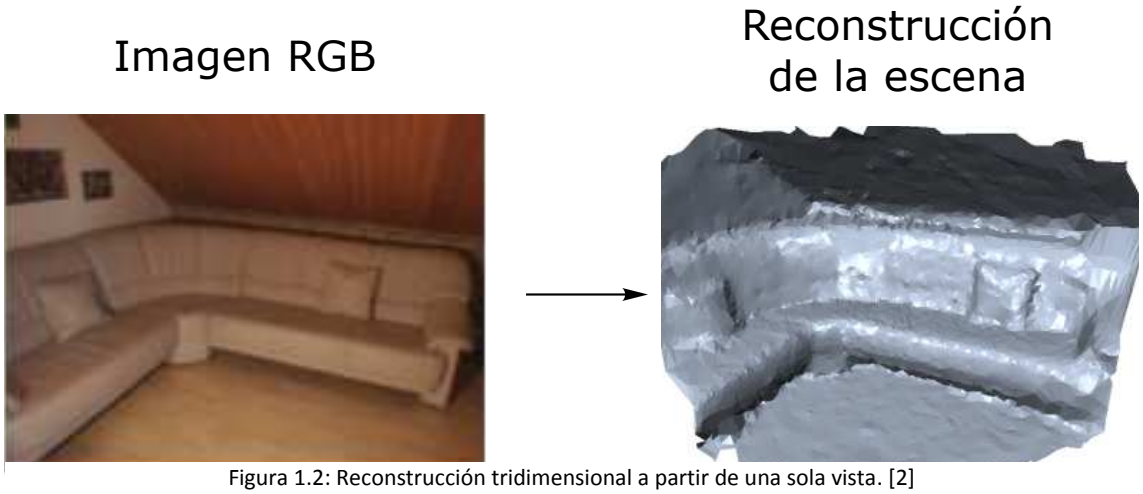


Figura 1.2: Reconstrucción tridimensional a partir de una sola vista. [2]

resolver este problema, una herramienta fundamental en visión por computador. Problemas como la detección de objetos o clasificación de escenas son resueltos a través de esta. El aprendizaje automático tiene como objetivo que los ordenadores sean capaces de "aprender" por sí mismos, siempre a partir de unos datos de entrenamiento (denominados dataset) que les sirven de ejemplo para establecer su propio modelo del problema, que puede llegar a ser muy complejo. Los algoritmos de aprendizaje automático trabajan con patrones aprendidos de los datos suministrados, que pueden ser formas, gradientes de color, o cualquier estructura que aparezca en las imágenes. Ante una nueva imagen, el algoritmo aplicará estos patrones aprendidos para predecir la profundidad de la escena, sin ser necesaria la formulación de un modelo para su resolución.

Los datos suministrados al algoritmo deben ser tales que se conozca la profundidad de todas las vistas. Con el desarrollo de las cámaras kinect, existe una gran cantidad de imágenes RGBD disponibles que pueden ser empleadas en el proceso de aprendizaje. Estas se pueden encontrar en los denominados datasets, que consisten en conjuntos de grandes cantidades de datos empleados por la comunidad de aprendizaje automático. Es de gran importancia la elección del dataset, ya que el algoritmo solo puede capturar las relaciones que aparezcan en los datos de los que dispone.

Existen diversos estudios en los que se ataca este reto aplicando distintos algoritmos de aprendizaje automático: Make3D [3], Karsch&al [4], Ladicky&al [5], etc. Recientemente, el algoritmo de redes neuronales de aprendizaje profundo está dando excelentes resultados, superando al estado del arte. Ejemplo de ello son Eigen&al [6] o Liu&al [7], donde es implementado de diferentes formas. Por ello, este es el algoritmo seleccionado para este proyecto.

En concreto, el objetivo de este TFG es el de resolver el problema de reconstrucción 3D a partir de una única vista aplicando una red neuronal artificial (RNA) de aprendizaje profundo. Se diseñarán nuevos modelos de RNA en los que se intentará obtener la reconstrucción 3D de una forma precisa, sirviendo a la vez como trabajo de investigación de este algoritmo.

Para la ejecución de este proyecto, el pipeline se puede desglosar en las siguientes tareas:

1. Revisión bibliográfica del estado del arte
2. Estudio, comparativa y elección del dataset RGBD
3. Revisión de herramientas para su implementación: librerías, software, hardware, etc.
4. Desarrollo del modelo de la RNA
   4.1. Diseño de la estructura de la RNA
   4.2. Implementación
   4.3. Análisis de resultados y rediseño del modelo

Tras la ejecución de estos pasos, se han diseñado y analizado tres modelos distintos de redes neuronales, llegando a conseguir resultados similares a los del estado del arte. Estos modelos y sus respectivos resultados, así como la metodología seguida para obtenerlos, serán explicados a lo largo de la memoria.

2 REDES NEURONALES ARTIFICIALES (RNA)

Este es el algoritmo de aprendizaje automático seleccionado para resolver el problema. Posee una estructura capaz de representar relaciones complejas de los datos, siempre que disponga de los parámetros y datos suficientes. Si bien son conocidas desde hace décadas, recientemente han experimentado una difusión debido a los últimos avances de la computación que han logrado que el entrenamiento de las redes se pueda realizar en tiempos aceptables, a la vez que la adquisición de las grandes cantidades de datos que requieren las redes neuronales ha sido cada vez más fácil debido al progreso de las TIC.

De este modo se resolver el problema de la inferencia de la profundidad a partir de una única imagen, así como de forma paralela ahondar en el funcionamiento de las redes neuronales, del que todavía queda mucho por entender e investigar.

2.1 Modelo de una RNA

La unidad básica de este las redes neuronales es la célula ('cell') o unidad, que trata de simular una neurona biológica al tomar información de entrada de varias fuentes $X = (x_1, x_2, \dots, x_n)^T$ y producir una salida que representa el procesamiento que ha hecho la neurona a partir de la información de entrada.

Matemáticamente, la activación de una célula a partir de la información de entrada es la siguiente:

$$h_{W,b}(X) = f(W^T X) = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Como se puede observar, se trata de un producto escalar (más un término de sesgo o bias b) del vector de entrada $X = (x_1, x_2, \dots, x_n)^T$ y los parámetros o pesos propios de la célula $W = (w_1, w_2, \dots, w_n)^T$, al que se le aplica la función de activación $f()$. Los parámetros de la célula W serán obtenidos en el proceso de aprendizaje de forma automática, mientras que $f()$ puede ser cualquier función $f: R \longmapsto R$. En este TFG, se ha trabajado con la función sigmoidal logística (Figura 2.1) al ser una de las más usadas en las activaciones de las neuronas.

$$f(z) = \frac{1}{1 + exp(-z)}$$

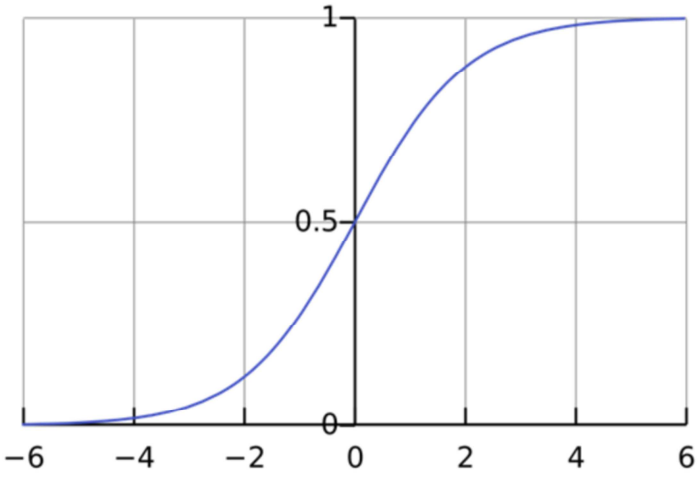


Figura 2.1: Expresión y representación gráfica de la función sigmoidal logística.

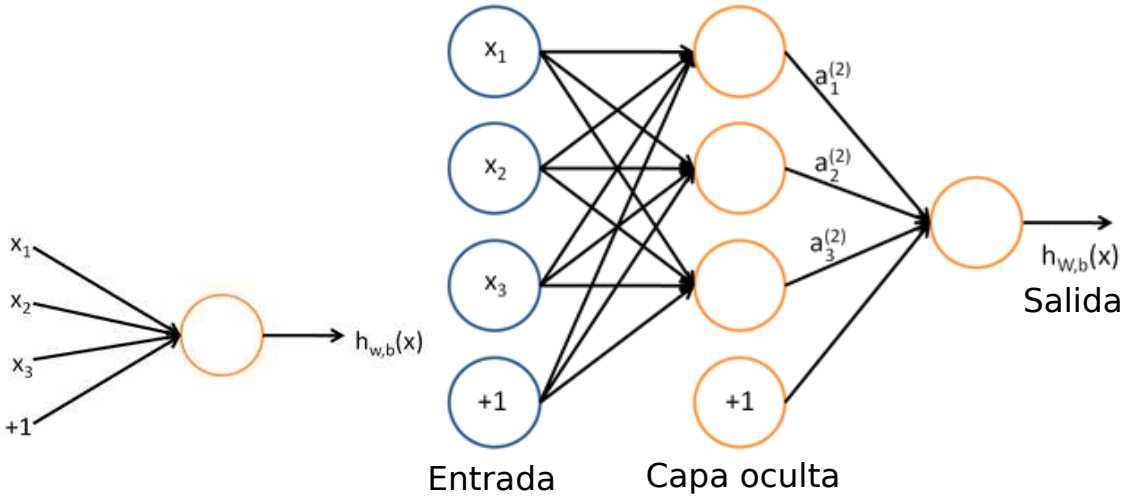


Figura 2.2: Respresentación de una neurona (izda.) y de una red neuronal con una capa oculta (dcha.). [8]

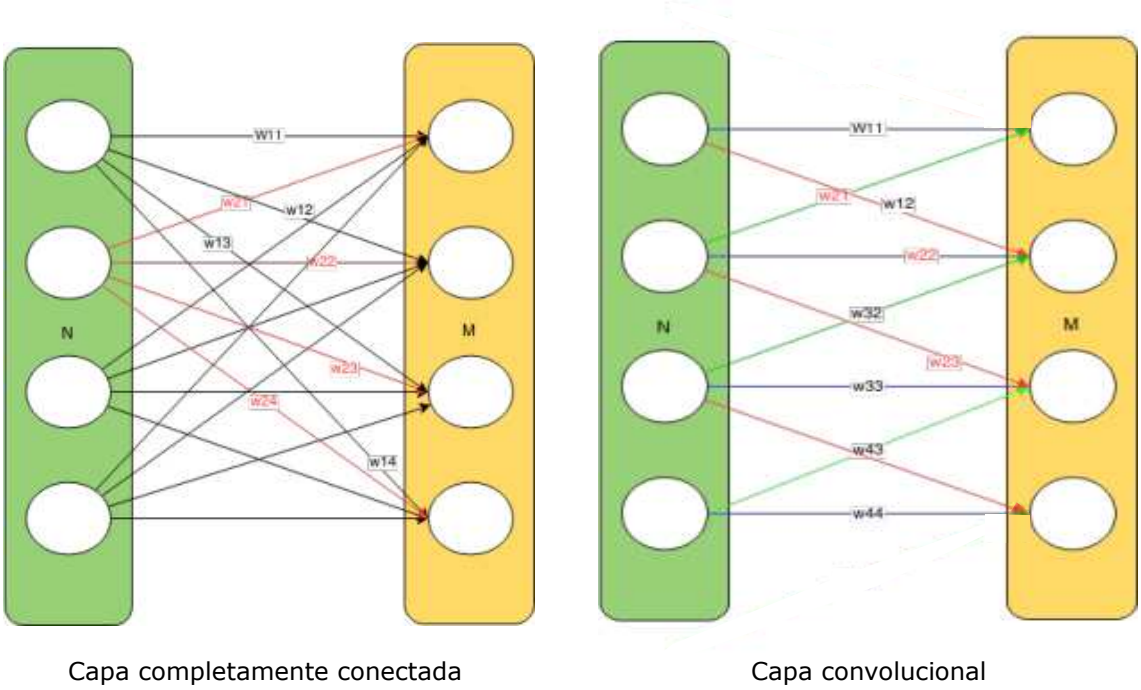
Con la estructura de una neurona definida, puede observarse que se trata en realidad de una regresión logística, uno de los algoritmos básicos de aprendizaje automático. Este algoritmo es capaz de establecer modelos lineales sobre los datos de entrada, a los que les es aplicada la activación sigmoidal logística. Por lo tanto se trata de un modelo que es capaz únicamente de representar relaciones relativamente simples. El poder de las redes neuronales aparece al agrupar un conjunto de neuronas para que trabajen interaccionando unas con otras, de un modo similar al que lo hacen nuestras propias neuronas. Esto se logra agrupando las neuronas en capas. Las neuronas de una misma capa procesan de forma paralela la misma información de entrada. Si después de una capa de neuronas se añade otra nueva capa, lo que se obtiene es un nuevo grupo de neuronas que trabajan sobre la información elaborada por la capa anterior de neuronas. Cuando una neurona emplea la salida de otra neurona como información de entrada, se dice que están conectadas. En la Figura 2.2 se representa un modelo simple, donde entre la entrada y la salida se introduce una capa de neuronas. A todas las capas distintas de la entrada y la salida se les suele llamar capas ocultas, ya que desde un punto de vista de caja negra, dichas capas procesan pasos internos del algoritmo para la obtención de la predicción de la salida a partir de la información de entrada.

2.2 Estructura de las capas

2.2.1 Capa completamente conectada ('fully-connected')

En esta capa, todas las neuronas de la capa actual (capa amarilla en Figura 2.3) están conectadas con todas las neuronas de la capa anterior (capa verde). Ello hace que sea una capa genérica al no condicionar ni restringir ninguna posible relación entre las neuronas de ambas capas.

Cuando se desconoce la estructura del problema, las capas totalmente conectadas resultan idóneas ya que permiten capturar todas las relaciones posibles. Sin embargo, son costosas computacionalmente al requerirse un elevado número de parámetros para establecer las conexiones de todas las neuronas. Esta capa requiere $K\cdot (L+1)$ parámetros, donde K es el número de neuronas de la capa actual y L es el número de neuronas de la capa anterior. Por este coste computacional no puede aplicarse a datos de elevado número de dimensiones.



Capa completamente conectada Capa convolucional

Figura 2.3: Representación esquemática de dos capas comunes de RNA. [9]

2.2.2 Capa localmente conectada: capa convolucional

Cuando la capa completamente conectada resulta demasiado costosa computacionalmente, se pueden quitar parte de las conexiones entre las capas para eliminar parámetros y disminuir el coste computacional del algoritmo. Este tipo de capas se les denomina como capas localmente conectadas.

Una de las capas localmente conectadas más difundidas es la capa convolucional (ver Figura 2.3). En este caso, cada neurona de la capa actual solo se conecta con un grupo local de neuronas de la capa anterior. Además, los parámetros W son los mismos para todas las neuronas de la misma capa. Es decir, en la Figura 2.3, las conexiones representadas por flechas del mismo color tienen los mismos pesos w_{ij}. Manteniendo el número de neuronas, el número de parámetros se ve considerablemente reducido con respecto a las capas totalmente conectadas. Esta reducción dependerá del número de conexiones locales que se mantengan.

2.3 Entrenamiento

Una vez definida la estructura de la red neuronal (entrada y salida de la red, número de capas, número de neuronas en cada capa) se procede al entrenamiento de la red a partir de los datos de entrenamiento. El objetivo del entrenamiento es minimizar una función de coste $F()$ que evalúe la diferencia entre la salida de la red y la verdadera salida de los datos, denominada etiqueta de los datos. Esto se realiza a través de un proceso iterativo:

- Se computan las activaciones de todas las neuronas, aplicando en cada una sus propios parámetros y sus correspondientes entradas, obteniendo como resultado la salida que obtiene la red de los datos de entrenamiento. Esto se denomina pase hacia delante (‘forward-pass’).

- Se aplica la función de coste $F()$ entre la salida de la red neuronal y la etiqueta de los datos de entrenamiento y se calcula el gradiente de dicha función de coste sobre la

salida de la red neuronal (última capa).

- El gradiente se propaga a todos los parámetros de la red neuronal empleando la regla de la cadena ('backward-pass').

- Con el gradiente de la función de coste respecto a todos los parámetros, éstos se actualizan en la dirección del gradiente negativo, de modo que el error o función de coste de la red descienda con cada iteración:

$$w_{ij}{}^{(l)} = w_{ij}{}^{(l)} - \alpha \frac{\partial}{\partial w_{ij}{}^{(l)}} F(W, b)$$

$$b_i{}^{(l)} = b_i{}^{(l)} - \alpha \frac{\partial}{\partial b_i{}^{(l)}} F(W, b)$$

Donde $w_{ij}^{(l)}$ es el peso asociado a la neurona *i* de la capa *l* con respecto a la neurona *j* de la capa anterior; $b_i^{(l)}$ es el término bias asociado a la neurona *i* de la capa *l*; α es el factor de aprendizaje; *F* la función de coste; y *W* y *b* la totalidad de los parámetros de la red neuronal.

2.4 Sobreajuste

Un problema general del aprendizaje automático es el sobreajuste o 'overfitting'. Sucede cuando el algoritmo se aprende únicamente los datos de entrenamiento, siendo incapaz de generalizar conclusiones y dando malos resultados cuando trata de predecir un ejemplo que no se haya visto en la fase de entrenamiento. Estos efectos aumentan en modelos con muchos parámetros y/o con cantidades de datos de entrenamiento demasiado pequeñas. A continuación, se explican brevemente algunas técnicas para remediar el sobreajuste.

2.4.1 Ampliación de la cantidad de datos

Este problema puede solucionarse aumentando la cantidad de datos de entrenamiento. Es la mejor forma de solucionar el sobreajuste, ya que es el propio algoritmo el que corrige sus errores al disponer de más información de la que aprender. Sin embargo no siempre es aplicable ya que puede ser imposible acceder a más datos o resultar demasiado costoso en esfuerzo y/o recursos.

2.4.2 Regularización

Cuando aparece sobreajuste, la variable predicha suele poseer bruscos cambios para adaptarse a los datos de entrenamiento (en la Figura 2.4, la línea de frontera se adapta totalmente a los datos de entrenamiento). Ello implica que los parámetros del modelo toman valores de elevado valor absoluto para obtener dichas bruscas variaciones.

Por ello, la regularización añade a la función de coste un término que penaliza elevados valores de los parámetros. En la norma L2, se emplea la suma cuadrática del valor de los parámetros. Dicho término va ponderado por un factor λ frente a la función de coste del error de predicción. Valores muy bajos λ no solucionan el problema del sobreajuste al apenas modificar la función de coste, mientras que valores demasiado elevados harán que la optimización se centre en el término regularizador en vez de en la función de coste del error del problema, obteniendo modelos excesivamente simplificados (problema de sesgo o subajuste).

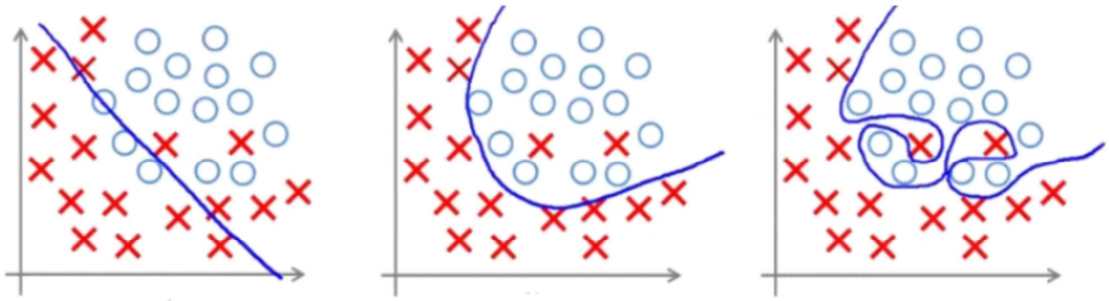


Figura 2.4: Representación simplificada del sobreajuste/subajuste en un problema de clasificación. [10]

Los parámetros del modelo entrenado serán aquellos que minimicen conjuntamente la función de coste y el término regularizador:

$$W = argmin_W \sum_i [F(W, X_i, Y_i)] + \lambda R(W),$$

Si norma L2: $R(W) = \|W\|^2$

donde F es la función de coste del problema; R el término de regularización con su ponderación λ; X_i son los datos de entrada disponibles e Y_i sus predicciones deseadas; y W los parámetros del modelo o algoritmo. La ponderación λ no es aprendida por el algoritmo, sino que según el valor de λ la solución del algoritmo variará. Se trata de un híper-parámetro que debe ser ajustado por el programador, al igual que el número de capas ocultas, número de neuronas en cada capa oculta, etc.

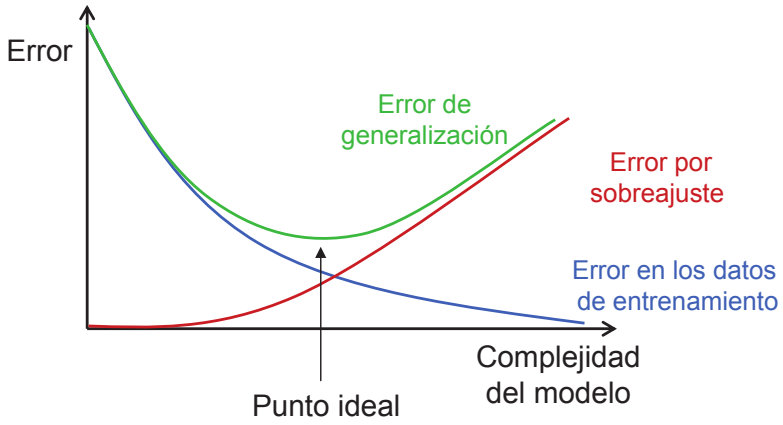


Figura 2.5: Gráfica cualitativa de la evolución del error con la complejidad del modelo, y la aparición del sobreajuste. [11]

2.4.3 Dropout

Esta técnica consiste en que, en cada iteración de entrenamiento, para aquella capa en la que se aplique el dropout, cada una de sus neuronas tendrá una probabilidad 'p' (ratio de dropout) de ser desactivada. Este ratio de dropout es otro híper-parámetro de la red. Por desconectar o desactivar se refiere a que no se calcula el output de dicha neurona, ni es tenido en cuenta para el cómputo de la siguiente capa. Tampoco se calcula su gradiente respecto a la función de coste. En la Figura 2.6 se representa este proceso.

La efectividad de este método está basada en que empíricamente las neuronas pueden llegar a trabajar dependiendo demasiado unas de otras, de modo que cada una por separado no llega a dar una información significativa. Al desprender aleatoriamente las neuronas en cada iteración, se fuerza a que cada una de ellas proporcione información significativa por sí misma independientemente del resto, lo que ayuda a remediar el sobreajuste ya que las relaciones deben generalizarse al contar cada vez con información distinta (distintas neuronas activadas) [12]. Indicar que el dropout no es exclusivo con la regularización, pudiendo aplicarse ambas técnicas de forma simultánea.

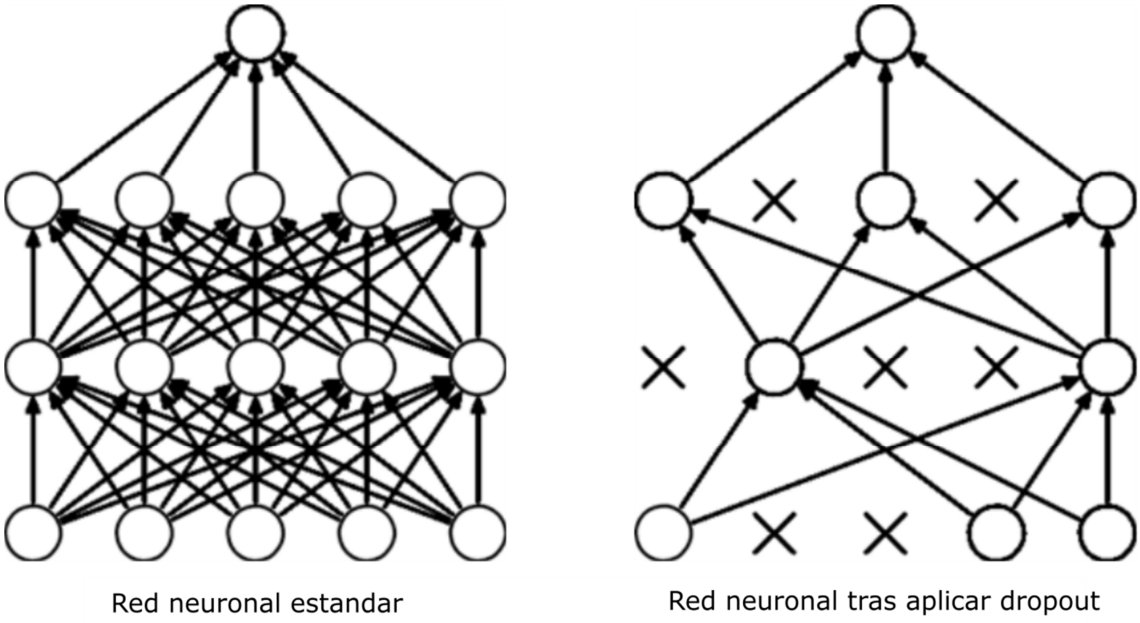


Figura 2.6: Representación de una RNA antes y tras aplicar dropout. [12]

2.5 Aprendizaje no supervisado: autoencoder

En el aprendizaje supervisado, para realizar cualquier tarea se necesita de los denominados datos etiquetados, como se ha supuesto en los apartados anteriores. Sin embargo, también pueden extraerse información de datos sin etiquetar: es lo que se llama aprendizaje no supervisado. Tiene la principal ventaja de que no necesita datos etiquetados que pueden ser escasos o costosos de conseguir. Es típico disponer de una cantidad no muy grande de datos etiquetados, mientras sí se disponga de una mayor cantidad de datos sin etiquetar.

Dentro de las redes neuronales artificiales, los autoencoders son unos modelos representativos de aprendizaje no supervisado. En ellos, se pasa de la capa inicial de los datos

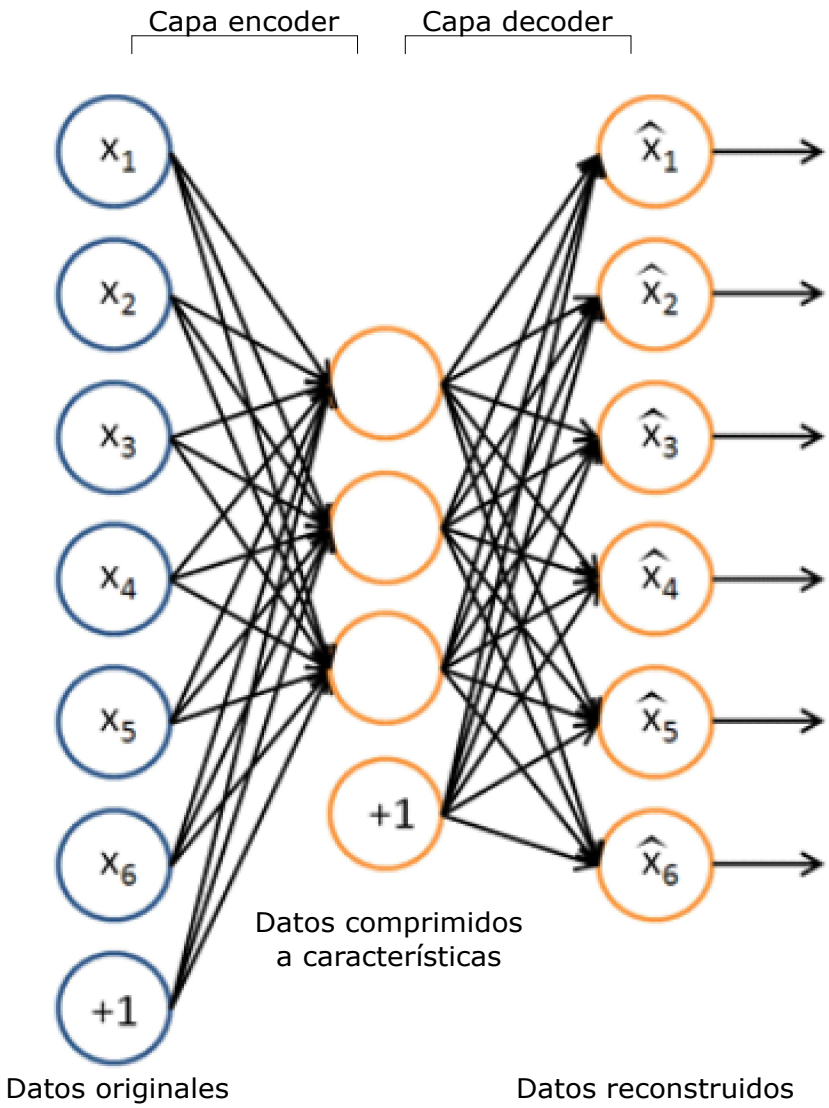


Figura 2.7: Esquema de un autoencoder. [8]

originales a una capa intermedia con menos neuronas o con alguna propiedad particular (p. ej. dropout), para luego reconstruir los datos originales (ver Figura 2.7). De este modo, se consigue una capa oculta de neuronas que contiene la información de los datos originales comprimida, denominada encoder. La capa que reconstruye los datos a partir de la compresión se denomina decoder. Al codificar los datos de entrada con el encoder, se obtienen unos valores que se denominan características de dichos datos, ya que su extracción ha sido aprendida por la red neuronal a partir de los datos para su propia reconstrucción. Al contener la información de los datos de entrada en un menor número de dimensiones, tienen interesantes aplicaciones, como p.ej. el pre-entrenamiento no supervisado (ver apartado 2.6.1).

2.6 Aprendizaje profundo ('Deep learning')

El aprendizaje profundo consiste en trabajar con sucesivas abstracciones del problema, llegando hasta arquitecturas capaces de modelar relaciones de alto nivel. De este modo, se puede resolver problemas muy complejos, donde otras técnicas de aprendizaje automático han dado resultados poco precisos.

En el caso de una red neuronal artificial, esto se consigue aumentando las capas

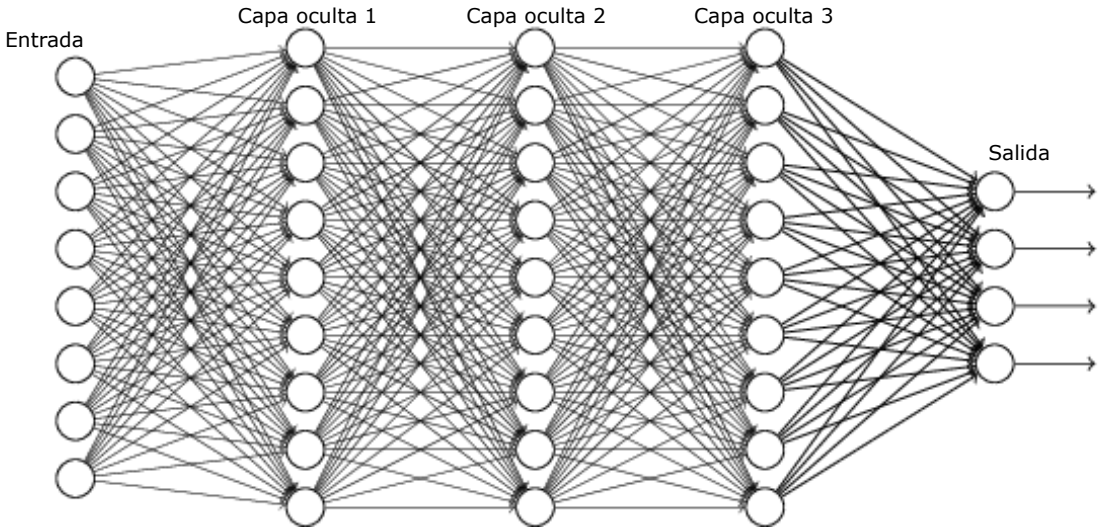


Figura 2.8. Representación de una red neuronal profunda o de aprendizaje profundo. [13]

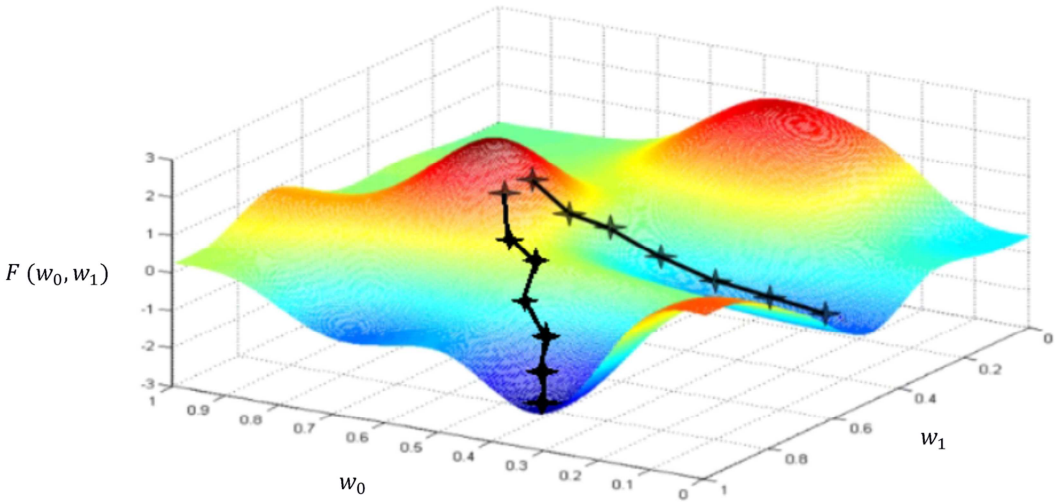


Figura 2.9: Ejemplo de función de coste no convexa, donde según la semilla el gradiente descendiente finaliza en distintos mínimos. [10]

ocultas del modelo (ver Figura 2.8). Cada capa de neuronas trabaja sobre la salida de la capa anterior, llegando a procesar cada vez de mayor nivel y complejidad. Este procesamiento puede realizarse también con una única capa oculta, pero para poder modelar estas relaciones de alto nivel (poder de representación), el número de neuronas crece exponencialmente con la disminución del número de capas [8].

Las redes neuronales profundas conllevan también una serie de desventajas. Debido al mayor poder de representación de las redes profundas, si son entrenadas con insuficientes datos presentarán con más facilidad problemas de sobreajuste, por lo que necesitan grandes cantidades de datos que no siempre se encuentran disponibles. La optimización de las redes neuronales, profundas y no profundas, presenta una función de coste no convexa. Esto causa que dependiendo de la inicialización de la semilla para el proceso iterativo, el modelo pueda converger a un mínimo local y no al global del problema (ver Figura 2.9). Mientras que en redes superficiales o poco profundas este mínimo suele seguir dando resultados aceptables,

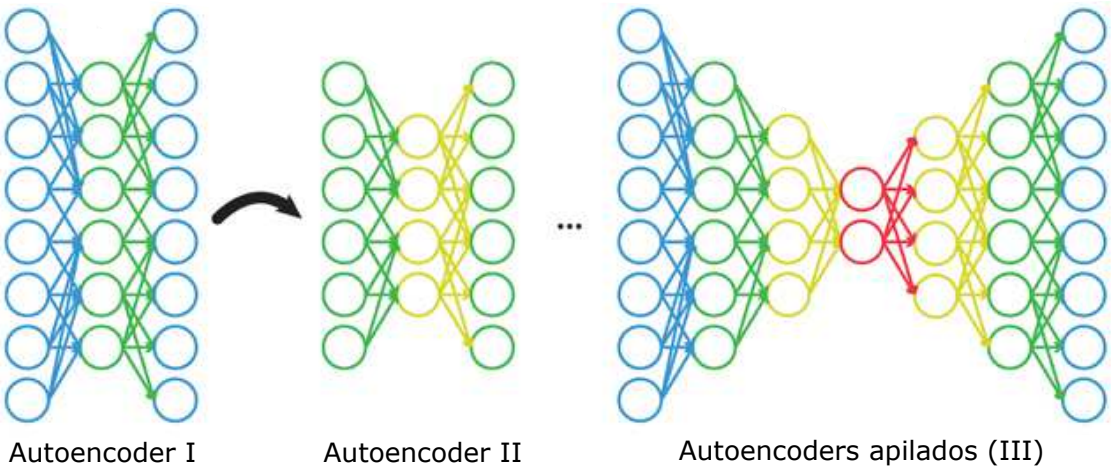


Figura 2.10: Representación de la construcción de tres autoencoders apilados. [14]

esto ya no es así en redes neuronales profundas, haciendo que los algoritmos basados únicamente en el gradiente local dejen de funcionar tan bien, llegando en ocasiones hasta la no convergencia. Finalmente, aparece un fenómeno llamado 'difusión de gradientes'. Al ejecutar la propagación hacia atrás de los gradientes, estos disminuyen en magnitud, provocando que únicamente las últimas capas se vean modificadas en el proceso de aprendizaje, mientras que en las primeras el gradiente es tan pequeño que apenas varían. De este modo, entrenar una red profunda con una inicialización aleatoria de sus parámetros da un resultado similar a entrenar una superficial (solo aprenden las últimas capas) sobre una entrada corrupta por las primeras capas de la red [8].

Para remediar estos, problemas se han desarrollado una serie de estrategias de entrenamiento por capas, que experimentalmente han demostrado dar buenos resultados:

2.6.1 Pre-entrenamiento no supervisado: autoencoders apilados

El gradiente descendiente lleva a la red neuronal a la cuenca de atracción en la cual ha sido inicializada la semilla, que es el punto del espacio multidimensional donde se localizan los parámetros del modelo. Por tanto, con una inicialización aleatoria no se está asegurando en absoluto que dicha semilla sea introducida en una cuenca de atracción donde el error sea pequeño, o la solución obtenida no sufra de sobreajuste.

Una solución es el pre-entrenamiento no supervisado [15]. Consiste en la inicialización de los parámetros de la red con unos valores predeterminados obtenidos a partir de los propios datos de entrenamiento. Estos valores pueden obtenerse a partir de unos autoencoders apilados. Estos autoencoders apilados consisten en varios autoencoders construidos cada uno sobre las características del anterior (ver Figura 2.10), obteniendo sucesivas compresiones de los datos de entrada. Las características son cada vez de mayor abstracción al haber pasado por una mayor cantidad de encoders.

Las primeras capas de la red neuronal se inicializan con las capas encoders de los autoencoders apilados. De este modo, la semilla inicial está situada en una zona donde las primeras capas son capaces de capturar la estructura de los datos de entrada. Si la propia estructura de los datos está relacionada con la variable a predecir, la cuenca de atracción donde ha sido inicializada poseerá un buen mínimo local. En [15] muestran como en los

experimentos llevados a cabo no es que se consiguiera un error menor realizando el pre-entrenamiento, sino que dicha solución generaliza mejor y sufre un menor sobreajuste.

2.6.2. Ajuste fino ('fine-tuning')

El ajuste fino es el último proceso para el entrenamiento de una red profunda, y consiste en tratarla como un único bloque en el entrenamiento, actualizando todas las capas de su red, igual que el entrenamiento estándar de una red no profunda. Al situar una semilla inicial con el pre-entrenamiento no supervisado de los autoencoders apilados, se minimizan los inconvenientes mencionados sobre el entrenamiento de las redes profundas. Las primeras capas ya han sido entrenadas, disminuyendo la importancia de la difusión de gradientes, y la semilla es inicializada en una zona de bajo error y buena generalización, corrigiendo la tendencia al sobreajuste de las redes profundas [15].

3 ESTRUCTURA DE LOS MODELOS

En la predicción de profundidad a partir de una imagen, la imagen RGB es el dato de entrada y la imagen de profundidad es la salida del problema. La red neuronal tendrá estas mismas entradas y salidas y la implementación de la red será tal que reciba el valor de intensidad RGB de los píxeles de la imagen como entrada, y su última capa predecirá el valor de profundidad de cada píxel (ver Figura 3.1). Las capas ocultas la red deberán tener una estructura capaz de capturar las relaciones entre la entrada y la salida en el proceso de entrenamiento.

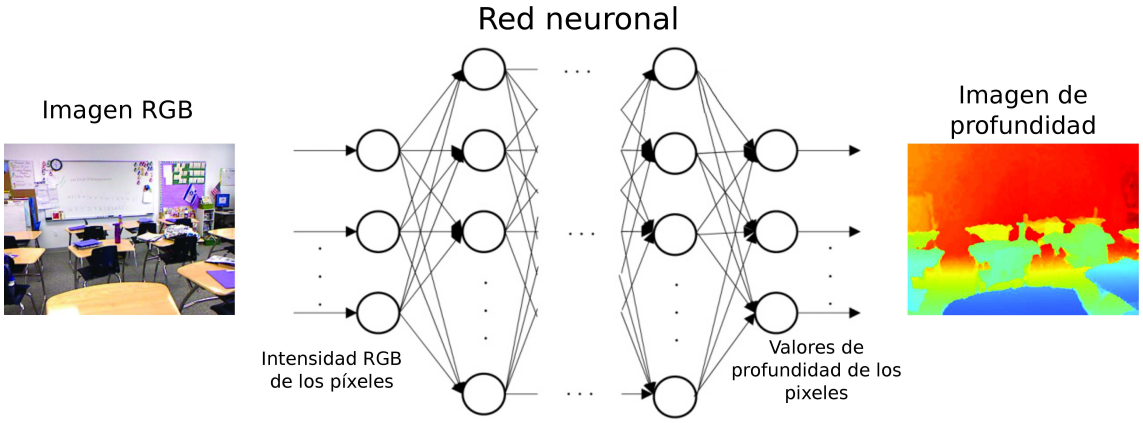


Figura 3.1: Esquema de la implementación de la RNA para la resolución del problema.

Esta red neuronal podría entrenarse directamente, diseñando una estructura concreta e inicializando sus parámetros aleatoriamente. Sin embargo, los resultados experimentales de [15] muestran que las mejoras del pre-entrenamiento no supervisado ya aparecen con una única capa oculta. Estos resultados se obtienen sobre el MNIST database [16], que consiste en clasificar imágenes de dígitos escritos a mano según el valor del dígito (0-9). Para ello emplea autoencoders apilados como pre-entrenamiento de los datos de entrada de la red.

En nuestro caso se realiza una regresión sobre una variable compleja en comparación con el valor de un dígito: la profundidad de una escena. Por tanto, puede que una inicialización donde se capture la estructura de la entrada (RGB) no sea suficiente para predecir la salida (imagen de profundidad). Podrían necesitarse varias capas tras los autoencoders apilados de la imagen RGB para conectar estas características RGB con la profundidad de la escena, por lo que se tendría de nuevo una red profunda con los mismos problemas que se trataban de resolver con el pre-entrenamiento no supervisado.

Para solucionar dicho problema, se propone aplicar la técnica de pre-entrenamiento no supervisado tanto a la entrada como a la salida de la red. Esta es la principal contribución del proyecto, ya que hasta ahora en el pre-entrenamiento solo se analizaba la estructura de la entrada. El motivo es el mismo que se explica en [15]: si con unos parámetros que capturan la estructura compleja de la entrada resulta más fácil para el algoritmo predecir la salida, cuando esta última sea también compleja, unos parámetros que atrapen también la estructura de la salida facilitarán la resolución del problema, ya que consistirá en la conexión características aprendidas de la entrada con características aprendidas de la salida, relación que puede ser más simple de aprender que la conexión original entrada-salida.

Es decir, lo que se busca es un mapeo entre características de las imágenes RGB sobre características de las imágenes de profundidad. Las personas no asociamos un color a un valor de profundidad directamente, sino características suyas, como por ejemplo una línea a una discontinuidad de profundidad. Por tanto, todos los modelos aquí presentados se basan en esta idea: RGB → Características RGB → Características profundidad → Profundidad.

El proceso para el pre-entrenamiento de los autoencoders de la salida es similar al explicado para pre-entrenar los autoencoders asociados a la entrada, salvo que en vez de escoger las capas encoder de la salida, se tomarán las capas decoders. Así se obtiene a partir de unas características codificadas la salida original.

El proceso completo de entrenamiento de la red neuronal será el siguiente (ver Figura 3.2):

- Fase 1. Pre-entrenamiento no supervisado de los autoencoders de imágenes RGB (entrada) e imágenes de profundidad (salida). El número de autoencoders puede variar según el modelo planteado. En este Fase 1 no se establece ninguna conexión entre imagen RGB e imagen de profundidad, siendo ambos autoencoders completamente independientes.
- Fase 2. Entrenamiento de la capa de conexión. Se implementa una capa tal que conecte las características RGB y las características de imagen de profundidad obtenidas en el Fase 1. Esta nueva red tendrá la siguiente estructura: encoder RGB (entrenada en Fase 1) →Capa de conexión → Decoder de profundidad (entrenada en Fase 1). Durante esta fase del entrenamiento, las capas encoder RGB y decoder de imagen de profundidad se dejan fijas o congeladas, actualizando solo las que hacen de conexión. Si esto no se hiciera, al decoder de la imagen de profundidad le llegaría una información corrompida por la capa de conexión inicializada aleatoriamente. Por tanto, debido tanto al problema de difusión de gradientes como el de cuencas de atracción, la solución alcanzada se vería influenciada por dicha inicialización aleatoria, ya que las últimas capas perderían rápidamente su capacidad de decoder para adaptarse a la información corrompida por la capa intermedia.
- Fase 3. Fine-tuning de la red final. Las capas se inicializan con las obtenidas en el Fase 2, manteniendo la misma estructura. La diferencia con el Fase 2 es que ahora todas las capas se actualizan en el entrenamiento de la red.

A estos procesos se les denominará a lo largo de la memoria como Fase 1, 2 ó 3 (Figura 3.2).

Otro método de entrenamiento puede ser el entrenamiento directo de la red final, inicializando sus capas aleatoriamente. De este modo se comienza directamente en el Fase 3, salvando los dos primeros. Sin embargo, es predecible que aparezca un mayor sobreajuste o incluso la no convergencia conforme aumente la profundidad de la red (apartado 2.6), aunque incluso con una única capa oculta se aprecian los beneficios del pre-entrenamiento como ya se ha indicado.

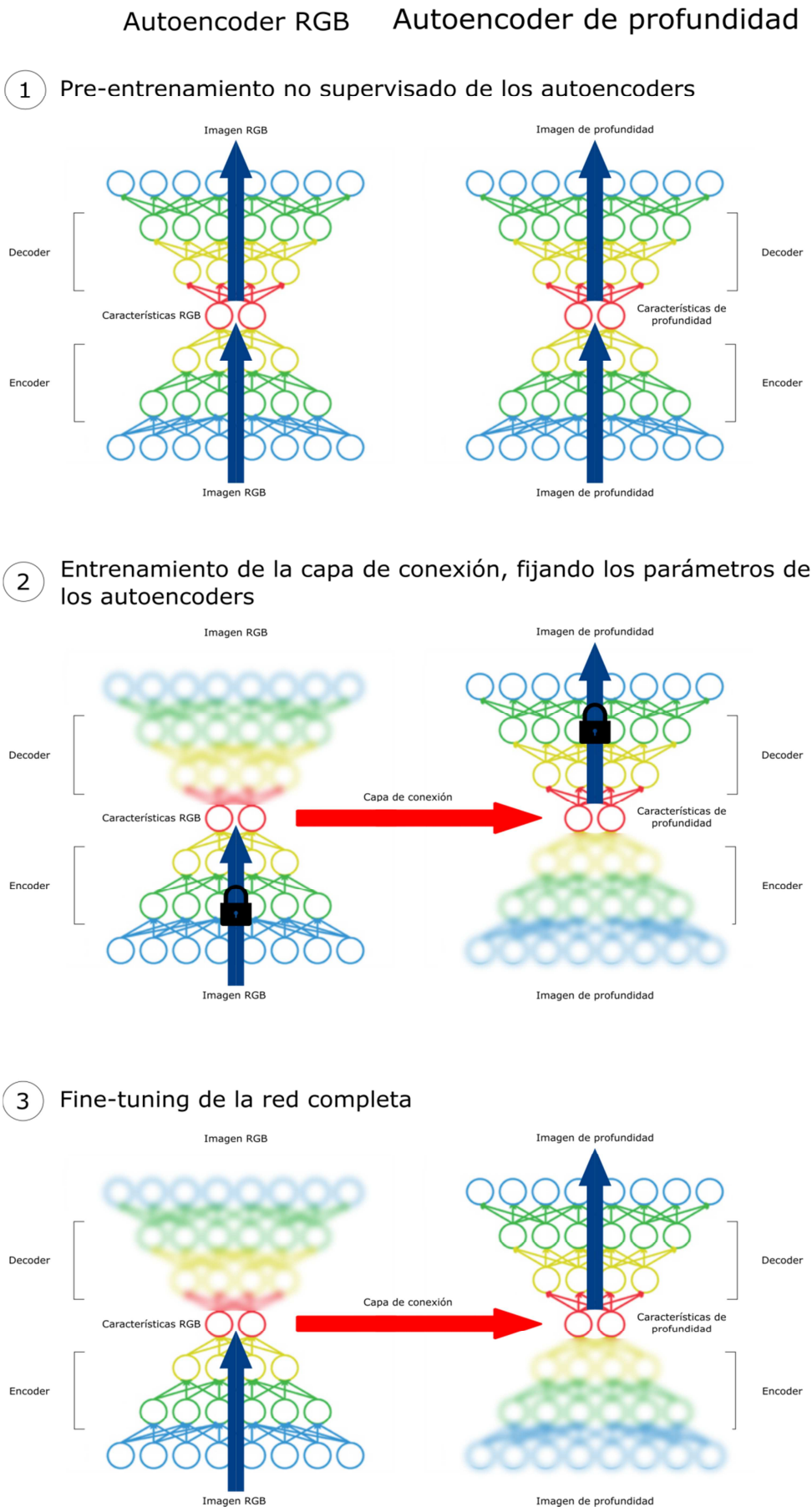


Figura 3.2. Esquema de las tres Fases de entrenamiento de los Modelos.

Se han diseñado tres modelos basados en este proceso de entrenamiento, distinguiendo entre si emplea como entrada una imagen RGB entera ("global") o solo una ventana de esta ("local"), y el uso o no del aprendizaje profundo:

- Modelo 1 (M1): Local y sin aprendizaje profundo
- Modelo 2 (M2): Global y sin aprendizaje profundo
- Modelo 3 (M3): Global y con aprendizaje profundo

El tratamiento de una imagen RGB entera con capas totalmente conectadas es computacionalmente caro debido al elevado número de dimensiones de la entrada y al peso computacional de este tipo de capas. Por tanto, esta limitación tiene que ser abordada por los tres modelos propuestos.

3.1 M1: Local y sin aprendizaje profundo

Se trata del modelo más simple de todos. Según lo explicado, la aplicación de una RNA a una imagen entera queda descartada. Por ello, y para hacer una primera aproximación al problema, se aplica la idea anteriormente explicada (RGB → Características RGB → Características profundidad → Profundidad) de la forma más directa posible. Esto es, se implementa cada transformación con una capa totalmente conectada sobre pequeñas ventanas de 60x60 extraídas de forma aleatoria (ver Figura 3.3), obteniendo un total de dos capas ocultas. Estas ventanas RGB contienen 60·60·3 = 10.800 dimensiones, reduciendo el coste computacional del modelo.

Este modelo podrá inferir la profundidad de pequeñas ventanas de dimensión 60x60. Una forma de implementar el modelo para el trabajo con imágenes enteras sería realizar un escaneo de la imagen entera por ventanas 60x60, de los que la red neuronal inferiría su correspondiente profundidad, montando la imagen de profundidad completa a partir de las profundidades locales obtenidas en cada zona de la escena. El número de parámetros de esta red depende del número de neuronas en cada una de las capas ocultas: suponiendo una compresión de 1/10 en cada uno de los autoencoders, el número de parámetros del modelo alcanza los 13 millones.

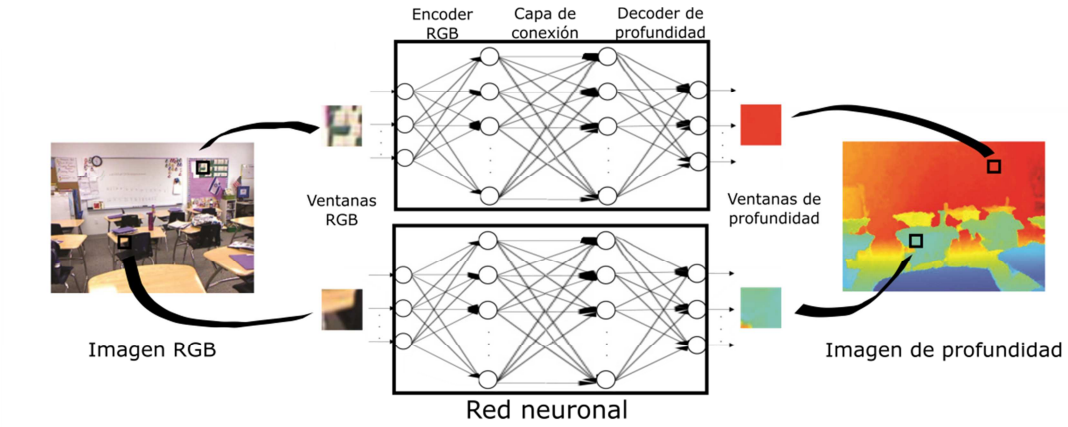


Figura 3.3. Esquema de la arquitectura del Modelo 1

Esta RNA solo dispone de la apariencia local de una zona de la imagen, a partir de la cual debe poder inferir su profundidad, sin disponer del resto de la imagen. Esto es significativo, ya que se asume que no se trata de un problema de carácter global en cuanto a la imagen, sino que se puede descomponer en pequeñas ventanas en paralelo y ser resuelto localmente.

3.2 M2: Global y sin aprendizaje profundo

Este modelo posee la misma estructura de red que M1, implementando cada transformación (RGB → Características RGB → Características profundidad → Profundidad) con una capa totalmente conectada. Pero estas son aplicadas sobre imágenes enteras subsampleadas a una resolución parecida a la de las ventanas analizadas con el M1: 48x64 (ver Figura 3.4). De esto modo, se aligera el coste computacional pero aplicando la red sobre ventanas enteras.

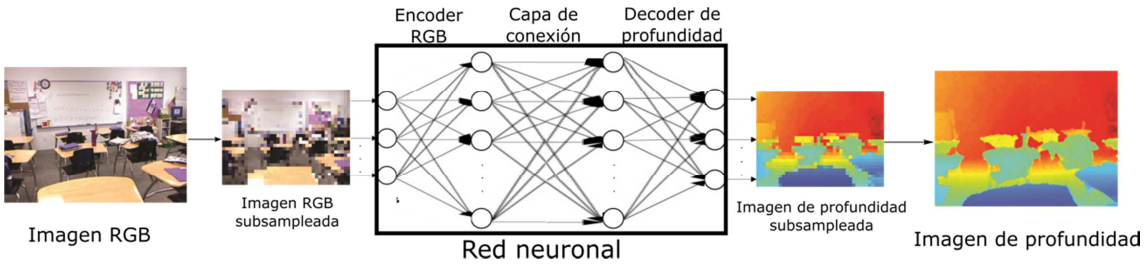


Figura 3.4. Esquema de la arquitectura del Modelo 2

La única diferencia estructural de la red es que se las dimensiones de la entrada y salida deben adaptarse a la nueva resolución. De querer obtenerse una predicción de profundidad cuya resolución se corresponda con la original de las imágenes, esto se puede realizar con métodos de 'upsampling', cuyos efectos no son analizados en este trabajo.

A pesar de no cambiar la estructura de la red neuronal con respecto a M1, el problema es muy distinto, ya que la entrada contiene ahora una imagen entera subsampleada. Ello implica que ya no se trata de resolver el problema con un carácter local a través de pequeñas ventanas, sino que la imagen entera es tratada como entrada, asumiendo un carácter global en la resolución del problema. Por tanto, información global de la imagen tales como líneas o puntos de fuga, grandes estructuras geométricas, etc. podrán ser capturadas y ser empleadas por la red.

Como en M1, el número de parámetros depende del número de neuronas de cada capa oculta. Suponiendo de nuevo una compresión de las características de 1/10 se obtiene una cantidad aproximada de diez millones de parámetros, algo inferior a la del Modelo 1 al ser las dimensiones de entrada y salida algo menores con la nueva resolución.

De forma más intuitiva, mientras que a las personas nos resulta muy difícil obtener la profundidad absoluta de una pequeña ventana de una imagen, nos es relativamente sencillo predecir la de una imagen entera, aunque se haya reducido su resolución. Ello se debe, como ya se ha indicado, a que el problema de la estimación de la profundidad a partir de una sola vista necesita, no solo de la estructura local de la zona, sino de la global de la escena.

3.3 M3: Global y con aprendizaje profundo

En los dos modelos anteriores se ha utilizado únicamente las capas totalmente conectadas, y el modelo final poseía dos capas ocultas. En este último modelo se propone el uso de las capas convolucionales como medio de conexión entre las distintas capas. De este modo se reduce el coste computacional del modelo, permitiendo incrementar la resolución de la entrada y salida, así como el número de capas de la red y aplicar el aprendizaje profundo. Mantener la resolución de las imágenes originales conlleva todavía un alto coste computacional, por lo que sigue siendo necesario subsamplear las imágenes, pero no hasta tan bajas resoluciones como en el caso de M1 y M2.

Estas capas convolucionales, además de permitir trabajar con datos de mayor dimensión, confieren una estructura espacial al modelo al aplicarse en imágenes debido a la forma en que trabaja el operador convolución (ver Anexo B). Las activaciones de la capa posterior solo se relacionan con una cierta zona local de la capa previa, y no con la totalidad de ella como hacían las capas totalmente conectadas. Ello provoca que no se pueda detectar con una sola capa patrones globales, sino únicamente locales.

En la primera capa convolucional de la RNA estos patrones locales, como los bordes, formas simples, gradientes, etc. pueden ser detectados por las unidades ocultas. La importancia de las convoluciones reside en que las activaciones se almacenan ordenadas espacialmente en una matriz, montando de nuevo una imagen. Esta imagen contiene las activaciones de las neuronas, que en el caso de tratarse de un autoencoder, son las características de los datos de entrada. De esta forma, la siguiente capa puede trabajar con esta información ordenada espacialmente en una imagen y asignar distintos significados a un mismo patrón según su localización espacial en la imagen original. Esta estructura espacial es algo característico de este problema, ya que no significa lo mismo una línea por la zona inferior (probablemente el borde inferior de un mueble) a una lateral (pared), central (mesa, cama,…) o superior (techo, lámpara,…).

En la tarea de clasificación de imágenes, es común aplicar varias capas convolucionales a las imágenes, obteniendo en primer lugar características locales, que en cada capa posterior van adquiriendo un carácter más global y de más alto nivel. Con la última capa obtenida, es común aplicar una capa totalmente conectada para obtener la variable deseada. Aplicar dicha estructura en este problema en particular, implicaría que la obtención de la profundidad de la escena se basa en la ponderación de una colección profundidades predeterminadas asociadas cada una a una célula de la última capa oculta, por la activación de dicha neurona: es decir, interpolaría mapas de profundidades predeterminados.

Por ello es por lo que se aplica una simbiosis entre dicha estructura y el modelo propio de este trabajo: la estructura de la red será tal que aplique una conexión entre características RGB y características de profundidad, pero estas obtenidas a través de autoencoders convolucionales y su operación inversa, deconvolucionales (ver Anexo B), en vez de los modelos anteriores donde se obtenían con capas de producto escalar. Las capas se basan en convoluciones que tan buen resultado han dado en [6], pero en vez de conectar las características así obtenidas directamente a un mapa de profundidad, se conectan las características RGB con las características de imagen de profundidad.

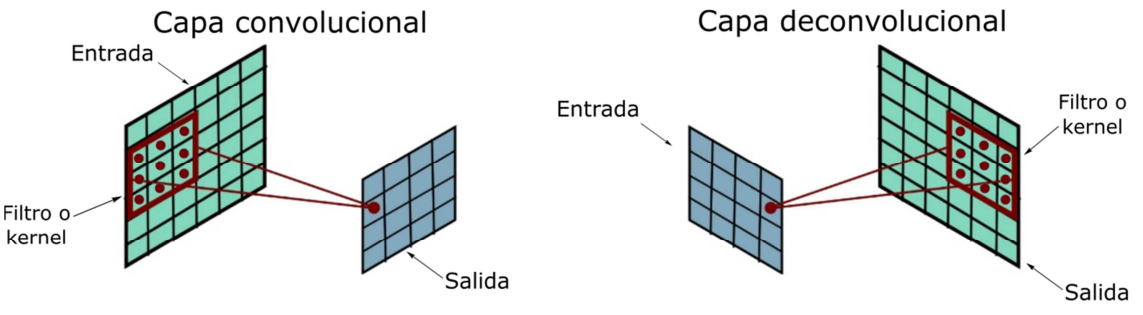


Figura 3.5: Representación explicativa de las capas convolucionales y deconvolucionales aplicadas en imágenes.

De este modo ya se puede aplicar el esquema del pre-entrenamiento no supervisado de los autoencoders apilados con capas convolucionales y deconvolucionales. Se emplearan unas compresiones de orden similar a las de [6], puesto que han demostrado dar buen resultado. Sin embargo, en este caso no pueden ser tan profundas, ya que mientras que en la red neuronal de [6] solo se añaden dos capas completamente conectadas tras las capas convolucionales para obtener la profundidad, el encoder RGB de este modelo (M3) requiere al menos una capa de conexión y después el decoder correspondiente de la imagen de profundidad. Unido a limitaciones de velocidad del entrenamiento, se restringen los autoencoders apilados a dos capas de compresión. La estructura de la red resulta la mostrada en la Figura 3.6.

Por limitaciones temporales para la optimización de las dimensiones de los filtros y el número de características por capa, estas se escogen de forma que se escalen aproximadamente con las del modelo de [6]. Tanto las dimensiones escogidas como los parámetros de las capas convolucionales y deconvolucionales que permiten obtenerlas se encuentran en la Figura 3.6. Las imágenes se subsamplean hasta 95x131. Estos valores eran originalmente 96x128, pero encajar las dimensiones de salida de un autoencoder convolucional no es posible para todas las resoluciones, solo lo permiten determinadas. Por ello se cambia a 95x131, la más cercana que cumple dicha condición.

Las dimensiones de las capas se obtienen ajustando dos parámetros de las capas convolucionales y deconvolucionales: el tamaño del filtro (‘kernel size’) y la cantidad de pixeles que se desplaza entre cada operación (‘stride’). Además en la primera capa convolucional de la red final y de los encoders se aplica el ‘pooling’, que reduce la dimensión de las capas ocultas. Esto divide la imagen correspondiente a la capa convolucional en distintas submatrices, transformando cada una de ellas en un escalar según una norma definida: en este caso devuelve el máximo de dicha submatriz (‘max-pooling’). Así es aplicado en [6], por lo que este método parece funcionar bien en este problema en particular.

Las características comprimidas en los autoencoders no son completamente globales: la salida de ambos encoders (RGB y profundidad) son 2000 características de las que cada una se posee una matriz 2x2, correspondiéndose con las cuatro esquinas de la imagen de entrada. Esto es así por dos motivos: en primer lugar, los filtros aplicados en las convoluciones y deconvoluciones son de menor tamaño, acelerando el proceso de entrenamiento y solventando problemas de memoria de la implementación del Modelo; en segundo lugar, la relación de compresión se mantiene similar a la empleada en [6].

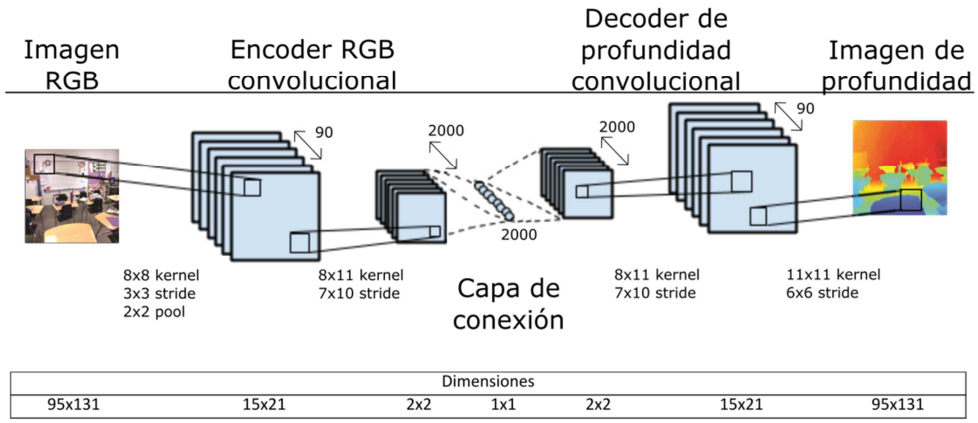


Figura 3.6: Esquema de la arquitectura del Modelo 3.

La capa de conexión que conecta las características RGB con las de profundidad no puede ser completamente conectada como en los anteriores modelos. La razón son limitaciones de memoria: se requeriría una matriz cuadrada de orden 8.000[1], por lo que se decide crear unas unidades de conexión intermedias entre las características RGB y las características de profundidad. Por tanto, la capa de conexión se diseña en dos fases con una capa intermedia de 2.000 unidades.

Por tanto, de 2.000 características RGB de 2x2 se obtienen 2.000 unidades intermedias y globales (dimensión 1x1). A partir de estas unidades intermedias, se computan las características de la imagen de profundidad. De este modo, se relacionan todas las características RGB con todas las características de profundidad a través de una capa intermedia que condensa la información y que por su menor tamaño (aunque ahora se aplique dos capas) alivia el coste computacional del entrenamiento.

Esta estructura así definida requiere de más de sesenta millones de parámetros, unas seis veces más que los Modelos 1 y 2. Las capas más pesadas son las completamente conectadas de la capa de conexión, donde cada una requiere 16 millones de parámetros.

Con esta estructura se trabaja, al igual que en el Modelo 2, capturando la estructura global del problema. Pero ésta no es obtenida a través de capas completamente conectadas, sino que se aplican las restricciones espaciales de las capas convolucionales que parecen ser coherentes con el problema aquí analizado. Además, se llega a una red de cinco capas ocultas, presentando una red con aprendizaje profundo. De forma adicional, dichas capas convolucionales permiten trabajar con mayores resoluciones de imágenes al requerir un menor número de parámetros. No resulta necesario un subsampleo tan brusco como en M2, por lo que la imagen de entrada al Modelo 3 sufrirá una menor pérdida de información con respecto a la imagen original de entrada.

[1] La librería Caffe (apartado 4.2) instalada en la GPU del ordenador no soporta la implementación de una capa de este tamaño.

4 METODOLOGÍA

4.1 Dataset

4.1.1 Elección del dataset

Existe un amplio abanico de datasets RGB-D disponibles donde entrenar los modelos. Entre los más difundidos se pueden mencionar los siguientes: RGBD Object dataset [17], NYU Depth Dataset [18], SUN3D [19], TUM Benchmark dataset [20], etc. Esta decisión es de gran importancia, puesto que la RNA aprende a partir de los datos de entrada. Cuanta mayor sea la variedad y cantidad de los datos, más información podrá aprender potencialmente la RNA.

El dataset escogido en este proyecto es el NYU Depth Dataset V2 [18]. La razón de la elección de este dataset en particular frente a otros datasets de RGB-D es porque muchos de los artículos publicados relacionados con esta área lo emplean como punto de referencia donde comparar resultados. La importancia de NYU Depth Dataset se basa en la gran variedad que este posee. Muchos datasets RGB-D se encuentran orientados al SLAM (Localización Y Mapeado Simultáneos), centrándose más en el movimiento de la cámara que en la variedad de escenas a lo largo del dataset; otros. NYU Depth Dataset V2 está orientado a la segmentación de interiores a partir de imágenes RGB-D, por lo que dispone de una gran heterogeneidad de escenas (464 en total). Esto hace que comprenda una mayor diversidad de riqueza e información visual frente a otros datasets.

NYU Depth Dataset V2 consta de 407.024 imágenes de vídeo RGB-D, en el formato de salida de la cámara. Es decir, no han recibido procesamiento de ningún tipo, aunque en la propia página web se dispone de código M para ello. Además, se dispone del denominado 'labeled dataset', una selección de 1.449 imágenes contenidas en un archivo binario de matlab que han sido etiquetadas de forma densa, y los huecos que aparecen en las imágenes de profundidad han sido rellenados aplicando el método de colorización de Levin et al's [21].

Sin embargo, 1.449 imágenes pueden resultar escasas para entrenar una red neuronal profunda. Por ello, se ha creado una nueva selección de imágenes extrayendo una de cada 85 imágenes del dataset completo, y procesándolas de modo similar a las pertenecientes al 'labeled dataset'. Además, se les aplicó un efecto espejo a todas ellas para duplicar el número de datos, obteniendo aproximadamente unas 11.000 imágenes. Este nuevo dataset será denominado en el documento como 'extension dataset'.

Figura 4.1: Imágenes del NYU Dataset. De izda. a dcha., imagen RGB, imagen de profundidad y segmentación densa de la imagen.

4.1.2 Partición del dataset

Al comparar los resultados del algoritmo implementado para escoger el de mejor rendimiento o el valor de sus híper-parámetros, no puede hacerse mediante sus resultados en los datos de entrenamiento, ya que un modelo con sobreajuste da muy buenos resultados, cuando en realidad no generaliza ante nuevos ejemplos (ver apartado 2.4). Por tanto, se deben evaluar los resultados en un conjunto de datos distinto al de entrenamiento. Este nuevo conjunto de datos es el de validación, y consta de ejemplos nuevos que el algoritmo no ha visto en su fase de entrenamiento. De este modo se pueden comparar los resultados de las distintas redes y ajustar sus híper-parámetros sin posibilidad de que presenten sobreajuste sobre estos datos.

El problema del sobreajuste va todavía más lejos, y es que al basar la elección de la estructura de la red y de sus híper-parámetros en la minimización de la función de coste en el conjunto de validación, estos valores se están aprendiendo en dicho conjunto. Por tanto, estos valores pueden padecer sobreajuste, tal y como lo pueden cometer los parámetros del modelo en el conjunto de entrenamiento. Ello implica que los resultados obtenidos en el conjunto de validación serán mejores de lo que mostrarían en datos completamente nuevos, al haberse aprendido los híper-parámetros a partir de ellos. Este efecto crecerá conforme mayor sea el número de híper-parámetros seleccionados y las estructuras analizadas.

Para evitar estos efectos debe definirse un nuevo conjunto con nuevos ejemplos que no aparezcan en los dos anteriores. Es el denominado conjunto de test. El objetivo de este conjunto de datos es dar una métrica fiable de los resultados del modelo, no ayudar a la elección de ninguno de sus parámetros. Una partición típica del total de los datos puede ser 70% aprendizaje, 20% validación, 10% test.

La metodología aplicada es la siguiente: se entrenan todos los modelos en el conjunto de entrenamiento y se optimizan sobre el de validación. Finalmente, para el modelo escogido, que será aquel que mejores resultados de sobre el conjunto de validación, se dará su métrica de rendimiento sobre el conjunto de test.

Estas particiones se deben de hacer sobre los datasets empleados. Existe una partición oficial de entrenamiento/test para el 'labeled dataset', donde 795 se destinan al entrenamiento y 654 al test. Este último set se ha partido en 436 de validación y 218 de test. Para el 'extension dataset', se ha hecho una partición de 70% de escenas para el entrenamiento (~8.000), 20% destinadas a validación (~2.100) y 10% a test (~1.100). Todos los modelos han sido entrenados y testeados en ambos datasets, aunque por meras cuestiones de limitaciones de tiempo, sus híper-parámetros han sido ajustados únicamente en uno de ellos.

- Modelo 1: optimizado en el 'labeled dataset'. Se han extraído 10.000 ventanas de 60x60 de forma aleatoria del conjunto de entrenamiento, 3.000 del de validación y 1.500 del de test.
- Modelo 2: híper-parámetros optimizados en las imágenes del 'labeled dataset'. A pesar de que solo dispone de 1.449 imágenes o ejemplos, aproximadamente diez veces menos que en el Modelo 1, el proceso de obtención del 'extension dataset' estaba comenzando en el momento en el que se planteó este modelo.

Por no desaprovechar un lapso de tres semanas de tiempo se procedió con el 'labeled dataset'.

- Modelo 3: debido al mayor número de parámetros del que dispone, y al tratarse de una red neuronal profunda de 5 capas ocultas, los híper-parámetros han sido optimizados en el 'extension dataset' por la mayor cantidad de datos que ofrece. En este caso ya se encontraba disponible cuando se comenzó con el entrenamiento del Modelo.

4.2 Software y hardware

Al igual que en el caso de la elección del dataset, existe un amplio catálogo de librerías para trabajar con redes neuronales. Por citar algunas: Fast Artificial Neural Network (FANN) [22], Caffe Berkeley Vision [23], Open NN [24], Theano [25] [26], Pybrain [27], etc.

La elección ha sido trabajar con Caffe Berkeley Vision [23]. Las razones fundamentales han sido la velocidad de procesamiento al poderse implementar en la GPU, la implementación a través de manejables archivos de configuración prototxt, y la disponibilidad de un 'wrapper' en Python con el que analizar los resultados.

Esta librería ha sido instalada en un PC con Ubuntu 12.04 LTS (Precise) como OS. Este PC disponía de una GPU NVIDIA GeForce 8400 GS. Esta tarjeta posee una 'compute capability' de 1.1, lo que la hace incompatible con la librería Caffe. Por ello, se procedió a la adquisición e instalación de la tarjeta GeForce GTX 750, GPU que sí es compatible con la librería.

El trabajo de programación de este proyecto se puede diferenciar en tres clases:

- Código en Matlab para un primer procesamiento del dataset.
- Archivos prototxt para la implementación de las redes neuronales en Caffe, y código Bash de Linux para organizar secuencialmente el entrenamiento de estas.
- Código en Python para un segundo pre-procesamiento más fino de las imágenes (ver 4.3), y para el análisis de los resultados.

4.3 Pre-procesamiento de los datos

Las imágenes en formato binario de Matlab requieren de cierto procesamiento para poder ser empleadas como entrada en los Modelos 1, 2 y 3 implementados en Caffe [23]:

- En primer lugar deben de ser ajustadas a las resoluciones especificadas en cada Modelo. El método de 'downsampling' empleado ha sido el bicúbico, al ofrecer mejores resultados que otros métodos más simples como el 'Nearest Neighbour' o la interpolación lineal.
- Las imágenes han de ser barajadas aleatoriamente al entrenar la red con el gradiente descendiente estocástico.
- Por último, el formato de entrada más eficiente en el software Caffe es a través de ficheros lmdb, por lo que tanto las imágenes RGB como las de profundidad han sido transformadas a este tipo de datos.

4.4 Entrenamiento de los modelos

Los modelos deben ser entrenados variando todos sus híper-parámetros, hasta encontrar los valores óptimos en el conjunto de validación. Cuantos más híper-parámetros se ajusten y más valores por híper-parámetro sean implementados, estos habrán sido más optimizados y el error en el conjunto de test será menor.

Sin embargo, entrenar una sola RNA pasando por todas las Fases del entrenamiento representados en la Figura 3.2, requiere de unas veinte horas para M1 y M2, y en torno a unos cuatro días para M3. Por ello, el número de pruebas que se pueden realizar es limitado. En vez de realizar todos los ajustes de híper-parámetros sobre la red final, se sigue la siguiente estrategia de ajuste:

- Entrenamiento no supervisado de los autoencoders de RGB y de profundidad (Fase 1). La optimización de sus híper-parámetros no ha de ser exhaustiva pero sí lo suficientemente extensa para que los autoencoders ofrezcan unos buenos resultados, ya que estos híper-parámetros optimizados en la reconstrucción de los datos no tienen por qué coincidir con los de la red final.
- Entrenamiento de la capa de conexión (Fase 2), y optimización de sus híper-parámetros en M3, ya que en M1 y M2 no dispone de ellos al unir directamente las características RGB con las características de profundidad.
- Fine-tuning de la red final (Fase 3), y ajuste de los híper-parámetros de toda la red probando un limitado conjunto de valores debido al mayor coste computacional. Se parte de los valores de híper-parámetros suministrados por los pasos anteriores, menos costosos de optimizar, obteniendo así una semilla de híper-parámetros que han funcionado bien en las anteriores Fases del entrenamiento.

Esta metodología puede verse variada según los análisis de resultados de cada Modelo, con el fin de aprovechar los recursos disponibles en aquellas pruebas y modelos con mayor interés.

4.5 Función de coste

Para todos los Modelos, se ha seleccionado la función de coste $F()$ euclídea, es decir, la suma cuadrática de los errores de las predicciones. El motivo es que esta es la única función de coste implementada en el software Caffe [23] destinada a tareas de regresión. Es por ello que la mayoría de las métricas mostradas en el trabajo se muestran en errores cuadráticos medios, ya sean en forma de metros cuadrados o de intensidades RGB [1-255] al cuadrado.

$$F(W) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T}\|{x^1}_i - {x^2}_i\|^2$$

Donde F es la función de coste; N es el número de datoso imágenes; T es el número de píxeles por imagen; ${x^1}_i$ es la matriz de profundidad verdadera del ejemplo i; ${x^2}_i$ es la matriz de profundidad computada por la red neuronal del ejemplo i; y W son los parámetros del modelo.

4.6 Factor de escala

Al estar empleando en todas las capas activaciones sigmoidales logísticas (Figura 2.1), los datos se deben escalar de 0 a 1 ya que este es el rango de la función sigmoidal. En el caso de las imágenes RGB, esto se logra multiplicando los canales RGB por el factor 1/255.

El escalado de las imágenes de profundidad no posee solución tan directa ya que no existe un límite superior. Un factor demasiado bajo provoca que la profundidad de demasiados píxeles quede fuera de rango, y por tanto con imposibilidad de predecir su verdadero valor de profundidad. Por el contrario, un factor demasiado elevado hace que los datos de entrada de la red tomen valores reducidos, favoreciendo una predicción constante que de valores bajos de profundidad. El error sería pequeño al computarse en el entrenamiento, pero al trasladarlo a la unidad de longitud original (metros), dicho error tomará altos valores debido al elevado factor de escala. Al tratarse de un problema del intervalo de valores de profundidad que puede reconstruir el autoencoder, y no a una cuestión de patrones locales o globales de la profundidad, este factor es ajustado solo en M1 (ver apartado 5.1.1), el primero de los modelos en ser entrenado.

5 RESULTADOS

5.1 Entrenamiento y optimización de híper-parámetros

5.1.1 M1: Local y sin aprendizaje profundo

Se procede con la Fase 1 del entrenamiento. Las ventanas RGB de entrada a este Modelo contienen 60·60·3 = 10.800 variables. En este caso una compresión a 1.000 unidades o neuronas (~10 a 1) con una ponderación del término de regularización λ = 0,0005 consigue dar buenos resultados en el autoencoder (ver Figura 5.2). Estos valores fueron tomados del ejemplo ''MNIST autoencoder" del software Caffe, donde se utiliza el mismo valor de λ, y una compresión aproximada de 26/1 en cuatro capas (en este modelo se realiza en una única capa).

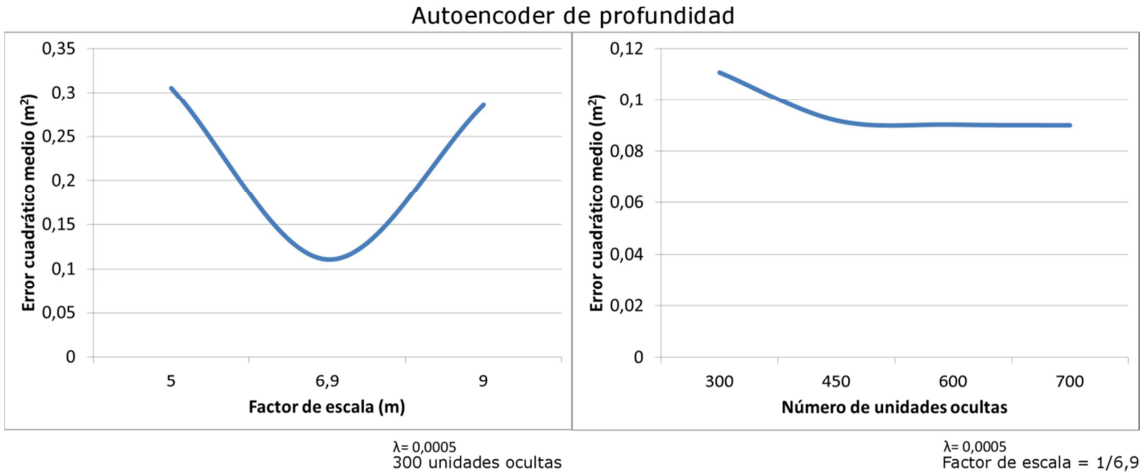


Figura 5.1: Gráficas del error cuadrático medio del autoencoder de profundidad según el factor de escala (izda.) o el número de unidades de la capa oculta (dcha.)

Como se indica en el apartado 4.6, el factor de escala las imágenes de profundidad se debe ajustar al no poseer una solución directa como el factor asociado a las imágenes RGB. Se testearon tres valores para el factor de escala de la profundidad:

- 5 metros debido a que a partir de ellos las medidas de la kinect suelen perder fiabilidad.
- 6,9 metros, al corresponderse con la media de profundidad del dataset más tres veces la desviación típica.
- 9 metros, aproximadamente la máxima profundidad encontrada en el dataset.

Como se observa en la Figura 5.1, el escalado de 6,9 metros es el que mejor resultados ofrece en el conjunto de validación, siendo el escogido para los tres modelos.

La compresión de las imágenes de profundidad se realiza desde 60·60·1 = 3.600 variables a 450 neuronas (ratio 8 a 1). La compresión es menor con respecto a las imágenes RGB, ya que el ratio 10 a 1 da aquí peores resultados, especialmente en la reconstrucción de discontinuidades en la profundidad. Como se puede observar en la Figura 5.1, el 'codo' de la curva del error comienza en las 450 unidades, lo que justifica dicha elección. Aun así, sigue presentando ciertos problemas a la hora de reconstruir los saltos bruscos de profundidad. Ello se debe posiblemente a la mayor cantidad en los datos de píxeles donde la profundidad es

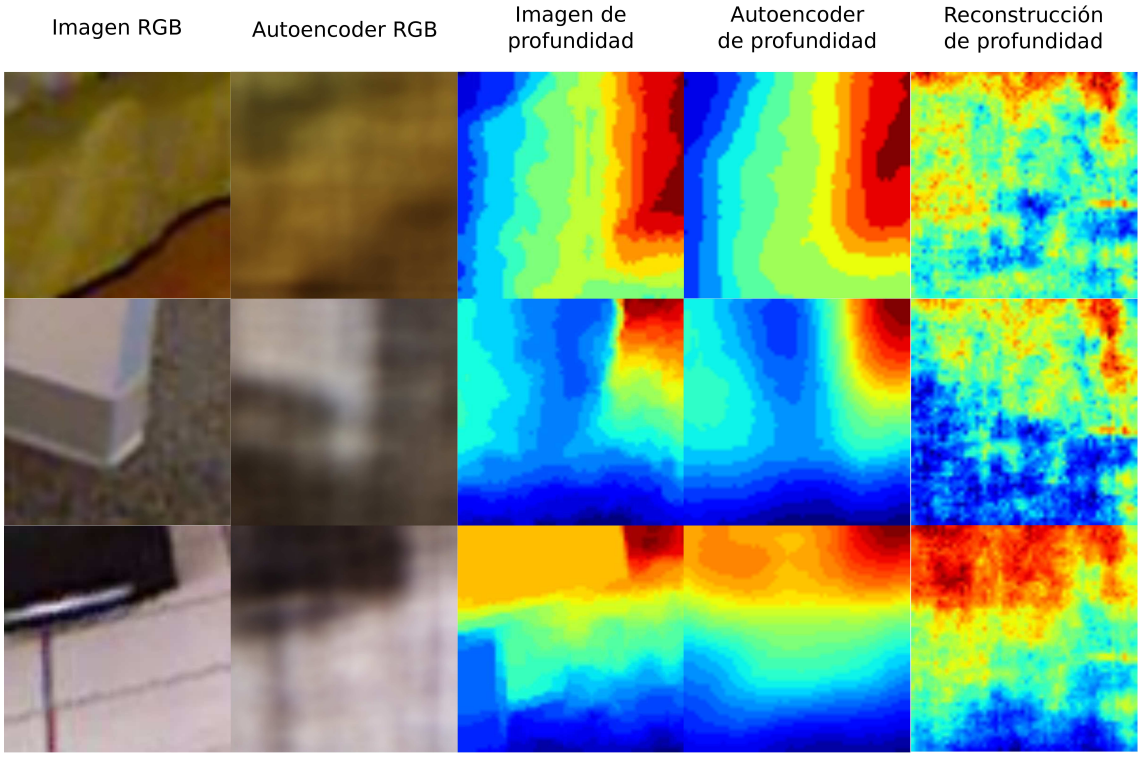


Figura 5.2. Resultados de M1 en el 'labeled dataset', seleccionados aleatoriamente. De izda. a dcha., imagen RGB original, reconstrucción del autoencoder RGB, imagen de profundidad original, reconstrucción del autoencoder de profundidad, y predicción de la red de la profundidad a partir del RGB.

continua o posee pequeños saltos sobre aquellos en los que existe una gran discontinuidad de profundidad, que puede provocar una tendencia a la continuidad en las reconstrucciones.

Se procede con las Fases 2 y 3 del entrenamiento, empleando los mismos hiper-parámetros de los autoencoders de la Fase 1. Para comprobar la efectividad de la Fase 2, se procede también saltando este paso, iniciando el fine-tuning (Fase 3) con la capa de conexión inicializada de forma aleatoria.

Los resultados de esta red (Tabla 5.1) no son satisfactorios, ya que la predicción de la misma profundidad siempre (profundidad media del conjunto de entrenamiento, que se corresponde a la última fila de la Tabla 5.1) da similares resultados, o incluso mejores cuando no se aplica la Fase 2 del entrenamiento. En el conjunto de entrenamiento sí que consigue batir dicha predicción media, pero aquello que aprende no es generalizable en el conjunto de validación.

En un análisis cualitativo, la red intenta casi exclusivamente ajustar la profundidad media de la ventana, sin tratar de obtener su forma (Figura 5.2). Esto es debido al error cuadrático al que está sometido el coste de la red: si se acierta aproximadamente la profundidad media de la ventana aunque no su forma, el error es aceptable; pero si se predice mal su profundidad media aunque se infiera su forma correcta, el error resulta muy elevado. Sin embargo, aquello aprendido en el conjunto de entrenamiento no es generalizable. Al no existir un elevado sobreajuste (el error en conjunto de validación no llega al triple que en el conjunto de entrenamiento) que justifique los malos resultados, se concluye que no merece la pena seguir con este Modelo.

	Error cuadrático medio [m^2]	
	Conjunto de entrenamiento	Conjunto de validación
Capa de conexión inicializada aleatoriamente (sin Fase 2)	0,937	2,527
Capa de conexión pre-entrenada (con Fase 2)	0,911	2,466
Error profundidad media	2,494	

Tabla 5.1: Resultados del Modelo 1 en el 'labeled dataset' de NYU

Por tanto, la conclusión de este Modelo es que no se puede predecir la profundidad de una imagen a partir de pequeñas ventanas por separado, o al menos el Modelo 1 es incapaz de hacerlo. Algo que quizá era de esperar, ya que ni siquiera nosotros podemos predecir la profundidad de una ventana de una imagen (a no ser que por fortuna aparezca un objeto que reconozcamos y podamos estimar su escala: monitor, silla, etc.). Ambos autoencoders son capaces de realizar su tarea de compresión-descompresión, pero no resulta posible unir la información que ambos proporcionan.

5.1.2 M2: Global y sin aprendizaje profundo

Los autoencoders de este modelo obtienen resultados satisfactorios (Figura 5.3) al ser entrenados con los mismos híper-parámetros que los del Modelo 1 en la Fase 1, por lo que se procede directamente a las siguientes Fases del entrenamiento sin intentar una mayor optimización.

Tras ejecutar las Fases 2 y 3 del entrenamiento, se puede observar en la Tabla 5.2 el error cuadrático se ve reducido en torno a un 30% con respecto al Modelo 1 para $\lambda = 0{,}0005$; es decir, empleando los mismos híper-parámetros que el Modelo 1. Pero el error de la predicción constante de la profundidad media del conjunto de entrenamiento se ve todavía más reducido, obteniendo este modelo un error cuadrático superior al de la profundidad media. Ello implica que aunque se haya mejorado la precisión con respecto al Modelo 1, también ha disminuido la dificultad de la relación entrada-salida al pasar de las ventanas del Modelo 1 a las imágenes subsampleadas. Esto resulta intuitivo si pensamos en patrones de profundidad globales de la imagen como suelos, paredes, techos, etc. que en este caso se repiten continuamente, mientras que dichos patrones globales no aparecen en el caso del Modelo 1.

	Error cuadrático medio [m^2]		Término de regularización	Dropout ratio
	Conjunto de entrenamiento	Conjunto de validación		
Sin Fase 3	0,1637	1,8257	0,0005	-
Tras Fase 3	0,1240	1,8836		
Sin Fase 3	0,6351	1,3598	0,005	-
Tras Fase 3	0,3267	1,3718		
Tras Fase 3	0,3459	1,4798	Distintos en cada capa	-
Sin Fase 3	0,5933	1,3437	0,0005	0,5
Tras Fase 3	0,4721	1,3911		
Error profundidad media	1,7670			

Tabla 5.2: Resultados de las variantes del Modelo 2 en el 'labeled dataset' de NYU.

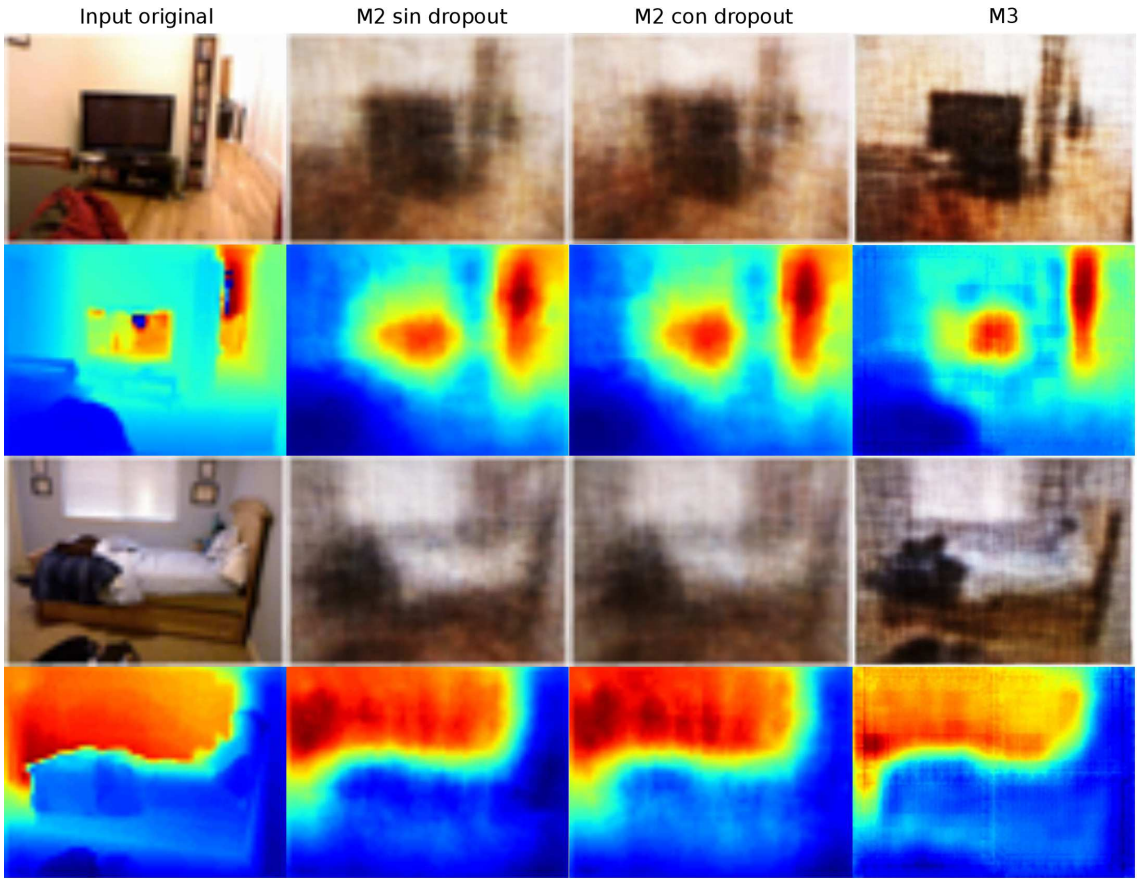


Figura 5.3. Resultados seleccionados aleatoriamente de los autoencoders en el 'labeled dataset' de distintos modelos. Los asociados a M2 se corresponden con los marcados en verde en la Tabla 5.2, y el asociado a M3 se corresponde con el seleccionado en el apartado 5.1.3.

Sin embargo, sí que aparece un elevado sobreajuste: el error en el conjunto de validación llega a ser hasta 15,19 veces mayor que el error en el conjunto de aprendizaje. Al incrementar el término regularizador un orden de magnitud hasta $\lambda = 0{,}005$ en todas las capas de la red y en todo el proceso de entrenamiento no supervisado, el error cuadrático disminuye un 25,5% más. Se obtiene finalmente el primer modelo capaz de superar a la predicción constante de la media de profundidad.

Mientras que ambos autoencoders funcionan correctamente en sus respectivas tareas, al construir la red neuronal final aparece el sobreajuste, especialmente tras el fine-tuning (Fase 3). Al incrementar el valor de λ tan solo en la capa de conexión en esta última Fase, quizá se logre remediar el sobreajuste sin suavizar en exceso la solución. En la Tabla 5.2 aparece indicado como con un término de regularización "Distintos en cada capa", donde se obtienen unas métricas peores, quedando este método descartado.

Otra medida estudiada es la de combinar la regularización el dropout para combatir el sobreajuste. Al aplicar un ratio de dropout igual a 0,5 en todas las capas de la red, se puede disminuir el parámetro regularizador a 0,0005 y mejorar ligeramente las métricas respecto al caso anterior, confirmando la efectividad del dropout para combatir el sobreajuste. Aun así, cualitativamente las reconstrucciones siguen teniendo un suavizado similar, aunque con una

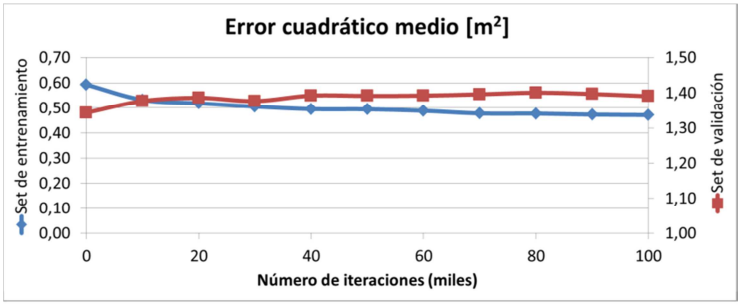


Figura 5.4: Evolución del error cuadrático medio en el proceso de fine-tuning (Fase 3) de la versión del Modelo 2 con dropout en los conjuntos o sets de entrenamiento y validación.

métrica ligeramente mejor.

En este punto, el algoritmo es capaz de reconocer los puntos de fuga en las imágenes, y orientar por tanto el gradiente de profundidad en dicha dirección (Figura 5.5). Sin embargo, solo lo hace cuando las líneas de fuga son claras y numerosas. Además, apenas reconoce grandes objetos como mesas o camas, ya que no se reflejan en el mapa de profundidad predicho. A pesar de obtener resultados satisfactorios y mejorar a algoritmos no basados en redes neuronales, dicho modelo está todavía lejos de las métricas obtenidas en [6], el cual emplea el aprendizaje profundo de las redes neuronales, aunque dispone del orden de millones de imágenes extraídas del dataset NYU Depth Dataset [23].

Resulta contraintuitivo que al ejecutar el fine-tuning de la red completa, el error en el conjunto de validación aumente. Ello implica que la semilla otorgada por la inicialización de los autoencoders (y la capa de conexión) da realmente buenos resultados y la intuición de que las características RGB y las características de profundidad están relacionadas queda demostrada para este modelo. Confirma también la hipótesis de [15]. En este caso, al intentar mejorar al modelo optimizando según el gradiente descendiente dejando cambiar todas las capas, aquello que se aprende no generaliza y acaba poseyendo un mayor error en el conjunto de validación. En la Figura 5.4 se muestra la evolución del error del Modelo 2 con dropout en el conjunto de entrenamiento y en el de validación a lo largo del proceso de fine-tuning.

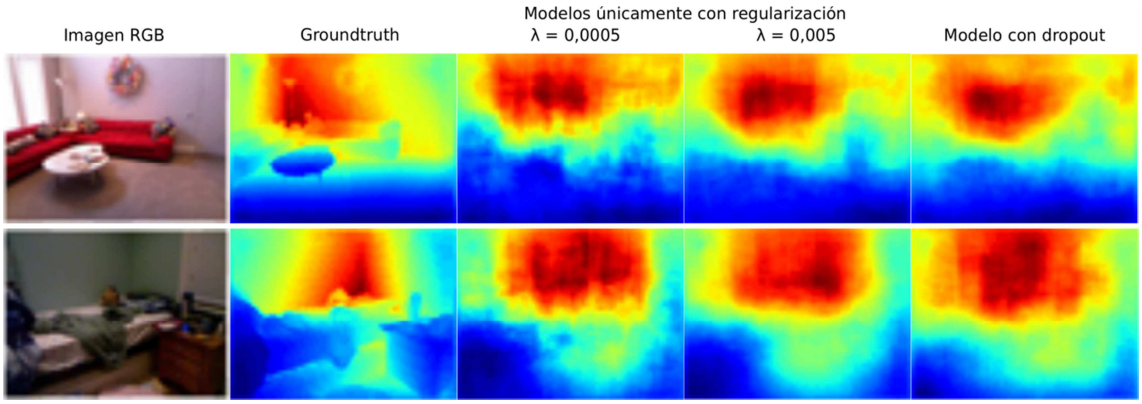


Figura 5.5. Imágenes de las predicciones de distintas variantes del Modelo 2 sobre dos ejemplos del 'labeled dataset' de NYU. Han sido seleccionadas de modo que posean objetos voluminosos y un punto de fuga, para observar el acierto de su reconstrucción.

5.1.3 M3: Global y con aprendizaje profundo

Se repite de nuevo el entrenamiento no supervisado de los autoencoders. El número de características por capa y los 'strides' de cada capa han sido elegidos por similitud con [6] (Figura 3.6). Solo resta ajustar el término regularizador λ.

Tan solo se testean dos valores para este híper-parámetro, debido de nuevo a la lentitud del entrenamiento y a causas explicadas en el apartado 4.4. Los resultados de los autoencoders, tanto el de una capa de compresión como el de dos (el segundo se construye a partir del primero), se muestran en la Tabla 5.3. Se observa que en ambos resulta mejor el menor valor de λ cuando se entrena la primera capa, mientras que sucede al contrario cuando se entrenan completos con las dos capas. Esto se justifica con el aumento de parámetros al incluir las dos capas. Al no incrementarse el tamaño del conjunto de datos empleado, el aumento del número de parámetros tiende a acrecentar el fenómeno de sobreajuste, por lo que resulta más eficaz el modelo con mayor término de regularización.

Ambos autoencoders de dos capas así construidos ofrecen mejores resultados que los asociados a los modelos anteriores (Figura 5.3, entrenados en 'labeled dataset', pero los resultados son idénticos a los entrenados en el 'extension dataset'). Sin embargo, ello no implica que estas nuevas características de los autoencoders sean útiles para la inferencia de la profundidad.

Error cuadrático medio

		Capas del autoencoder		
	λ	1	2	
RGB	0,005	256 → 518		$Intensidad^2$
	0,00005	251 → 524		RGB
Profundidad	0,005	0,04660 → 0,06289		m^2
	0,00005	0,04542 → 0,08406		

Tabla 5.3: Métricas de los autoencoders de una y dos capas de M3 en el conjunto de validación del 'extension dataset'.

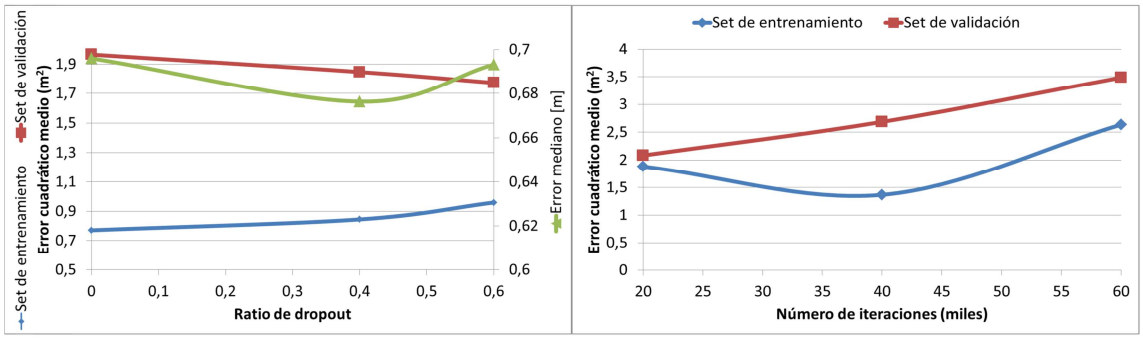


Figura 5.6: A la izda., errores cuadráticos en el conjunto o set de entrenamiento y validación y error mediano en el conjunto o set de validación para el entrenamiento de la capa de conexión de M3 según se varía el ratio de dropout. A la dcha., evolución del error cuadrático al entrenar M3 sin ningún tipo de pre-entrenamiento, con inicialización aleatoria. Resultados asociados al 'extension dataset'.

En la Fase 2, la capa de conexión ha sido entrenada con y sin dropout, debido a que en el Modelo 2 aparece sobreajuste en el entrenamiento de esta capa y esta técnica mejora ligeramente su funcionamiento. En la Figura 5.6 se observan los resultados sin dropout, y con un dropout ratio igual a 0,4 y 0,6. Se escoge finalmente un ratio igual a 0,6 debido a que presenta una disminución de un 4% en el error cuadrático, frente a un aumento del 2% en el error mediano con respecto al ratio igual a 0,4.

Resta ejecutar el fine-tuning a la red final (Fase 3). En este caso se dispone de cinco capas ocultas, por lo que es previsible que el efecto de difusión de gradientes a lo largo de la red tenga un efecto notorio en el entrenamiento. Para remediar este potencial problema se propone el siguiente método: dado que los gradientes reducen su valor absoluto a lo largo de la propagación de gradientes, el factor de aprendizaje de cada capa será ajustado de modo que las actualizaciones tomen valores similares en todas las capas de la red. Ello implica a su vez decrementar la ponderación del término de regularización λ, ya que en caso contrario su efecto crecerá conforme descendemos en la red ya que el factor de aprendizaje aumenta. Otra opción en cuanto al término de regularización, empleada en [8], es aplicarlo tan solo en la última capa, evitando que las primeras capas atiendan principalmente a este factor debido a la disminución del gradiente de la función de coste.

Sin embargo, se desconoce cuál es el valor óptimo por el que aumentar el factor de aprendizaje. Por ello se realiza una prueba en la que se ejecuta el fine-tuning hasta diez mil iteraciones de la red en distintos escenarios:

- Se mantiene el factor de aprendizaje constante en toda la red. Valor multiplicador unidad.
- En cada capa el factor de aprendizaje se multiplica por 2 hacia atrás. Valor multiplicador de valor 2.
- En cada capa el factor de aprendizaje se multiplica por 5 hacia atrás. Valor multiplicador de valor 5.

Además, cada uno de estos tres casos se desglosa en otros dos: se ajusta el término regularizador conforme la variación del factor de aprendizaje (señalado como 'Ajustado' en la Tabla 5.4), o tan solo se aplica el término regularizador en la última capa. Los resultados se reflejan en la Tabla 5.4. En este caso el fine-tuning sí que es efectivo, no como en el Modelo 2, al menos para los dos primeros valores multiplicadores. Se obtienen mejores resultados cuando solo se aplica el término regularizador a la última capa, aunque sea por una diferencia que no llega al 1%. Los resultados asociados a los valores multiplicadores 1 y 2 son muy similares. Mientras que el primero presenta una ventaja inferior al 1% en el error cuadrático, el segundo posee un error mediano menor en un 11%. Al tratarse de la decisión final, se puede costear continuar con el entrenamiento de ambos métodos hasta las noventa mil iteraciones con el fin de asegurar la decisión más adecuada.

		10k iteraciones		90k iteraciones	
Valor multiplicador	Término regularizador	Error cuadrático medio [m^2] Conjunto de validación	Error mediano [m]	Error cuadrático medio [m2] Conjunto de validación	Error mediano [m]
1	Ajustado	1,681	0,6966		
	Última capa	1,679	0,6903	1,750	0,6188
2	Ajustado	1,711	0,6208		
	Última capa	1,687	0,6205	1,567	0,6046
5	Ajustado	2,076	0,7629		
	Última capa	2,072	0,7650		
				30k iteraciones sin fine-tuning (Paso 3)	
Modelo M3.1				1,938	0,6892

Tabla 5.4: Resultados de M3 aplicando distintos métodos para su entrenamiento, y de M3.1. Resultados asociados al 'extension dataset'

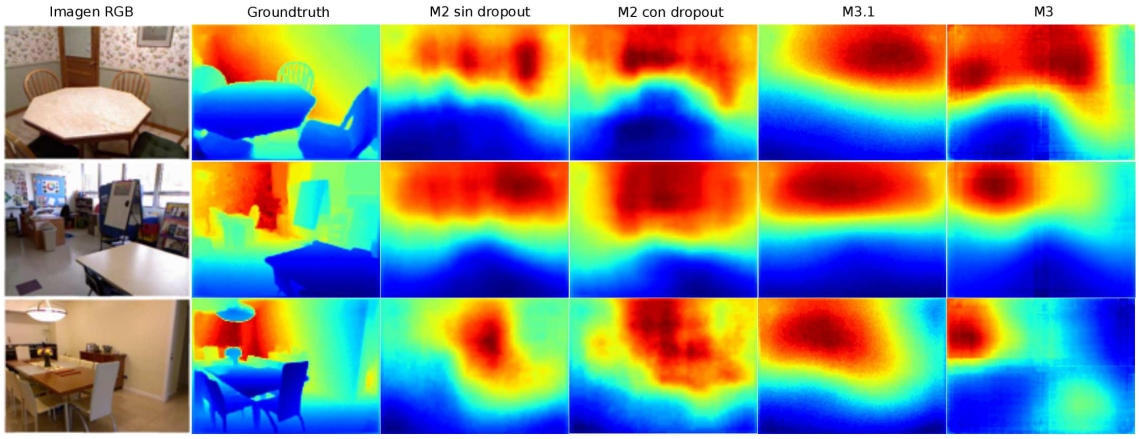


Figura 5.7. Predicciones de los modelos asociados a M2 marcados en verde en la Tabla 5.2, a M3.1 y M3 entrenado según el apartado 5.1.3, todos ellos entrenados en el 'extension dataset'. Imágenes seleccionadas de modo que posean objetos voluminosos y un punto de fuga, para observar el acierto de las reconstrucciones.

Las métricas en el Modelo entrenado hasta las noventa mil iteraciones muestran que el modelo con un valor multiplicador de dos supera al que tiene la unidad. Ello demuestra la eficacia de este método para combatir la difusión de gradientes, de modo que no solo sean las últimas capas las que se beneficien del proceso del fine-tuning, sino que lo haga la red al completo. En este caso, el error de validación asociado al valor multiplicador unidad es mayor a las 90k iteraciones que a las 10k, de modo similar a lo que sucede en el fine-tuning del Modelo 2. Sin embargo el error mediano sí que disminuye. Ello implica que aquellos errores que eran pequeños-medianos a las 10k iteraciones, siguen disminuyendo y provoca una fuerte bajada en el error mediano de validación. Sin embargo, aquellos que eran grandes, continúan aumentando, provocando que el error cuadrático se eleve.

Las reconstrucciones del Modelo 3 así entrenado (valor multiplicador dos, y término regularizador solo en la última capa) se muestran en la Figura 5.7. Estas reconstrucciones superan a las del Modelo 2. Son capaces de colocar los puntos de fuga de forma más precisa. Además, los objetos voluminosos como las mesas se reflejan de forma más clara en el mapa de profundidad.

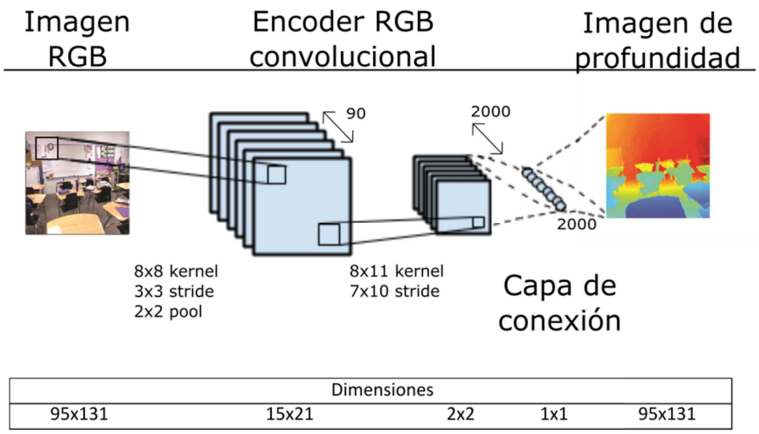


Figura 5.8. Esquema de la arquitectura del Modelo 3.1

A pesar de que el Modelo 3 se encuentra optimizado y entrenado, se realizan dos experimentos adicionales con el fin de obtener un conocimiento más minucioso del funcionamiento de las redes neuronales.

En el primero de ellos, se repite el entrenamiento de exactamente la misma red, con mismos híper-parámetros y valor multiplicador a través de las capas. Sin embargo, esta vez no se le aplica ningún pre-entrenamiento, inicializando todas las capas de forma aleatoria y saltando directamente a la Fase 3 del entrenamiento. La evolución del error se muestra en la Figura 5.6 tanto para el conjunto de entrenamiento como para el de validación. El error a las 60k iteraciones en el conjunto de entrenamiento es mayor que a las 20k. Como reflejan los resultados de [15], a partir de cinco capas ocultas las redes comienzan a fallar en la convergencia, ya que ni siquiera el error de aprendizaje disminuye. Esto vuelve a confirmar la necesidad de aplicar pre-entrenamiento a las redes neuronales profundas.

En otro experimento se construyó una red con una estructura similar a la del Modelo 3. Como se explica en el apartado 3, en tareas de clasificación es común aplicar varias capas convolucionales y finalmente una capa totalmente conectada para obtener la inferencia de la red, de modo similar a [6] en su modelo 'coarse'. Aquí se implementa una estructura similar, donde se aplica el mismo autoencoder RGB pre-entrenado que en el Modelo 3 y la misma capa de conexión (también con un ratio de dropout = 0,6). La diferencia reside en obtener la imagen de profundidad a partir de una única capa totalmente conectada, en vez de las capas deconvolucionales del decoder de profundidad. Se aplicaron los mismos híper-parámetros que en el Modelo 3. Los mejores resultados se obtuvieron sin realizar la Fase 3. Al realizar el fine-tuning las métricas empeoraban considerablemente. Los resultados aparecen en la Tabla 5.4, indicados como "Modelo M3.1". La métrica obtenida resulta en torno a un 25% superior a la del Modelo 3, mostrando la eficacia del pre-entrenamiento no supervisado de las últimas capas. Sin embargo, hay que tener en cuenta que esta nueva red dispone de tres capas ocultas en vez de las cinco del M3, por lo que parte de este peor resultado podría achacarse a esta pérdida de número de capas.

5.2 Comparación con el estado del arte

		Labeled dataset		Extension dataset	
		Error cuadrático medio [m^2]	RMSE lineal [m]	Error cuadrático medio [m^2]	RMSE lineal [m]
M1		2,471	1,317	-	-
M2		1,364	1,048	1,854	1,193
M3		1,322	1,045	1,662	1,101
Profundidad media		1,767	1,244		
Make3D		-	1,214		
Karsch&al		-	1,2		
Eigen&al		-	0,871		
Liu&al	Sin CRF	-	0,985		
	Con CRF	-	0,824		

Tabla 5.5: Comparación en los conjuntos de test de los tres Modelos con el estado del arte.

En la Tabla 5.5, se muestran las métricas en el conjunto de test de los tres Modelos propuestos[2], tanto en el 'labeled dataset' como en el 'extension dataset'. Aparece además del error cuadrático medio, el error de la raíz cuadrada de la media ('Root-Mean-Squared Deviation', RMSE, ver Anexo C), ya que es una métrica comúnmente empleada por la comunidad científica para comparar resultados.

Hay dos aspectos interesantes en los resultados. En primer lugar, M3 ofrece los mejores resultados en ambos datasets. Mientras que en el 'extension dataset' mejora en torno a un 9% a M2, esta mejora apenas llega en el 'labeled dataset' a un 3% en el error cuadrático medio y prácticamente no mejora el RMSE. Esto se explica con la menor cantidad de datos del 'labeled dataset' y la facilidad que tienen las redes neuronales profundas a sufrir de sobreajuste, especialmente cuando la cantidad de datos es pequeña. Los resultados de M3 en el 'labeled dataset' se obtienen tras finalizar la Fase 2, ya que al realizar la Fase 3 (fine-tuning) el sobreajuste aumenta todavía más llegando hasta 1,531 m^2 de error cuadrático medio, demostrando de nuevo el buen funcionamiento del pre-entrenamiento [15], donde incluso la semilla dada por este da mejores resultados que la red final entrenada a partir de esta. El error en el conjunto de entrenamiento llega a ser hasta diez veces menor que el de test.

En segundo lugar, las métricas de los modelos empeoran en el 'extension dataset' con respecto al 'labeled dataset', especialmente en el caso de M2. Esto resulta llamativo, ya que una mayor cantidad de información debería resultar en un mejor funcionamiento del algoritmo. Sin embargo, hay que percatarse de algo: el denominado 'labeled dataset' se encuentra destinado principalmente a tareas de segmentación de interiores. Ello hace que del dataset completo NYU v2 se escojan imágenes en las que es común que aparezca toda la habitación en perspectiva, tomadas usualmente desde una esquina de la misma. Ello provoca

[2] El M1 no se llegó a entrenar en el 'extension dataset' debido a limitaciones de memoria de Python. Sin embargo, debido a que ya se extrajeron 14.500 ventanas del 'labeled dataset', la ganancia de datos no habría sido significativa. Tampoco lo habría sido la mayor dificultad del 'extension dataset', al entrenarse sobre ventanas locales donde la localización de la cámara no es tan importante. Esto, junto al mal funcionamiento en el 'labeled dataset', desestimó la búsqueda de alternativas para conseguir entrenar M1 en el 'extension dataset'.

que la tarea de predecir la profundidad sea algo más simple ya que en muchas de ellas la localización de la cámara con respecto a la habitación es similar, disminuyendo la variedad de patrones de profundidad de la escena con respecto a si estas son tomadas de forma continua a partir de los vídeos disponibles de NYU v2 como hace el 'extension dataset'.

En comparación con el estado del arte, los modelos no consiguen los resultados de [6] o [7], aunque sí consiguen batir a modelos no basados en aprendizaje profundo, como [3] o [4], un buen resultado especialmente para M2 ya que no es una RNA de aprendizaje profundo. Hay que mencionar que en [6] se emplea la totalidad del NYU Dataset, al que se le aplican transformaciones para aumentar artificialmente los datos, llegando hasta las 2M de imágenes en el conjunto de entrenamiento. En cuanto a [7], M2 y M3 se sitúan muy cerca de su RNA de ocho capas ocultas sin CRF, un buen resultado considerando que M2 solo dispone de dos capas ocultas y que M3 dispone de cinco y sus híper-parámetros no han sido ajustados en el 'labeled dataset'. Al aplicar el CRF en la RNA de [7], la métrica mejora considerablemente. Sin embargo, el CRF es algo adicional a la estructura de la RNA y también podría ser aplicado tanto a M2 como M3, pudiendo obtener una mejoría similar.

6 CONCLUSIONES

En el presente proyecto se ha realizado un estudio de investigación con el fin de resolver el problema de reconstrucción tridimensional a partir de una imagen, empleando como herramienta las redes neuronales artificiales con y sin aprendizaje profundo.

La predicción de la profundidad a partir de una sola imagen es un problema relevante, y con una gran variedad de aplicaciones asociadas a campos como modelado 3D, robótica, entendimiento de escenas e imágenes, etc.

A lo largo de este TFG se ha realizado un extenso trabajo. Comenzando por la revisión bibliográfica inicial; el estudio, comparativa y selección del dataset; la revisión de software y hardware disponible; y finalizando con el propio trabajo de investigación consistente en el desarrollo de los modelos de RNA para resolver el problema propuesto y su posterior análisis. Con los resultados obtenidos cercanos al estado del arte, los objetivos propuestos al comienzo del proyecto se pueden considerar cumplidos.

La mayor contribución de este proyecto es el pre-entrenamiento no supervisado aplicado a la salida del problema, ya que hasta ahora solo se aplicaba a la entrada del mismo. En los modelos diseñados, este ayuda al algoritmo a encontrar una solución con buena generalización, especialmente cuando la cantidad de datos empleada es pequeña y la cantidad de parámetros del modelo es elevada. En varias ocasiones, incluso la semilla obtenida por este pre-entrenamiento no supervisado, seguida de la Fase 2, da mejores resultados que el fine-tuning final de la red neuronal a partir de esta (Fase 3). El uso del pre-entrenamiento no supervisado sobre la salida es extensible a cualquier otra aplicación cuya salida tenga cierta complejidad, difundiendo el interés de este trabajo más allá del problema particular aquí analizado.

Para el diseño, implementación y análisis de los modelos, se han escrito del orden de decenas de 'scripts' en Python para trabajar con imágenes en el software Caffe Berkeley Vision [23]. Estos códigos son totalmente reutilizables, facilitando y agilizando futuros trabajos dentro del mismo ámbito.

Se ha tenido que realizar una extensa batería de pruebas para realizar los análisis incluidos en el proyecto, limitadas en varias ocasiones por los recursos disponibles, y por una serie de percances que retrasaron el proyecto: una intervención quirúrgica en el mes de marzo congeló el avance del proyecto durante un plazo aproximado de treinta días; y varios apagones durante el mes de Julio paralizaron los servicios informáticos cuando se procesaba el NYU Depth Dataset [18] por completo, con el fin de entrenar los Modelos propuestos en la totalidad del dataset de modo análogo a [6], obligando a reiniciar el proceso en repetidas ocasiones. Actualmente, el dataset se encuentra descargado y en mitad de un proceso de aumentación artificial de datos. Estos resultados estaban planeados para su inclusión en esta memoria, pero los retrasos sufridos han impedido que se hayan podido realizar a tiempo. Como consecuencia, durante los próximos meses se realizarán estas pruebas restantes para completar esta investigación.

7 BIBLIOGRAFÍA

[1] Neitra 3D Pro. Disponible en http://triayaam.com/index.php/neitra-3d-pro/

[2] Habbecke, M., & Kobbelt, L. (2008, June). Laser brush: a flexible device for 3D reconstruction of indoor scenes. En *Proceedings of the 2008 ACM symposium on Solid and physical modeling* (pp. 231-239). ACM. Disponible en https://www.graphics.rwth-aachen.de

[3] Saxena, A., Sun, M., & Ng, A. Y. (2007, October). Learning 3-d scene structure from a single still image. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on* (pp. 1-8). IEEE.

[4] Karsch, K., Liu, C., & Kang, S. B. (2012). Depth extraction from video using non-parametric sampling. En *Computer Vision–ECCV 2012* (pp. 775-788). Springer Berlin Heidelberg.

[5] Ladicky, L., Shi, J., & Pollefeys, M. (2014, June). Pulling things out of perspective. En *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on* (pp. 89-96). IEEE.

[6] Eigen, D., Puhrsch, C., & Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. En *Advances in Neural Information Processing Systems* (pp. 2366-2374).

[7] Liu, F., Shen, C., & Lin, G. (2014). Deep convolutional neural fields for depth estimation from a single image. *arXiv preprint arXiv:1411.6387*.

[8] Ng, A., Ngiam, J., Yu Foo, C., Mai, Y., & Suen, C. (2013). UFLDL Tutorial [online]. [fecha de consulta: Enero 2015]. Disponible en http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial

[9] Peng, L. (2014). Fully-connected, locally-connected and shared weights layer in neural networks [Mensaje en un blog]. Recuperado de https://pennlio.wordpress.com/2014/04/11/fully-connected-locally-connected-and-shared-weights-layer-in-neural-networks/

[10] Ng, A. (2014). Machine Learning, *Stanford AI Lab* [Tutorial online]. Recuperado de https://www.coursera.org/learn/machine-learning

[11] Montesano, L. (2013). "Tema 2. Regularización". Aprendizaje automático, *Universidad de Zaragoza* [Apuntes de asignatura].

[12] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929-1958.

[13] Michael A. Nielsen, "Neural Networks and Deep Learning", *Determination Press*, 2015

[14] Berniker M and Kording KP (2015) Deep networks for motor control functions. *Front. Comput. Neurosci*. 9:32. doi: 10.3389/fncom.2015.00032

[15] Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning?. *The Journal of Machine Learning Research*, *11*, 625-660.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, 86(11):2278-2324, November 1998.

[17] Lai, K., Bo, L., Ren, X., & Fox, D. (2011, May). A large-scale hierarchical multi-view rgb-d object dataset. En *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (pp. 1817-1824). IEEE.

[18] Silberman, N., Hoiem, D., Kohli, P., & Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. En *Computer Vision–ECCV 2012*(pp. 746-760). Springer Berlin Heidelberg.

[19] Xiao, J., Owens, A., & Torralba, A. (2013, December). SUN3D: A database of big spaces reconstructed using sfm and object labels. En *Computer Vision (ICCV), 2013 IEEE International Conference on* (pp. 1625-1632). IEEE.

[20] Sturm, J., Engelhard, N., Endres, F., Burgard, W., & Cremers, D. (2012, October). A benchmark for the evaluation of RGB-D SLAM systems. En *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (pp. 573-580). IEEE.

[21] Levin, A., Lischinski, D., & Weiss, Y. (2004, August). Colorization using optimization. En *ACM Transactions on Graphics (TOG)* (Vol. 23, No. 3, pp. 689-694). ACM.

[22] Nissen, S. (2003). Implementation of a fast artificial neural network library (fann). *Report, Department of Computer Science University of Copenhagen (DIKU)*, *31*

[23] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. & Darrell, T. (2014, November). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093.*

[24] Lopez, R. (2014). Open NN: An Open Source Neural Networks C++ Library [software]. Retrieved from www.cimne.com/flood

[25] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., ... & Bengio, Y. (2012). Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*.

[26] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., ... & Bengio, Y. (2010, June). Theano: a CPU and GPU math expression compiler. En *Proceedings of the Python for scientific computing conference (SciPy)* (Vol. 4, p. 3).

[27] Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., ... & Schmidhuber, J. (2010). PyBrain. *The Journal of Machine Learning Research*,*11*, 743-746.

8 ÍNDICE DE FIGURAS

9 ÍNDICE DE TABLAS