Universidad Zaragoza
1542

Trabajo Fin de Máster

Estimación de profundidad con redes neuronales profundas en vídeos de endoscopias

Depth estimation with deep neural networks in endoscopy videos

Autor

David Recasens Lafuente

Director

Javier Civera Sancho

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2020

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. David Recasens Lafuente,

con nº de DNI 17768014P 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) Máster, (Título del Trabajo) Estimación de profundidad con redes neuronales profundas en vídeos de endoscopias

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

Zaragoza, 17 de noviembre de 2020

RECASENS LAFUENTE DAVID - 17768014P

Firmado digitalmente por RECASENS LAFUENTE DAVID - 17768014P
Fecha: 2020.11.17 10:40:00 +01'00'

Fdo: David Recasens Lafuente

AGRADECIMIENTOS

Por encima de todo, agradecer a los mejores amigos que una persona puede tener: mis padres, Ana y Roberto, y mi hermana, Patricia. Ellos dieron forma a la persona que soy y son los pilares que me sustentan en los buenos y malos momentos. Sin su infinito apoyo, nada de esto habría sido posible.

Quiero agradecer a los amigos que conocí y que me siguen acompañado tras la universidad, con mención especial a Caravaca, Jorge Luís, Eva, Anzhela, a los *quimicefers* y a los amigos de toda la vida, César y Luismi.

Finalmente quiero decirle gracias a mi tutor Javier Civera y al resto de compañeros del proyecto EndoMapper porque han conseguido que estos 8 meses de trabajo los haya disfrutado.

RESUMEN

Ya lo promulgaba Isaac Asimov en 1964 con sus predicciones a 50 años: “Se pondrá mucho esfuerzo en el diseño de vehículos con cerebros robóticos”. Y así lo está siendo: sirva como ejemplo la industria automotriz, volcada desde hace unos años en la conducción autónoma. El principal protagonista de este gran revuelo es la inteligencia artificial y la revolución no se limita a este único sector, habiéndose demostrado muy útil en multitud de campos. Y entre ellos, uno de los más prometedores y desafiantes, el de la medicina.

Éste es el marco en el que se desarrolla este trabajo, que forma parte del proyecto Endomapper. Endomapper aspira a llevar la endoscopia al reino de la realidad virtual y aumentada, haciendo posibles nuevas interfaces que faciliten el trabajo del personal médico, y en un futuro más lejano la navegación autónoma, posibilitando que pequeños robots realicen tareas en el interior de nuestro cuerpo. Uno de sus pilares fundamentales es la creación de mapas tridimensionales en tiempo real a partir de vídeos monoculares de endoscopias. Para este cometido es vital contar con un sistema preciso y fiable que consiga extraer la profundidad píxel a píxel de las secuencias de entrada teniendo en cuenta las particularidades del problema: el entorno intracorpóreo carece de buena textura y su estructura se encuentra en constante deformación.

Existen avanzados sistemas de recontrucción 3D que utilizan modelos geométricos alimentados con varias imágenes de una misma escena en entornos no rígidos. Aquí, en su lugar, se ha apostado por modelos basados en aprendizaje profundo que únicamente requieren de imágenes independientes. Esta disruptora rama de la inteligencia artificial a veces promete resultados más propios de una novela de ciencia ficción. Sin quedarse en falsas promesas, en este trabajo se detalla cómo es capaz de proporcionar estimaciones de profundidad densas, precisas y robustas que mejoran el estado del arte en vídeos de endoscopias.

Índice

Capítulo 1

Introducción

1.1. Motivación

Con 18,1 millones de casos nuevos en 2018, el cáncer continúa rondando los primeros puestos de morbi-mortalidad (muerte causada por enfermedades) a nivel mundial, y la tendencia sigue en aumento [1]. Para 2040 se prevé un aumento de casos del 63.1 %. Los tumores colorectales, en concreto, son los terceros más comunes (el 10.2 % del total en 2018). La transformación de un tumor benigno a uno maligno puede durar años, y la detección temprana resulta crucial para tratarlos a tiempo. Por ello, el número de colonoscopias es actualmente muy elevado y se espera que la demanda se duplique entre 2016 y 2036 [2].

Los endoscopistas trabajan habitualmente a contrarreloj en procedimientos que requieren de una competencia excelente y, como consecuencia, en torno al 17 % de los pólipos colorectales pasan desapercibidos [3]. Disponer de asistencia computacional para comprobar si el colon ha sido observado en su totalidad reduciría este porcentaje drásticamente. Sería por ejemplo de gran utilidad que los especialistas tuvieran el apoyo de interfaces de realidad aumentada. Una posibilidad podría consistir, a modo de ejemplo, en un sistema de guía auxiliar que le mostrara qué regiones han sido revisadas y cuáles no, indicándole además el lugar del colon en el que se encuentra en cada momento. Para ello se necesitan algoritmos capaces de mapear tridimensionalmente el colon y auto-localizarse dentro de él en tiempo real. Por si fuera poco, sería ideal que dichos algoritmos utilizasen únicamente el vídeo capturado por un endoscopio monocular estándar (una sola cámara), para que así pueda ser implementado en cualquier centro médico del mundo. A la tecnología que aspira a la creación de mapas y autolocalización a partir de imágenes se le conoce como VSLAM (del inglés "Visual Simultaneous Localization and Mapping") [4]; y este trabajo se enmarca en su aplicación concreta en endoscopias.

El primer sistema de VSLAM monocular vio la luz en 2003 [6]. Desde entonces

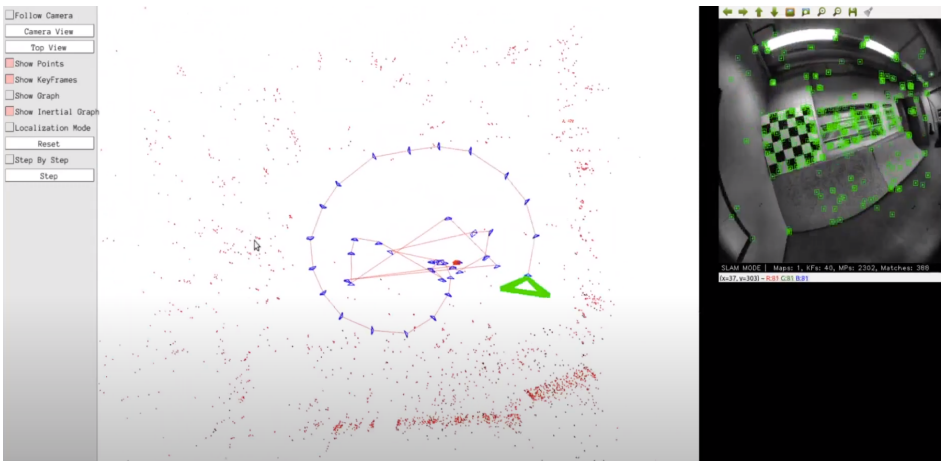

Figura 1.1: Reconstrucción 3D de la escena y trayectoria de la cámara a partir de puntos característicos (puntos rojos). Fuente: ORB-SLAM3 [5].

ha habido un gran esfuerzo investigador y los algoritmos han mejorado notablemente. Pero casi todos ellos asumen un entorno rígido. En la figura 1.1 puede verse como uno de estos sistemas es capaz de estimar la trayectoria de una cámara y crear una reconstrucción de puntos con elevada precisión y robustez. Sin embargo, el interior de los seres vivos no es rígido y los sistemas de VSLAM presentan un desempeño pobre en estos entornos. DefSLAM [7] y SD-DefSLAM [8] son pioneros en este sentido al proponer un avanzado sistema de SLAM visual en entornos intracorpóreos deformables (figura 1.2). Pero, debido a la dificultad del problema, su desempeño se encuentra todavía sustancialmente por detrás de los sistemas que funcionan en entornos rígidos.

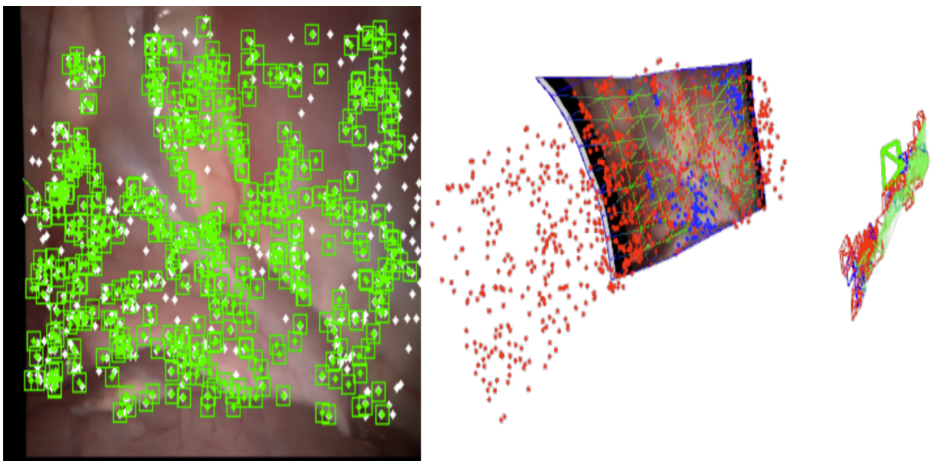

Figura 1.2: SD-DefSLAM trabajando en un entorno intracorpóreo. Izquierda: puntos característicos en la imagen endoscópica. Derecha: movimiento de la cámara y mapa deformable estimado. Fuente: SD-DefSLAM [8].

Mejorar los sistemas de VSLAM para que se ejecuten eficaz y eficientemente en vídeos de endoscopias es precisamente el objetivo del proyecto EndoMapper [9], en el que se integra el presente trabajo. Al abordar este reto, se observa que la estimación

de profundidad a partir de un vídeo monocular es un aspecto fundamental. Al tratarse de endoscopias, a la no-rigidez de las estructuras internas se le suma la falta de textura de los tejidos orgánicos, la inestabilidad de la cámara durante la grabación y la pobre iluminación. El sistema SD-DefSLAM, anteriormente mencionado, trata de abordar estas particularidades con modelos geométricos que requieren de varias imágenes de una misma escena en entornos no rígidos. Esta tecnología ofrece buenos resultados, pero presenta varias limitaciones:

- Profundidad dispersa: los mapas no son densos, y están compuestos únicamente por unos cientos de puntos esparcidos por la imagen.
- Falta de robustez: el software no es capaz de ejecutarse satisfactoriamente en todo tipo de secuencias.
- Precisión moderada: las profundidades estimadas tienen a veces errores considerables.
- Mapas planos: DefSLAM y SD-DefSLAM parten de una malla plana que adaptan a la superficie estimada, causando que sus mapas no capten correctamente las grandes diferencias de profundidad.

Para paliar estas limitaciones, en este trabajo haremos uso del aprendizaje profundo [10]. El aprendizaje profundo tiene la capacidad de extraer a partir de datos de entrenamiento patrones de muy alto nivel que a un humano le resultarían evidentes, pero complejas de describir formalmente con reglas matemáticas (como es la percepción de profundidad al ver una fotografía). Esta capacidad le ha permitido mejorar dramáticamente el estado del arte en muchos campos de la inteligencia artificial como la detección de objetos, el reconocimiento por voz o la traducción automática, entre otros [11]. Ligando con la tarea que aquí nos concierne, ha demostrado un desempeño excelente en varias tareas relacionadas con la reconstrucción 3D a partir de imágenes (por ejemplo, estimación de profundidad a partir de una vista [12], reconocimiento de lugares [13] o detección de objetos dinámicos [14]). Haciendo uso del aprendizaje profundo se pueden replantear los modelos de estimación de profundidad clásicos, pudiendo conseguir un sistema que proporcione mapas de profundidad densa, aplicable a un mayor tipo de secuencias y con una alta precisión. Para que dichas redes neuronales sean capaces de realizar este cometido han de ser previamente entrenadas con gran cantidad de imágenes. Una de las particularidades del algoritmo que se usa aquí es que no necesita tener la profundidad real de las imágenes para aprender a estimarla (aprendizaje no supervisado). En concreto, aprende profundidad extrayendo información de imágenes estéreo o de la consistencia temporal entre imágenes cercanas.

1.2. Objetivos

Este trabajo tiene como propósito final el entrenamiento y evaluación de modelos basados en redes neuronales profundas para la recuperación de profundidad en vídeos monoculares de endoscopias. Para alcanzar esta meta, se proponen los siguientes objetivos intermedios:

- Revisión del estado del arte en la estimación de profundidad mono-vista. Búsqueda del algoritmo que mejor se adapte a las necesidades del problema y que reporte un mayor rendimiento.
- Obtención y tratamiento de los datos de entrenamiento y evaluación para hacerlos compatibles con las librerías de entrenamiento de redes neuronales.
- Generación de profundidad pseudo-verdadera (*pseudo-ground truth* en inglés) a partir de las imágenes médicas estéreo para poder tener un marco de referencia sobre el que evaluar el rendimiento de los modelos entrenados y compararlos con otros métodos existentes.
- Realización de múltiples experimentos con la red neuronal escogida y selección de hiperparámetros, entrenándola de diferentes maneras con los vídeos de endoscopias.
- Evaluación de los experimentos y comparación con el estado del arte.

1.3. Herramientas

El procesamiento de las imágenes de entrenamiento y la obtención de la profundidad *pseudo-ground truth* se ha realizado en el entorno de programación MATLAB R2020a, porque el software utilizado para este objetivo está en este lenguaje. Por esta misma razón se ha utilizado Python 3.8.3 para el entrenamiento y experimentación de los modelos de redes neuronales para la estimación de profundidad. Más en detalle, hace uso de las librerías PyTorch 1.6.0 y OpenCV 4.4.0 para la construcción de la red y de Tensorboard 2.3.0 para su evaluación. Todo funcionando bajo CUDA 9.1, cuDNN 7.1.3.16 en Ubuntu 18.04. Para las figuras se han utilizado los programas Inkscape y draw.io.

1.4. Estructura de la memoria

El trabajo realizado se estructura en el presente documento de la siguiente manera:

– En el Capítulo 1 se presenta el problema y la importancia de la estimación de profundidad en vídeos de endoscopias y cómo se ha afrontado.

– En el Capítulo 2 se explican con el detalle todos los conceptos necesarios para entender la investigación realizada.

– En el Capítulo 3 se desglosa el problema de la estimación de profundidad, mostrando los retos a vencer, el estado del arte y la forma escogida de resolverlo.

– En el Capítulo 4 se comentan y evalúan los experimentos realizados.

– En el Capítulo 5 se extraen las conclusiones más importantes derivadas del trabajo realizado y se aventuran las contribuciones adicionales que se podrían agregar para mejorar los actuales resultados.

Capítulo 2

Fundamentos

2.1. Endoscopia

Una endoscopia es un procedimiento donde los órganos de un ser vivo son observados usando un instrumento llamado endoscopio [15]. Éste consiste en un tubo largo, delgado y flexible que porta una cámara con iluminación en un extremo cuyas imágenes se emiten directamente en una pantalla que el endoscopista mira durante la operación (figura 2.1). Dependiendo de la región que se inspeccione el procedimiento tiene un nombre distinto. El objetivo del proyecto EndoMapper es trabajar con endoscopias gastrointestinales, es decir, en todo el tubo digestivo. Sin embargo, debido a la falta de bases de datos de esta naturaleza a fecha de realización de este trabajo, aquí se va a trabajar con otro tipo de endoscopias denominadas laparoscopias. Esta técnica se diferencia de la anterior porque sirve para inspeccionar aquellas zonas que no pueden ser alcanzadas por orificios naturales. El acceso al interior del cuerpo se produce a través de un a pequeña incisión cutánea.

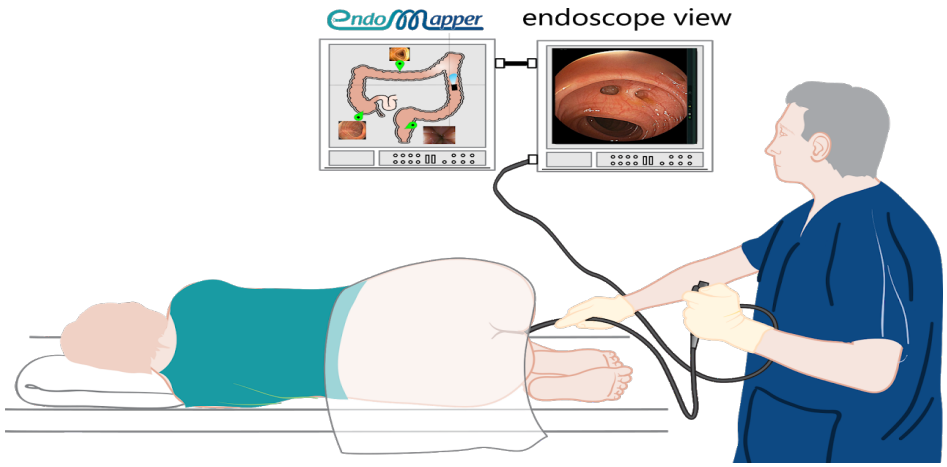

Figura 2.1: Representación gráfica de una endoscopia donde el profesional tiene a su disposición, además de la pantalla con la visualización del endoscopio, el sistema de apoyo virtual que se está desarrolando desde Endomapper. Fuente: JMM Montiel, licencia CC BY-SA 4.0 [16].

Para conseguir un mayor ángulo de visión, la cámara del endoscopio tiene una lente curva que consigue agrandarlo a costa de deformar ligeramente la imagen. Esto no representa un gran problema para el endoscopista, pero sí para los algoritmos de estimación de profundidad con los que se ha trabajado aquí. Por esta razón, es necesario corregir previamente esta distorsión de las imágenes. La forma de solucionarlo se explicará en la siguiente sección.

2.2. Visión por computador

El fin común de la visión por ordenador o visión artificial es la extracción de información a partir de imágenes digitales. Dicha rama de la ciencia incluye técnicas para, por ejemplo, recuperar la forma tridimensional y apariencia de objetos, rastrear su posición en un vídeo, reconocer facialmente a las personas o, por supuesto, estimar la profundidad de una escena. También se incluye el estudio de cómo influyen las características de las cámaras en la captura de las imágenes. A continuación, se van a explicar algunos términos básicos necesarios para entender el trabajo.

2.2.1. Geometría 3D

Un punto en el espacio euclídeo tridimensional puede ser representado usando coordenadas cartesianas ($\boldsymbol{x} = (x, y, z)^T \in \mathbb{R}^3$) o mediante sus coordenadas homogéneas ($\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w})^T \in \mathbb{P}^3$) [17]. También resulta de gran utilidad utilizar el vector aumentado ($\bar{\boldsymbol{x}} = (x, y, z, 1)^T$) para referirse al punto 3D. Saber que resulta muy cómodo pasar de un sistema a otro con $\tilde{\boldsymbol{x}} = \tilde{w}\bar{\boldsymbol{x}}$.

Dado un vector de traslación $\boldsymbol{t} \in \mathbb{R}^3$

$$\boldsymbol{t} = [t_x \; t_y \; t_z]^T \tag{2.1}$$

y una matriz de rotación $\boldsymbol{R} \in \boldsymbol{SO}(3)$

$$\boldsymbol{R} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} \tag{2.2}$$

donde $\boldsymbol{SO}(3)$ es el grupo ortogonal especial y, por tanto, $\boldsymbol{R}\boldsymbol{R}^{\boldsymbol{T}} = \boldsymbol{I}$ (matriz identidad) y $|\boldsymbol{R}| = 1$. La transformación euclídea 3D de un punto $\boldsymbol{x}$ puede expresarse como $\boldsymbol{x'} = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$ o

$$\boldsymbol{x'} = [\, \boldsymbol{R}\, \boldsymbol{t} \,]\, \bar{\boldsymbol{x}} \tag{2.3}$$

2.2.2. Modelo de cámara estenopeica

El modelo de cámara estenopeica (*pinhole*) (figura 2.2) es un sistema de cámara que modela una superficie opaca con un un pequeño orificio (símil del centro de la cámara o centro óptico) entre el objeto que se quiere captar y el sensor fotográfico (plano imagen). De esta manera, el sensor solo recibe aquellos rayos luminosos que parten del objeto y que pasan por el agujero. Debido a que los rayos de luz atraviesan el orificio con un cierto ángulo de entrada, la imagen captada por el sensor se encuentra invertida 180º. La distancia que separa el plano imagen del centro óptico es la distancia focal f.

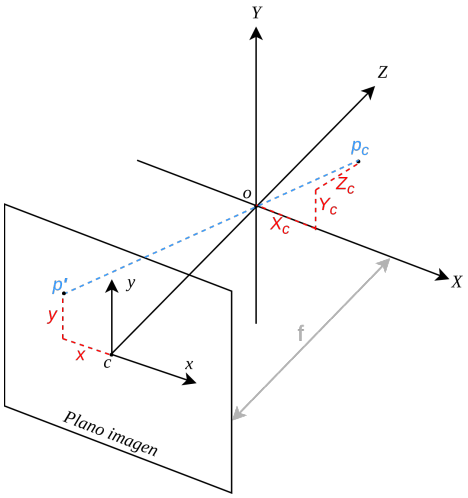

Figura 2.2: Modelo de cámara estenopeica donde se muestra la proyección de un punto en coordenadas cámara 3D $\boldsymbol{p_c}$ a un punto en coordenadas píxel 2D $\boldsymbol{p'}$. $\boldsymbol{o}$ es el centro óptico (origen del sistema de coordenadas cámara) y $\boldsymbol{c}$ el origen del sistema de coordenadas del plano del sensor de la cámara.

2.2.3. Calibración

En este subapartado primero se va a mostrar cómo influyen las características de la cámara en la proyección de un punto en coordenadas cámara a coordenadas píxel. Nótese como en la figura 2.2 el triángulo $\overline{\boldsymbol{ocp'}}$ es similar al triángulo formado por $\boldsymbol{o}$, $\boldsymbol{p_c}$ y $(0, 0, z_c)$. Por lo que, según la ley de triángulos similares [18] encontramos como pasar de coordenadas cámara a coordenadas píxel

$$\boldsymbol{p'} = [x\ y] = [f\frac{x_c}{z_c}\ f\frac{y_c}{z_c}] \tag{2.4}$$

Teniendo en cuenta que en las cámaras digitales el origen del sistema de coordenadas no suele encontrarse en el centro del plano imagen sino en un extremo ($\boldsymbol{c} = (c_x, c_y)$), la ecuación 2.4 queda como

$$\boldsymbol{p'} = [x\ y] = [f\frac{x_c}{z_c} + c_x\ \ f\frac{y_c}{z_c} + c_y] \tag{2.5}$$

El siguiente efecto a tener en cuenta es que los puntos en imágenes digitales están expresados en píxeles, mientras que los puntos en el plano imagen están en unidades métricas. Para realizar el cambio de unidades hay que multiplicar por una constante k o l, ya que la relación de aspecto de los píxeles puede que no sea la unidad, es decir, pueden ser píxeles no cuadrados. La ecuación 2.5 se transforma con ello de la siguiente manera

$$\boldsymbol{p'} = [x\ y] = [fk\frac{x_c}{z_c} + c_x \quad fl\frac{y_c}{z_c} + c_y] = [f_x\frac{x_c}{z_c} + c_x \quad f_y\frac{y_c}{z_c} + c_y] \tag{2.6}$$

Todo este modelo de proyección se puede expresar en forma matricial si usamos coordenadas homogéneas. Para convertir un vector euclídeo $[x\ y\ z]$ en coordenadas homogéneas se añade una nueva dimensión con valor unidad $[x\ y\ z\ 1]$. Si esta última coordenada no fuera 1, para pasar de coordenadas homogéneas $[x\ y\ z\ w]$ a euclídeas habría que dividir todas las dimensiones por el valor de ésta última: $[\frac{x}{w}\ \ \frac{y}{w}\ \ \frac{z}{w}\ \ 1]$. Haciendo uso a partir de ahora siempre de estas coordenadas homogéneas, el proceso de proyección de un punto 3D en coordenadas cámara a un punto 2D en coordenadas píxel queda como

$$\boldsymbol{p'} = \begin{bmatrix} f_x x_c + c_x z_c \\ f_y y_c + c_y z_c \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{p_c} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{I}\ \boldsymbol{0} \end{bmatrix} \boldsymbol{p_c} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{I}\ \boldsymbol{0} \end{bmatrix} \boldsymbol{p_c} \tag{2.7}$$

siendo $\boldsymbol{K}$ la matriz de calibración y $\boldsymbol{0}$ la matriz nula (de dimensión 3x1). Esta matriz $\boldsymbol{K}$ constituye los parámetros intrínsecos de la cámara, que son necesarios para pasar un punto de coordenadas cámara a coordenadas píxel. Sin embargo, como se puede ver en la figura 2.3, si el punto viene dado en referencia a un sistema de coordenadas mundo diferente al sistema de coordenadas de la cámara (lo que es lo normal), se requiere de un paso previo para realizar esta proyección.

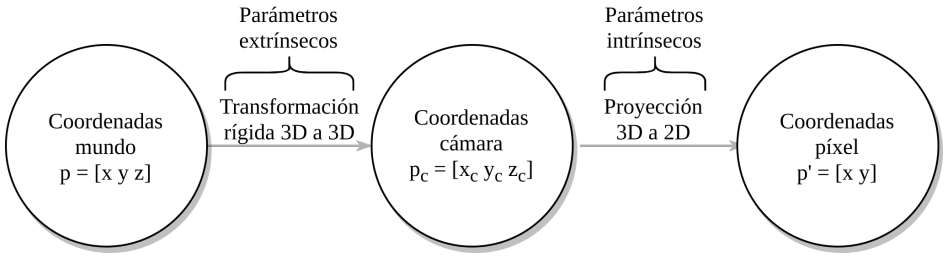

Figura 2.3: Proyección de un punto en coordenadas mundo a coordenadas píxel.

Esta transformación es rígida ya que solo consiste en la rotación y traslación necesaria para alinear y hacer coincidir el sistema de coordenadas mundo con el sistema

de coordenadas cámara, es decir, definen la posición de la cámara en el sistema de coordenadas del mundo real. Las variables necesarias para este cometido reciben el nombre de parámetros extrínsecos de la cámara y son la matriz de rotación $\boldsymbol{R}$

$$\boldsymbol{R} = \underbrace{R_z(\gamma)}_{deriva}\underbrace{R_y(\beta)}_{cabeceo}\underbrace{R_x(\alpha)}_{alabeo} = \begin{bmatrix} cos\gamma & -sen\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & 0 & sen\beta \\ 0 & 1 & 0 \\ -sen\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sen\alpha \\ 0 & sen\alpha & cos\alpha \end{bmatrix} \tag{2.8}$$

y el vector de traslación $\boldsymbol{t}$

$$\boldsymbol{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{2.9}$$

Teniendo en cuenta estas nuevas consideraciones, la proyección de un punto de coordenadas mundo a coordenadas píxel queda como

$$\boldsymbol{p'} = \boldsymbol{K}(\boldsymbol{R}\boldsymbol{p} + \boldsymbol{t}) = \underbrace{\boldsymbol{K}\left[\boldsymbol{R} \mid \boldsymbol{t}\right]}_{\boldsymbol{P}}\boldsymbol{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.10}$$

Siendo $\boldsymbol{P}$ la matriz de proyección. Para facilitar tareas posteriores es mejor expresarla en formato matriz 4x4 invertible

$$\tilde{\boldsymbol{P}} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} = \tilde{\boldsymbol{K}}\boldsymbol{E} \tag{2.11}$$

donde $\tilde{\boldsymbol{K}}$ es la matriz de calibración de rango completo y $\boldsymbol{E}$ es la transformación (euclídea) rígida 3D.

2.2.4. Proyección de un punto de una cámara a otra

Cuando se dispone de un sistema de cámaras multivista (más de una cámara observando la misma escena al mismo tiempo o la misma cámara pero en diferente instante espacio-temporal), es posible proyectar un punto de una cámara a otra. Este proceso se divide en cuatro pasos (ver figura 2.4). En el primero, el punto en coordenadas píxel de la cámara 1 ($\boldsymbol{p'_1}$) se proyecta a coordenadas cámara 1 ($\boldsymbol{p_{c1}}$) multiplicando por la inversa de la matriz de calibración de rango completo de la cámara 1 ($\tilde{\boldsymbol{K}}_\mathbf{1}$) y por la profundidad del punto (z_{c1}). Después, para pasar de coordenadas cámara 1 ($\boldsymbol{p_{c1}}$) a coordenadas mundo ($\boldsymbol{p}$) se multiplica por la inversa de la matriz de transformación rígida de la cámara 1 ($\boldsymbol{E_1}$). Una vez aquí hay que seguir los mismos pasos inversos pero para la cámara 2. Se multiplica por la matriz de transformación rígida de la cámara 2 ($\boldsymbol{E_2}$) para pasar a coordenadas cámara 2 ($\boldsymbol{p_{c2}}$), por la matriz de calibración de rango completo de la cámara 2 ($\tilde{\boldsymbol{K}}_\mathbf{2}$) para obtener $\boldsymbol{p'_2}$ y finalmente se

pasa de coordenadas homogéneas a coordenadas euclídeas para extraer las coordenadas píxel en formato $[x, y]$. Cuando se trabaja con cámaras estéreo se conoce y, por tanto, trabaja con una matriz de transformación rígida total

$$\boldsymbol{E_T} = \boldsymbol{E_1}^{-1}\boldsymbol{E_2} \tag{2.12}$$

que indica la posición del sistema de coordenadas de la cámara 2 respecto el de la cámara 1.

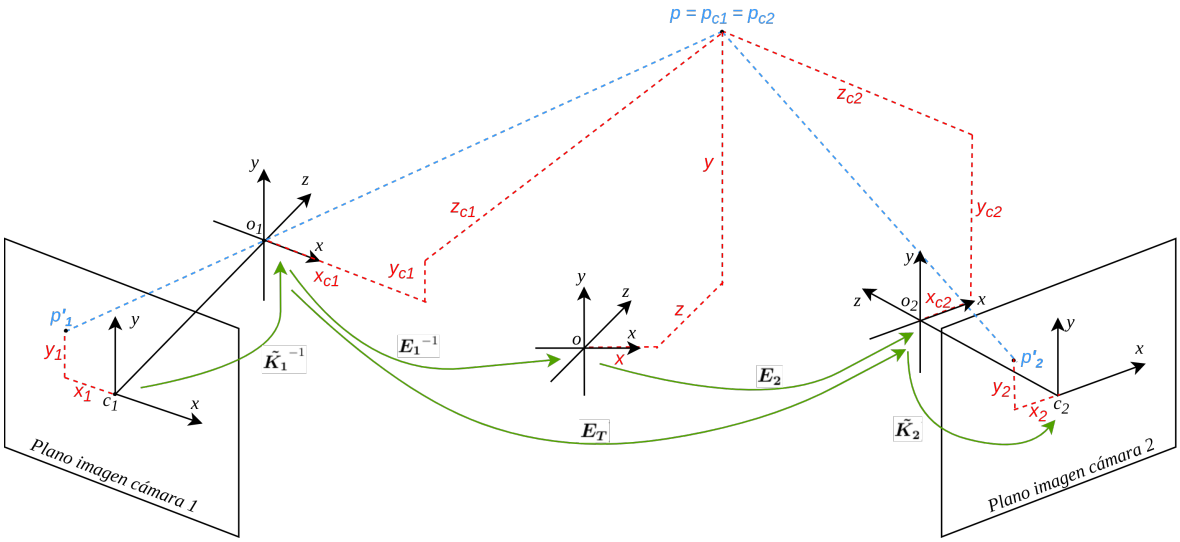

Figura 2.4: Modelo de cámara multivista donde se muestra la proyección de un punto en coordenadas píxel de la cámara 1 ($\boldsymbol{p'_1}$) a coordenadas mundo ($\boldsymbol{p}$), expresado también en coordenadas cámara 1 ($\boldsymbol{p_{c1}}$) y en coordenadas cámara 2 /$\boldsymbol{p_{c2}}$) y después a coordenadas píxel de la cámara 2 ($\boldsymbol{p'_2}$). $\boldsymbol{o}$ es el origen del sistema de coordenadas mundo, y $\boldsymbol{o_1}$ y $\boldsymbol{o_2}$ son los centros ópticos de las cámaras 1 y 2, respectivamente. $\boldsymbol{c_1}$ y $\boldsymbol{c_2}$ son los orígenes de los sistemas de coordenadas del plano del sensor de la cámara 1 y 2, respectivamente.

2.2.5. Corrección de la distorsión de las lentes

El modelo de cámara previamente expuesto asume que las cámaras obedecen a un modelo de proyección lineal donde líneas rectas en el mundo real resultan en líneas rectas en la imagen [17]. Desafortunadamente, lo normal es que las cámaras posean una lente que distorsiona la imagen captada. Este efecto se acentúa en el caso de los endoscopios, ya que cuentan con lentes de gran angular con notable distorsión radial que curva la imagen de una de las formas que se muestra en la figura 2.5. Debido a las lentes *ojo de pez* de los endoscopios, la distorsión radial en ellos es del tipo barril.

Para compensar este efecto, un modelo cuadrático de la distorsión suele ser suficiente y en nuestro caso lo es. Este modelo nos dice que las coordenadas en sistema de referencia cámara x_c y y_c se encuentran desplazadas hacia fuera (distorsión de barril)

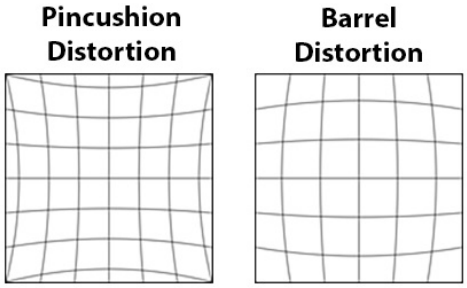

Figura 2.5: Izquierda: distorsión radial de tipo cojín (*pincushion distortion*). Derecha: distorsión radial de tipo barril (*barrel distortion*). Fuente: Oculus VR LLC [19].

o hacia dentro (distorsión de cojín) del centro de la imagen una distancia proporcional a su distancia radial:

$$\begin{aligned} \delta_x^r &= x_c(1 + k_1 r^2 + k_2 r^4) \\ \delta_y^r &= y_c(1 + k_1 r^2 + k_2 r^4) \end{aligned} \tag{2.13}$$

siendo k_1, k_2 y $r^2 = x_c^2 + y_c^2$ los parámetros de distorsión radial.

Para una corrección de la distorsión óptima, en este trabajo también se ha tenido en consideración la distorsión tangencial causada porque la lente nunca está perfectamente paralela al plano imagen. En este caso, los desplazamientos que hay que corregir para obtener las coordenadas cámara x_c y y_c correctas también tienen que ver con la distancia al centro de la imagen:

$$\begin{aligned} \delta_x^t &= x_c + [2p_1 x_c y_c + p_2(r^2 + 2x_c^2)] \\ \delta_y^t &= y_c + [2p_2 x_c y_c + p_1(r^2 + 2y_c^2)] \end{aligned} \tag{2.14}$$

siendo p_1, p_2 y r los parámetros de distorsión tangencial.

2.2.6. Geometría epipolar, rectificación y cálculo de la profundidad en imágenes estéreo

La geometría epipolar es propia de la visión estéreo. Un punto en el espacio 3D ($\boldsymbol{p}$) visto desde dos cámaras en distinta posición junto a los centros ópticos de ambas cámaras ($\boldsymbol{o_1}$ y $\boldsymbol{o_2}$) definen un plano, el plano epipolar (ver figura 2.6). En esta sección se ha utilizado un modelo de cámara similar al estenopeico, pero a diferencia de éste el centro óptico de cada cámara ha sido colocado detrás del plano imagen. Al hacer esto, la imagen capturada no se encuentra volteada 180º. La línea recta que une ambos centros ópticos se denomina línea epipolar, y el corte de esta línea con los planos imagen 1 y 2 da lugar a los epipolos o puntos epipolares 1 y 2 ($\boldsymbol{e_1}$ y $\boldsymbol{e_2}$). Un punto en la imagen 1 ($\boldsymbol{p'_1}$) puede proyectarse en el espacio 3D (punto $\boldsymbol{p}$) si se conocen los parámetros de la cámara 1 y la profundidad de dicho punto en el espacio. Sin embargo, si esta profundidad no es conocida, el punto podría estar en cualquier lugar de la recta $\overline{\boldsymbol{o_1 p'_1}}$ ($\boldsymbol{p}$, $\boldsymbol{p_1}$, $\boldsymbol{p_2}$ o

$\boldsymbol{p_3}$, por ejemplo). Esta recta es vista como un punto en el plano imagen 1 (punto $\boldsymbol{p'_1}$) pero como una recta que siempre pasa por el epípolo correspondiente ($\boldsymbol{e_2}$) en el plano imagen 2 (recta $\overline{\boldsymbol{e_2 p'_2}}$). Dicha recta $\overline{\boldsymbol{e_2 p'_2}}$ indica todas las posiciones posibles en las que podría caer la proyección del punto $\boldsymbol{p'_1}$ en el plano cámara 2.

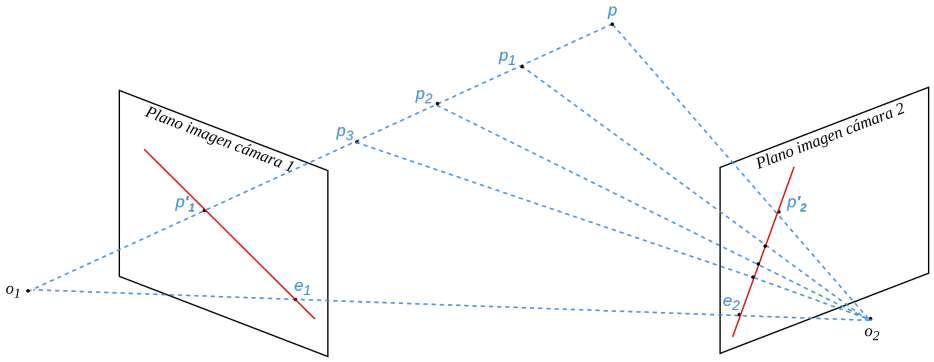

Figura 2.6: Geometría epipolar de dos cámaras estéreo.

Si la línea $\overline{\boldsymbol{e_2 p'_2}}$ fuera horizontal en lugar de tener una inclinación dependiente del punto escogido y estuviera a la misma altura que la línea $\overline{\boldsymbol{e_1 p'_1}}$ (también horizontal), el proceso de emparejamiento de un punto en una cámara con su homónimo en la cámara contraria sería mucho más sencillo. La zona de búsqueda pasaría de ser cualquier punto en la imagen a una línea. El proceso de transformación para que esto ocurra con cualquier punto de una imagen se denomina rectificación y se consigue proyectando ambas imágenes sobre un plano común (véase la figura 2.7) haciendo uso de los parámetros intrínsecos y extrínsecos de ambas cámaras. No olvidar que antes de rectificar hay que eliminar la distorsión provocada por las lentes.

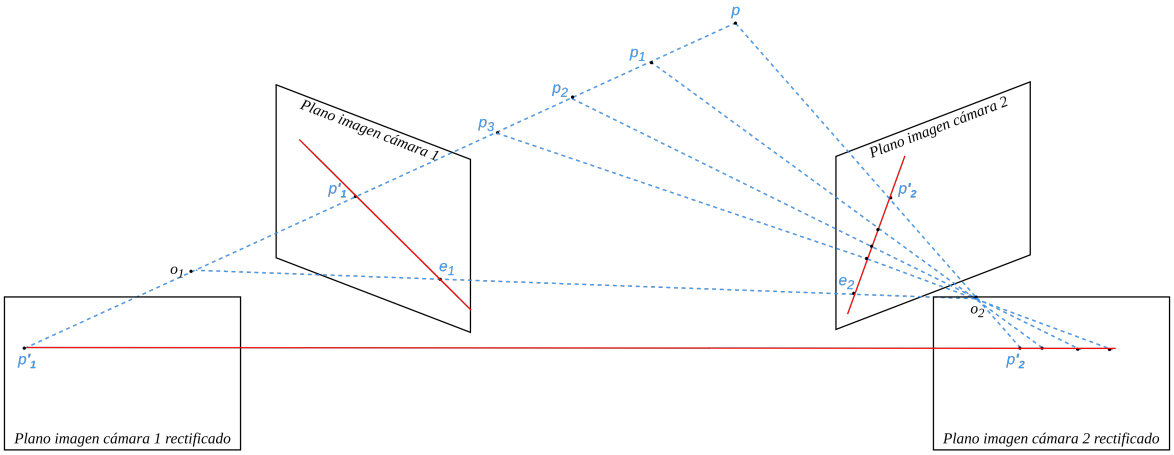

Figura 2.7: Rectificación de dos cámaras estéreo.

Otra de las ventajas de rectificar es que el cálculo de profundidad es mucho más sencillo. Si se conocen las coordenadas píxel en ambas cámaras rectificadas de un mismo punto ($\boldsymbol{p'_1}$ y $\boldsymbol{p'_2}$), la diferencia en valor absoluto de sus coordenadas horizontales

se denomina disparidad ($d = x_1 - x_2$). Sabiendo además la distancia focal f de ambas cámaras y la distancia euclídea entre los centros ópticos T (*baseline* en inglés), la profundidad del punto se calcula simplemente con

$$z_c = \frac{fT}{d} \tag{2.15}$$

2.3. Aprendizaje profundo

2.3.1. Aprendizaje automático

El aprendizaje profundo (*deep learning*) es un tipo específico de aprendizaje automático (*machine learning*), que a su vez es una rama de la inteligencia artificial. Para entender el aprendizaje profundo es necesario conocer los principios básicos del aprendizaje automático. El aprendizaje automático busca crear algoritmos y modelos que aprendan a realizar tareas en base a la experiencia [20]. La clave aquí es conseguir que la maquina aprenda los criterios relevantes automáticamente de observaciones pasadas y sea capaz de adaptarse a una nueva situación nunca antes vista (generalizar) sin que un humano codifique su comportamiento manualmente. La tarea que debe desempeñar suele poder expresarse con una función matemática del estilo

$$y = f(\boldsymbol{x}, \boldsymbol{w}) \tag{2.16}$$

donde $\boldsymbol{x}$ son los datos de entrada al modelo, y la salida del sistema, que puede ser continua (regresión) o discreta (clasificación), y $\boldsymbol{w}$ todos aquellos parámetros del modelo que son precisamente lo que se 'aprende'. El modelo debe ser invariante ante variaciones de la entrada irrelevantes, seleccionando las características correctas. Para medir su rendimiento normalmente se usan métricas sobre el porcentaje de clasificaciones correctas, de juegos ganados o de profundidad correctamente estimada.

Dependiendo de la clase de datos disponibles se pueden diferenciar cuatro grandes tipos de aprendizaje automático:

- Aprendizaje supervisado: si los datos de entrenamiento cuentan con su valor de salida correcto (*ground truth*). Por ejemplo, en estimación de profundidad, si además de las imágenes de entrada también hubieran imágenes de profundidad para todas ellas. En este caso la forma de entrenar al modelo sería minimizando el error entre la salida predicha por el modelo y la salida correcta.

- Aprendizaje no supervisado: si los datos de entrenamiento no cuentan con *ground truth*. En este caso, el algoritmo aprende a dar una salida en base a la estructura de los datos de entrenamiento sin necesidad de conocer su valor objetivo correcto.

– Aprendizaje semi-supervisado: si se tienen datos con y sin *ground truth*. La forma que aquí tiene el modelo de aprender es un híbrido entre aprendizaje supervisado y no supervisado.

– Aprendizaje por refuerzo [21]: en lugar de aprender a partir de datos pasados, el sistema busca de manera activa las acciones a realizar en una situación dada para maximizar una recompensa. Normalmente, hay una secuencia de estados y acciones en las que el algoritmo de aprendizaje interactúa con su entorno. En consecuencia, el modelo ajusta sus parámetros internos mediante un proceso de prueba y error.

2.3.2. Redes neuronales

En 1943 McCulloch y Pitts presentan el primer modelo neuronal moderno. La neurona de McCulloch-Pitts trata de simular el comportamiento de una neurona natural del cerebro humano y constituye la unidad básica de una red neuronal. Una neurona se encarga de realizar una suma ponderada de los valores de entrada y aplicar al resultado una función escalón, es decir, si la suma supera un valor concreto o umbral, la señal de salida vale 1 y en caso contrario 0. Esta definición de la neurona era poco flexible, por lo que no resultaba del todo útil. No fue hasta 1957 que Frank Rosenblatt inventa el Perceptrón [22]. Esta nueva definición de neurona artificial era más flexible y levantó un enorme revuelo en aquel entonces. Se pensaba que posibilitaría la creación de máquinas que fueran capaces de andar, hablar, ver, escribir, auto-reproducirse y ser conscientes de su propia existencia [23]. Un Perceptrón (figura 2.8) era similar a la neurona de McCulloch-Pitts, pero a diferencia de ésta cada señal de entrada tenía un peso asociado diferente al resto y la función de activación ya no se limitaba a un simple escalón, podía ser cualquier función no lineal. La función matemática de un Perceptrón es la siguiente

$$y = f(\sum_{j=0}^{n} w_j x_j) = f(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}) \tag{2.17}$$

donde y es la salida de la neurona, $\boldsymbol{x}$ el vector de entrada de dimensión deseada, $\boldsymbol{w}$ el vector de pesos y f la función de activación escogida. El valor de la primera entrada x_0 siempre es cero para que el peso w_0 sea el sesgo. De todas estas variables, solamente el vector de pesos y el sesgo se aprenden durante el entrenamiento.

Como era de esperar, el Perceptrón no sobrevivió a las expectativas. En 1969, Marvin Minsky y Seymour Papert mostraron que los Perceptrones (mono-capa) no podían resolver cualquier problema. Solo podían clasificar datos linealmente separables,

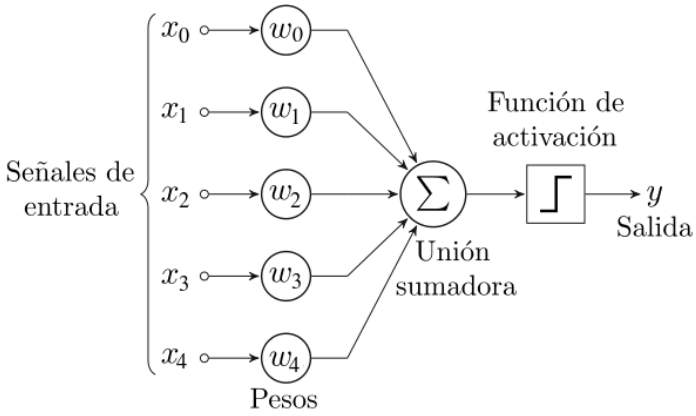

Figura 2.8: Perceptrón con cinco señales de entrada. Fuente: modificado de Alejandro Cartas CC BY-SA 4.0 [24] via Wikimedia Commons.

lo que limitaba enormemente su capacidad de aprendizaje y predicción. No fue hasta 1980 que se dio el resurgir de las redes neuronales al conseguirse notables éxitos con los Perceptrones multi-capa. Estos modelos consisten en la concatenación de varios Perceptrón multi-clase (más de una salida) uno tras otro (figura 2.9). Al colocarlos de tal manera, las capas intermedias reciben el nombre de capas ocultas. Las capas de entrada y salida tienen un número de neuronas que dependen de la dimensión de entrada y salida, pero las capas intermedias pueden tener tantas como se quieran. Si el número de nodos y capas ocultas es considerablemente grande, se consideran redes neuronales profundas (aprendizaje profundo). La forma de contabilizar el número de capas de una red es que este valor es igual al número de filas de nodos menos uno (no se cuenta la capa de entrada). Si la red neuronal consta de dos capas y la capa oculta tiene suficientes nodos, la red recibe el nombre de aproximador universal porque es capaz de aproximar bien cualquier función continua.

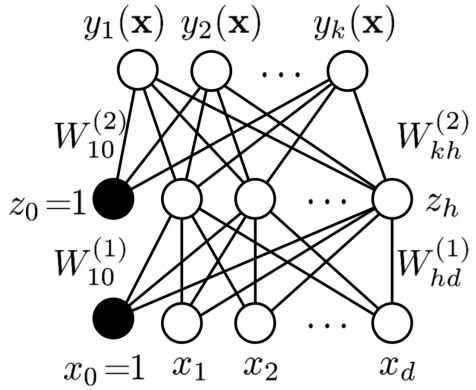

Figura 2.9: Perceptrón multi-capa. Fuente: B. Leibe [20].

En este caso los enredos de la red provocan que el modelo matemático para calcular

el valor de un solo nodo de la capa de salida sea mucho más complejo que antes

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right) \tag{2.18}$$

con $g^{(1)}$ y $g^{(2)}$ las funciones de activación no lineales escogidas, y d y h el número de entradas y de nodos en la capa oculta, respectivamente. Puede afirmarse que esta red neuronal no es más que un mapeo no lineal de unas variables de entrada $\{x_j\}$ en unas variables de salida $\{y_k\}$ controlado por un vector de pesos $\boldsymbol{w}$.

Aunque en teoría se podría escoger cualquier función de activación no lineal, cuando se trabaja con redes neuronales hay una máxima que siempre tiende a cumplirse: "El diseño más simple suele ser el más efectivo". Por eso las que se utilizan son de lo más simples. Existen multitud de opciones posibles, pero aquí se van a mostrar las más comunes y eficientes. La sigmoidal (figura 2.10a) tiene un suave gradiente y produce valores de salida normalizados entre 0 y 1. Su mayor problema es que para valores de entrada grandes la salida que produce es prácticamente constante. Esto hace que durante el entrenamiento la información no fluya bien cuando los valores son grandes porque el gradiente (derivada de la función) es casi nulo. La función de activación ReLU (figura 2.10b) evita esto último al propagar el gradiente de información con un factor constante. Esto facilita el entrenamiento en redes neuronales profundas y reduce el coste computacional notablemente. ReLU es ha convertido de hecho en el estándar de estas redes. Aún así, tiene ciertas limitaciones. ReLU tiene un sesgo de compensación intrínseco ya que sus salidas siempre serán positivas. Con la función de activación ELU (figura 2.10c) se consigue eliminar este sesgo a cambio de un coste computacional mayor.

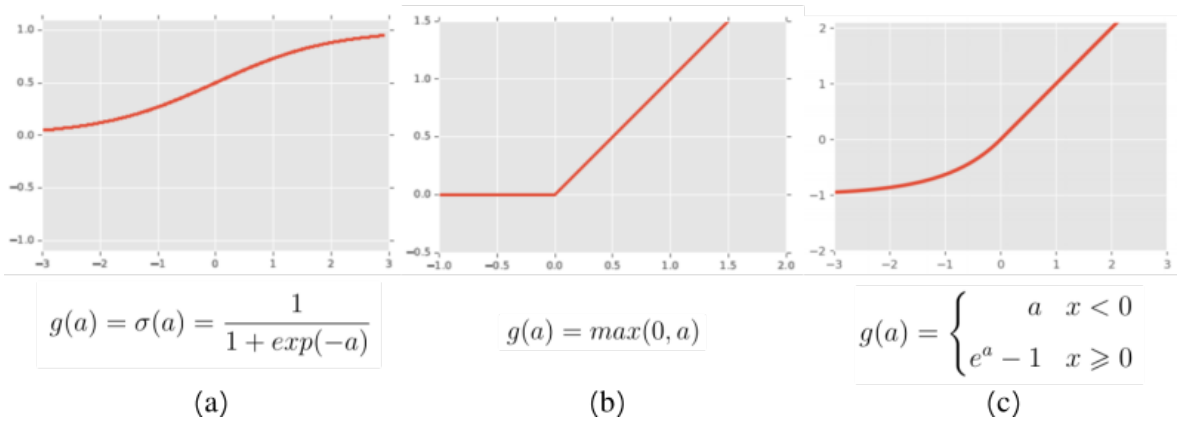

Figura 2.10: Funciones de activación. (a): Sigmoidal. (b): ReLU. (c): ELU. Fuente: modificado de B. Leibe [20].

A pesar de los buenos resultados cosechados, las redes neuronales cayeron de nuevo en el olvido ya que eran difíciles de entrenar, tenían parámetros poco intuitivos y

tendían a memorizar los ejemplos de entrenamiento en lugar de aprender. Esto último quiere decir que la red no se quedaba solo con las características relevantes de los ejemplos de entrenamiento, o lo que es lo mismo, los pesos se ajustaban de tal manera que la red solo daba resultados correctos si los datos de entrada habían sido vistos durante el entrenamiento. A este efecto se le llama sobreajuste (*overfitting*) y hay que tener mucho cuidado de que no ocurra durante el entrenamiento de una red neuronal para que sea capaz de generalizar a datos nunca antes vistos. Por si fuera poco, a partir de 1995 dentro del campo del aprendizaje automático, el interés se centro en otros métodos de aprendizaje como las máquinas de vectores de soporte (SVM o *Support Vector Machines*) que daban muy buenos resultados. Tuvieron que pasar 10 años más para que empezaran a aparecer progresos en las redes neuronales, con un mejor entendimiento de como entrenar satisfactoriamente redes neuronales profundas y unidades de procesamiento gráfico (GPU) más potentes. A pesar de estos avances, las redes todavía despertaban poco interés dentro de los investigadores del aprendizaje automático. Todo esto cambió en 2012 bajo el pretexto del reto anual de reconocimiento visual a gran escala de ImageNet (ILSVRC, *Imagenet Large Scale Visual Recognition Challenge*). Una competición que evalúa algoritmos para detección de objetos y clasificación de imágenes a gran escala [25]. Allí se presentó un modelo basado en redes neuronales convolucionales profundas que reducía a la mitad el ratio de error del estado del arte [26]. Desde entonces al aprendizaje profundo fue ampliamente adoptado.

2.3.3. Redes neuronales convolucionales

Para poder ser usadas en visión por computador, las redes neuronales deben adaptar su arquitectura. Como la entrada son imágenes, es decir, tensores 2D (o 3D en el caso de imágenes a color), las capas de la red deben también tener forma bi- o tri-dimensional. Otra característica básica es que como las imágenes de entrada pueden no ser perfectas, la red necesita ser robusta a desalineaciones. Además, por la propia naturaleza de la imagen, que es jerárquica, la red debe tener una estructura multi-capa. Esta jerarquía en las redes profundas surge porque las primeras capas aprenden características de bajo nivel, muy básicas en la imagen de entrada (como líneas o curvas). Las capas posteriores aprenden patrones de mayor complejidad (como partes de objetos o texturas) a partir de las anteriores. Por último, las capas finales son capaces de extraer características de muy alto nivel (como objetos completos o caras), que suelen estar relacionados con el objetivo final de la red. Las redes convolucionales (figura 2.11) se adaptan a todos estos requisitos. Este tipo de redes neuronales tienen una estructura de conectividad especial ya que pueden utilizar diferentes tipos de capas.

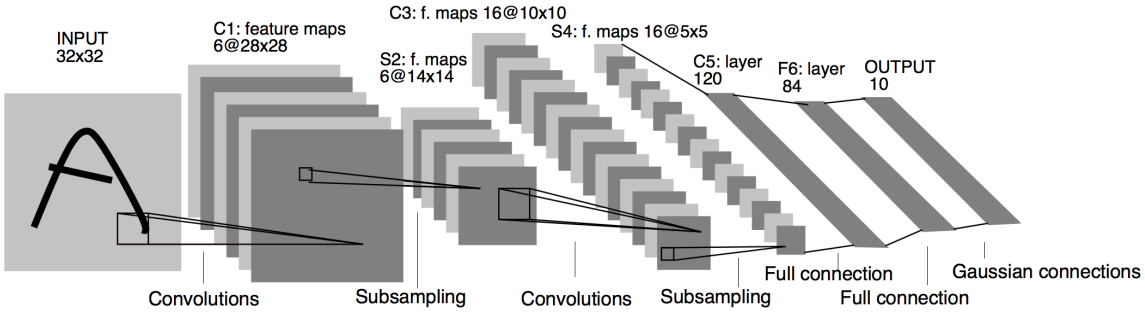

Figura 2.11: Arquitectura de una red neuronal convolucional, en concreto LeNet-5, diseñada para reconocimiento de dígitos. Fuente: LeCun et al. 1998 [27].

Una de ellas es la capa totalmente conectada (*full connection layer*), que es la clásica capa que utilizan los Perceptrones (figura 2.12a). En estas capas todas las neuronas están conectadas a todas las neuronas de la capa anterior, teniendo un valor único cada peso que relaciona cada par de neuronas. Si, por ejemplo, la imagen de entrada tuviera 1000x1000 píxeles, como cada neurona de la capa totalmente conectada se une con todas las anteriores, cada neurona tendría 1 millón de conexiones, es decir, 1 millón de parámetros únicos que han de ser aprendidos. Y si además de esto, la capa oculta tuviera tantas neuronas como la de entrada, habría un total de 1 billón de parámetros únicos solo para modelar una capa. Esto hace que el uso de estas capas sea inviable en capas intermedias. Solían colocarse al final de las redes convolucionales para ajustarse al número de neuronas de salida (como ocurre en la red de reconocimiento de dígitos LeNet-5 de la figura 2.11). Sin embargo, recientemente se ha demostrado que se puede conseguir el mismo efecto simplemente agregando más capas convolucionales que tienen la ventaja de reducir el número de parámetros drásticamente.

Otro de los tipos de capas es la localmente conectada (figura 2.12b), donde cada una de sus neuronas se conecta solamente con un grupo reducido de neuronas de la capa anterior. De esta forma el número de conexiones de cada neurona, y por ende de parámetros, es mucho menor. Entonces si, por ejemplo, tenemos una imagen de entrada 1000x1000, una capa oculta con el mismo número de unidades y cada una de ellas con un campo receptivo (también llamado filtro o kernel) de tamaño 10x10, el número de parámetros total es de 100 millones. Estas capas se usan cuando los datos de entrada tienen características locales. Sin embargo, se encuentran en desuso a favor de las capas convolucionales.

La evolución de la capa localmente conectada es la capa convolucional (figura 2.12c), la quintaesencia del aprendizaje profundo. Ésta es similar a la localmente conectada ya que cada neurona observa una región reducida de la entrada, pero en lugar de que cada neurona tenga parámetros únicos, estos se comparten entre todas las neuronas de la

capa convolucional. De esta forma, en el mismo ejemplo de una imagen 1000x1000, una capa oculta de 1000x1000 nodos y un kernel 10x10, tiene únicamente 100 parámetros que aprender. Es decir, un número independiente al tamaño de la imagen de entrada y del número de neuronas de la capa oculta. La magia ocurre cuando muchas de estas capas se utilizan para ver la misma entrada formando una sola capa (figura 2.12d). En consecuencia, la salida pasa a ser un tensor 3D en lugar del tensor 2D que era antes. El número de parámetros no se dispara, sino que crece linealmente con el número de capas apiladas juntas. La razón por la que algo así puede funcionar es que cada capa colocada unida, denominada mapa de características (*feature map*), se encarga de detectar una característica en concreto (por ejemplo, una capa busca líneas rectas mientras otra capa busca curvas). Al utilizarse muchos de estos mapas, la red es capaz de vislumbrar muchos y muy diferentes rasgos de la imagen que la hacen única. Conforme la red se hace más profunda, las capas convolucionales extraen características de mayor nivel, como formas complejas, y finalmente objetos, caras o el objetivo final de la red (la jerarquía que se explicaba al principio de esta subsección).

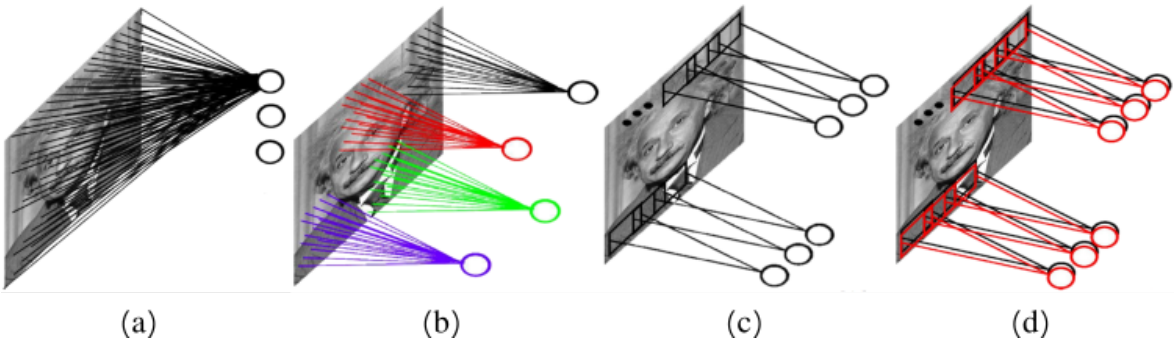

Figura 2.12: (a): Capa totalmente conectada. (b): Capa localmente conectada. (c): Capa convolucional con un solo mapa de características. (d): Capa convolucional con varios mapas de características. Cada círculo simula una neurona o nodo. Cada línea en (a) y (b) representa una conexión directa de un píxel de la imagen con una neurona y tiene un peso asociado. Fuente: adaptado de Marc'Aurelio Ranzato 2017 [28].

La capa de agrupación (*pooling layer*) es otro tipo de capa similar a la localmente conectada en la que cada nodo observa una región reducida de la entrada, pero a diferencia de ésta una capa de agrupación no tiene pesos que han de ser aprendidos. El numero de nodos de estas capas siempre es menor que el de la capa de entrada, porque el objetivo que se persigue cuando se usa esta capa es reducir la dimensión de la entrada. Dependiendo de la forma en que la capa hace esto recibe un nombre u otro. En una capa de agrupación máxima (*max pooling*), que es la forma más comunmente usada para reducir la dimensión, cada nodo se fija en el valor de cada nodo al que está conectado y se queda con el que tiene el valor más alto. Una capa de agrupación media (*average pooling*) reduce la dimensión al devolver el valor medio de todos sus nodos de

entrada.

También existen capas que se encargan de hacer lo contrario a las capas de agrupación, es decir, de aumentar la dimensión de la entrada. Este tipo de capas reciben en inglés el nombre de *unpooling layers*). Una forma de hacerlo es usando el método del vecino más cercano (*nearest neighbor*). Lo que hace cada nodo es simplemente copiar el valor del nodo respectivo más cercano de la entrada. De esta forma los nodos de salida tienen valores repetidos pero se ha aumentado la dimensión. Otra forma es usando capas convolucionales inversas, es decir, que partiendo de un nodo de entrada surjan múltiples nodos con valores dependientes de los pesos aprendidos.

2.3.4. Entrenamiento de redes neuronales

Por entrenamiento se entiende el proceso de variación de los parámetros de la red para reducir una función de coste (previamente diseñada) relacionada con el desempeño de la red para una tarea concreta. Por ejemplo, una función de coste sencilla para una red que predijera profundidad podría ser la distancia L1

$$L_1(I_a, I_b) = \|I_a - I_b\|_1 = \sum_{i=1}^{n} |I_a(i) - I_b(i)| \tag{2.19}$$

donde I_a es la profundidad estimada e I_b la profundidad *ground truth*. La función de coste es el sumatorio de la diferencia en valor absoluto para cada píxel i.

Durante el proceso de entrenamiento, el valor de la función de coste se reducirá. Si ésta tiene un valor bajo quiere decir que la red está haciendo bien su trabajo con los datos de entrenamiento. Por tanto, se espera que también funcionará decentemente bien con los datos de testeo.

Para entrenar la red lo primero es calcular el gradiente de la función de coste respecto cada parámetro que se quiere entrenar (pesos y sesgos). Esto da una idea del grado de influencia de cada parámetro en la función de coste de la red. Dependiendo de su valor y signo, el gradiente indica cómo hay que modificar dicho parámetro para reducir la función de coste. Este proceso se repite cada vez que se han visto un cierto número reducido de datos (*batch size*) durante el entrenamiento para que no sature la memoria disponible de la GPU. Cuando los datos son imágenes el tamaño del lote suele ser de 6 a 12 unidades. Cuando se han visto ya todos los datos de entrenamiento se dice que se ha completado una época. La red se entrena durante varias épocas hasta que la función de coste no se reduce más sin que se llegue a producir el ya comentado sobreajuste.

De todos los métodos de optimización disponibles, aquí se va a explicar el método Adam [29] que por su buen desempeño se ha convertido en el quasi-estandar y el que

se ha usado en este trabajo. Adam (Estimación de Momento Adaptativo o *Adaptative Moment Estimation*) hace uso de dos tecnologías de aprendizaje ya conocidas que son el momento y los ratios de aprendizaje adaptativos para cada parámetro. El momento modela un efecto similar a un balón rodando colina abajo, que rueda más rápido en dirección al mínimo local y oscila debido a su inercia. Para introducir el momento en las redes neuronales, se añade un elemento temporal que actualiza el valor del vector actual en función del anterior. La actualización de los momentos se calculan con

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{2.20}$$

donde m_t y v_t son las estimaciones del primer momento (media) y del segundo momento (varianza descentrada) de los gradientes, respectivamente, y g_t el gradiente de la función de coste en el instante t respecto cada parámetro que se entrena θ (como pesos o sesgos). Para evitar que estas variables estén sesgadas hacia cero, se calculan las siguientes estimaciones de primer y segundo momento tal que

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{2.21}$$

Estos valores son usados para actualizar los parámetros de la red con lo siguiente

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \tag{2.22}$$

donde η es el ratio de aprendizaje inicial. Por defecto se aconseja 0.9 para β_1, 0.999 para β_2 y 10^{-8} para ϵ.

2.3.5. Redes neuronales residuales

Durante el entrenamiento de una red neuronal típica, los primeros pesos que se actualizan son los de las últimas capas. En dichas capas el error es mayor, por lo que el gradiente respecto sus pesos es más acentuado y provoca que éstos varíen mucho. Conforme se van actualizando los pesos de capas más tempranas, el error se va reduciendo y, por tanto, la contribución al error por parte de cada peso va a menos haciendo que estos se actualicen menos. Este problema recibe el nombre de desvanecimiento del gradiente. Las redes neuronales recurrentes (ResNet, *residual neural networks*) son como las redes convolucionales pero con una simple y eficaz mejora estructural para hacer frente a este problema. Como se puede ver en la figura 2.13, cada dos capas tradicionales se coloca una conexión que posibilita que el flujo de información no tenga que pasar por ellas. Esto permite que el entrenamiento, el

gradiente de información fluya más fácilmente a las capas más tempranas. Tras aplicar esta nueva característica, las redes residuales son más fáciles de entrenar y consiguen unos resultados mejores al de las redes tradicionales. El secreto detrás del éxito de las ResNet es que al implementar estas conexiones de salto lo que se está consiguiendo en verdad es un ensamblaje de redes poco profundas. Está comprobado que de todos los cáminos posibles, las ResNet utilizan solo caminos relativamente poco profundos con 5 a 17 módulos ([30]). En consecuencia, si alguna capa es eliminada, la red sigue funcionando prácticamente igual.

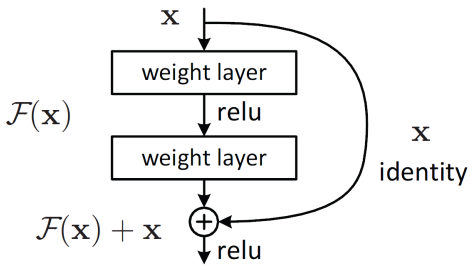

Figura 2.13: Componente básico de las redes residuales: dos capas convolucionales con una conexión de salto y función de activación ReLU. Fuente: © 2016 IEEE [31].

Una red neuronal recurrente típica, que de hecho es la que se usa en este trabajo, es la ResNet-18 (figura 2.14). Esta red consta de tan solo 18 capas convolucionales con sus respectivas conexiones de salto. Al principio cuenta con una capa convolucional y una de agrupación máxima (*pool*) que reduce la dimensión de la imagen de entrada. Al final cuenta con otra capa de agrupación media (*avg pool*) y una capa totalmente conectada para que la salida sea un vector de dimensión determinada (1000 en el ejemplo de la figura 2.14).

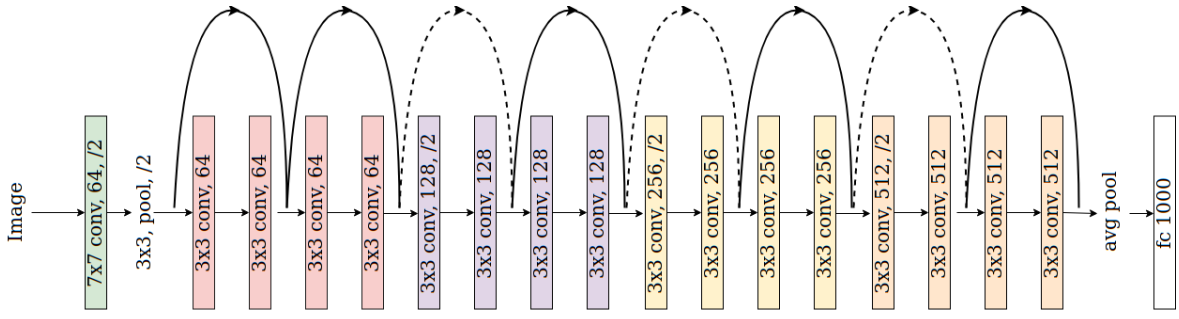

Figura 2.14: ResNet-18. El tamaño del kernel es de 7x7 para la primera capa y de 3x3 para el resto de capas. El número de mapas de características de cada capa es de 64 en la primera capa y se dobla cada cuatro capas residuales a la vez que la resolución de los mapas se reduce a la mitad (/2). Fuente: MDPI licencia CC BY 4.0 [32].

2.3.6. Autocodificador

Las arquitecturas convolucionales típicas comprimen la información de la imagen de entrada en un vector o un tensor de dimensiones reducidas (espacio latente o *latent space*), lo cual es útil para tareas de clasificación. Sin embargo, en nuestra aplicación de estimación de profundidad, queremos un valor para cada píxel, y por tanto una salida de la red que conserve la misma dimensión que la imagen de entrada. La arquitectura adecuada para conseguirlo está basada en autocodificadores (figura 2.15a), que no son más que una red convolucional que comprime la información en el espacio latente (*encoder*) seguida de otra red convolucional inversa que lo expande hasta obtener una imagen de nuevo (*decoder*). Una limitación de estas redes es que al comprimir toda la imagen en un número reducido de variables, las características más sutiles y no tan generales de la imagen de entrada se pueden perder. Para que la información de alta frecuencia pueda ser compartida entre la entrada y salida, se pueden colocar conexiones entre las capas con la misma dimensión del *encoder* y *decoder*. A este tipo de redes se las conoce como U-Net (figura 2.15b), y es la arquitectura que se usa para predecir profundidad en este trabajo.

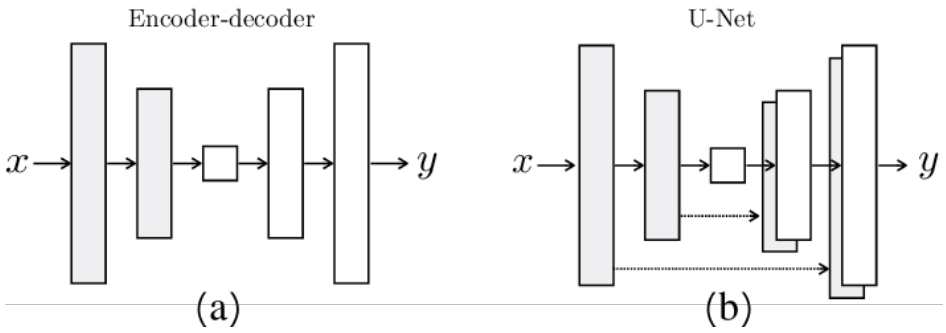

Figura 2.15: Arquitectura de un autocodificador (a) y de una U-Net (b). Fuente: © 2017 IEEE [33].

Capítulo 3

Estimación de profundidad

3.1. Retos

– Debido a la falta de bases de datos libres de colonoscopias humanas, aquí se va a trabajar con una base de datos de endoscopias de animales, por lo que hay que adaptarse a sus peculiaridades. En concreto, la base de datos escogida recopila secuencias grabadas con diferentes endoscopios estéreo, por lo que el sistema que se diseñe debe ser capaz de trabajar con vídeos que tienen diferentes resoluciones y parámetros de cámara.

– Las imágenes médicas en sí ya son un reto. Los algoritmos de estimación de profundidad suelen estudiarse en secuencias de entornos exteriores (como calles), donde la grabación es estable, los objetos tienen buena textura e iluminación y en su mayoría permanecen estáticos, y la disposición de la escena suele ser similar en todo el vídeo (si las imágenes muestran calles, por ejemplo, en la parte inferior siempre hay una carretera que tiende a un punto de fuga en el que arriba está el cielo y a los lados edificios). Esto no ocurre cuando se introduce una cámara en el cuerpo de un ser vivo donde los tejidos carecen de textura, están en constante deformación, hay una pobre iluminación con fuertes reflejos especulares, herramientas quirúrgicas, la cámara está muy cerca, el movimiento es errático, apenas hay unos centímetros de diferencia de profundidad y abundan las oclusiones. Desarrollar a mano algoritmos que tengan en cuenta todas estas dificultades supone un gran reto, lo que hace que los métodos como DefSLAM funcionen en un número reducido de secuencias. Con el aprendizaje profundo se aspira a tener un sistema robusto que funcione en todo tipo de secuencias.

– Para poder explotar fácilmente la información del par estéreo, hay que corregir previamente la distorsión de sus imágenes y rectificarlas.

– Uno de los retos de este trabajo es conseguir mapas de profundidad densa, es

decir, que todo píxel de la imagen de entrada tenga una profundidad estimada. Para lograrlo se va a usar aprendizaje profundo y, como la base de datos no tiene *ground truth*, hay que utilizar aprendizaje profundo no supervisado. Al no ser supervisado, la red lo tiene más complicado para aprender ya que la retroalimentación de la calidad de su desempeño es ruidosa y de señal débil.

- Estimar profundidad a partir de vídeos monoculares en lugar de estéreo, porque en las endoscopias médicas, las cuales son la aplicación futura de este trabajo, no se utilizan endoscopios estéreo. Esto trae consigo dos inconvenientes. El primero es que el problema por naturaleza es más complicado ya que no se puede aprovechar la consistencia visual de tener dos imágenes estéreo. El segundo es que, al no tener imágenes estéreo, no se puede triangular la posición 3D de los píxeles, por lo que la profundidad estimada estará a escala no métrica. Decir que durante el entrenamiento de las redes sí que se va a aprovechar el estéreo.

- Para poder evaluar el rendimiento de la red, es necesario tener *ground truth* con el que compararse. Como la base de datos utilizada no tiene, hay que crear un *pseudo-ground truth* utilizando las imágenes estéreo.

3.2. Estado del arte

El problema de la estimación de profundidad en imágenes de endoscopias ha sido tradicionalmente resuelto con métodos geométricos multivista [34]. Uno de estos métodos es DefSLAM [7], el primer sistema de SLAM visual monocular capaz de modelar la deformación del entorno y que consigue funcionar en endoscopias médicas. Al estar basado en el seguimiento y emparejamiento de puntos característicos entre imágenes de distintos instantes, la profundidad que resulta no es densa sino de unos pocos cientos de puntos independientes. En parte al ser un sistema pionero en un problema tan complicado, su precisión y robustez está limitada por varios factores.

Recientemente ha sido desarrollado un nuevo método que parte de las bases de DefSLAM. SD-DefSLAM [8] añade modelos más avanzados y usa una red neuronal convolucional para enmascarar los objetos quirúrgicos que aparecen en la escena. Gracias a estas mejoras, SD-DefSLAM lleva su rendimiento mucho más allá que DefSLAM, logrando una robustez y precisión mayores. A pesar de su mayor fiabilidad, sin embargo, la profundidad que estima sigue siendo no densa.

Gracias a los progresos en el aprendizaje profundo, existen arquitecturas que permiten trabajar en modo no supervisado. [35] es un trabajo pionero en el uso de aprendizaje profundo en imágenes de endoscopias. En concreto utiliza una red

condicional generativa-adversaria [36] para predecir profundidad densa. El problema de este tipo de redes es que, por la forma en que están diseñadas, la función de coste fuerza al modelo a aprender características que hagan la predicción plausible, en lugar de precisa. Por consiguiente, los mapas de profundidad que producen tienen apariencia realista pero en general precisión baja. Otro problema añadido es que esta red fue entrenada con imágenes sintéticas de un colon 3D recreado virtualmente y con un colon artificial hecho con un molde. Las secuencias artificiales utilizadas difieren mucho de una endoscopia médica real en muchos aspectos (como la textura, la estabilidad, los reflejos, etc.). Si a esto le sumamos el hecho de que el *ground truth* utilizado para evaluar el modelo es el colon artificial, hace que el rendimiento reportado sea cuestionable en endoscopias reales.

Existen multitud de métodos de aprendizaje profundo no supervisado que tienen un rendimiento notable, pero en vídeos de calles, que nada tienen que ver con una colonoscopia. El algoritmo de este tipo que mejor rendimiento reporta es Monodepth2 [37], que es por el que se ha apostado en este trabajo para adaptarlo y comprobar su desempeño en imágenes médicas.

3.3. Bases de datos

En este trabajo se han utilizado tres bases de datos. La primera de ellas consiste en un extensa recopilación de vídeos de calles de la ciudad alemana Karlsruhe, KITTI [38]. Ésta se ha usado porque el modelo de red neuronal del que se ha partido estaba diseñado específicamente para funcionar con esos datos. Por tanto, el uso de estos vídeos ha sido meramente para entrenar unos primeros modelos y comprobar así que la red funcionaba correctamente.

Debido a la inexistencia de bases de datos libres de colonoscopias, se utilizado la base de datos de vídeos de endoscopias *in vivo* del *Hamlyn Centre Laparoscopic* [39] del Imperial College de Londres.

La tercera base de datos empleada es $ImageNet$ [40]. $ImageNet$ es una inmensa base de datos con mas de 14 millones de imágenes de todo tipo.

3.4. Métodos

3.4.1. Libelas

Como se ha explicado anteriormente, es necesario tener algún tipo de *ground truth* para poder evaluar los errores del algoritmo de estimación de profundidad. Como la base de datos *Hamlyn* no tiene, se ha utilizado el programa Libelas (*Library for*

Efficient Large-Scale Stereo Matching [41]). Para poder trabajar con estas imágenes estéreo, éstas han sido rectificadas como se explica en la sección 2.2.6. Libelas busca los emparejamientos estéreo de todos los píxeles y devuelve las imágenes de disparidad y, como los parámetros de cámara del *Hamlyn* están disponibles, la profundidad puede obtenerse a partir de la disparidad con la ecuación 2.15.

La funcionamiento de Libelas es el siguiente. Lo primero que hace es buscar una serie reducida de puntos característicos que resultan fácilmente reconocibles en ambas imágenes. Después, conociendo la disparidad de esos puntos de apoyo, es posible por triangulación crear una malla 2D de disparidad previa que cubre la totalidad de los píxeles (ver figura 3.1). Esto ayuda a reducir las ambigüedades que le puedan surgir a un algoritmo final, encargado de emparejar los puntos en ambas imágenes a partir de la intensidad de los píxeles.

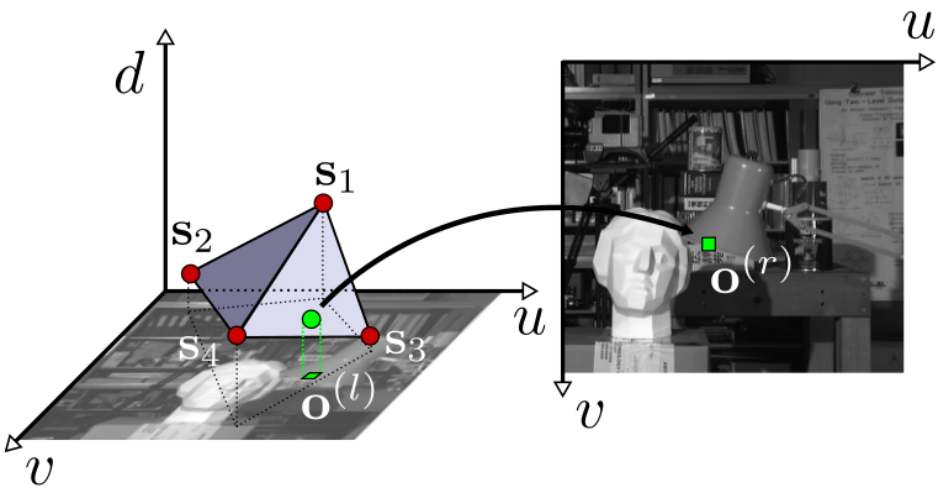

Figura 3.1: Visualización 3D de la malla de disparidad previa formada con 4 puntos de apoyo ($\boldsymbol{s_1}$, $\boldsymbol{s_2}$, $\boldsymbol{s_3}$ y $\boldsymbol{s_4}$) y del emparejamiento de un punto en la imagen derecha ($\boldsymbol{o^{(r)}}$) a partir de una observación de un punto cualquiera en la imagen izquierda ($\boldsymbol{o^{(l)}}$). Fuente: Modificado de [41] Copyright © 2011, Springer-Verlag Berlin Heidelberg.

3.4.2. Monodepth2

En este apartado se detalla la red de predicción de profundidad Monodepth2, la cual solo necesita una imagen monocular en la entrada para estimar su mapa de profundidad. Esta red tiene tres tipos de entrenamiento no supervisado posibles: estéreo (usa la consistencia espacial conocida de las cámaras estéreo), monocular (utiliza la consistencia temporal que existe en las imágenes consecutivas de un vídeo) y estéreo-monocular (aprovecha consistencia espacial y temporal).

Entrenamiento no supervisado

La idea básica es entrenar una red neuronal que prediga la apariencia de una imagen objetivo I_t desde el punto de vista de otra imagen origen $I_{t'}$. Restringiendo la red a calcular la imagen objetivo usando una variable intermedia, aquí profundidad o disparidad, se puede extraer el modelo de profundidad buscado. Este problema es por naturaleza ambiguo, existen infinidad de profundidades por píxel que reconstruyen correctamente la imagen objetivo dada la posición relativa entre ambas vistas. La posición relativa de cada de cada punto de vista origen respecto la posición de la imagen objetivo se expresa como $\boldsymbol{E}_{t\to t'}$ (matriz de transformación rígida total de la ecuación 2.12). El problema se resuelve calculando el mapa de profundidad denso D_t que minimiza el error de reproyección fotométrico por píxel L_p

$$L_p = \min_{t'} pe(I_t, I_{t\to t'}) \tag{3.1}$$

$I_{t\to t'}$ es la imagen reconstruida que ha de parecerse a la imagen objetivo a partir de la imagen origen $I_{t'}$, el mapa denso de profundidad estimada D_t, la posición relativa de ambas imágenes $\boldsymbol{E}_{t\to t'}$ y la matriz de calibración de la cámara $\boldsymbol{K}$

$$I_{t\to t'} = I_{t'}\langle proj(D_t, \boldsymbol{E}_{t\to t'}, \boldsymbol{K})\rangle \tag{3.2}$$

siendo $proj(D_t, \boldsymbol{E}_{t\to t'}, \boldsymbol{K})$ las coordenadas 2D de todos los píxeles de la imagen origen $I_{t'}$ proyectados sobre el punto de vista objetivo en función de la profundidad estimada por la red para cada píxel D_t, la matriz de calibración $\boldsymbol{K}$ y la matriz de transformación rígida total $\boldsymbol{E}_{t\to t'}$, tal y como se explica en el apartado 2.2.4. $\langle\rangle$ es el operador de muestreo, encargado de formar una nueva imagen donde el color de sus píxeles viene de buscar en la imagen origen $I_{t'}$ el color de los píxeles indicados por $proj()$.

En la ecuación 3.1, pe es el error de reconstrucción fotométrico, que se calcula como

$$pe(I_t, I_{t\to t'}) = \frac{\alpha}{2}(1 - SSIM(I_t, I_{t\to t'})) + (1-\alpha)\|I_t - I_{t\to t'}\|_1 \tag{3.3}$$

Esta función devuelve el grado de similitud entre las dos imágenes. Cuanto mayor es su valor, peor es la calidad de la imagen reconstruida. $\alpha = 0{,}85$, $\|I_t - I_{t\to t'}\|_1$ es la distancia L1 de la 2.19 y SSIM [42] es el índice de similitud estructural

$$SSIM(I_t, I_{t\to t'}) = \frac{(2\mu_t\mu_{t\to t'} + c_1)(2\sigma_{t,t\to t'} + c_2)}{(\mu_t^2 + \mu_{t\to t'}^2 + c_1)(\sigma_t^2 + \sigma_{t\to t'}^2 + c_2)} \tag{3.4}$$

que mide el cambio estructural de la información entre ambas imágenes. Este índice va de 0 a 1, siendo 1 cuando las imágenes son iguales. μ es el valor medio, σ^2 la varianza,

y c_1 y c_2 son

$$\begin{cases} c_1 = (k_1 L)^2 \\ c_2 = (k_2 L)^2 \end{cases} \tag{3.5}$$

con $k_1 = 0{,}01$, $k_2 = 0{,}03$ y L el rango dinámico de valores de cada píxel, en este caso, como las imágenes son de 8 bits, $L = 2^8 - 1 = 255$.

Cuando se calcula el error de reproyección L_p puede ocurrir que algunos píxeles que son visibles en la imagen objetivo no lo sean en alguna de las imágenes origen, ya sea porque son píxeles que se salen del campo de visión o porque quedan ocluidos por un objeto de la escena. Cuando las correspondencias son buenas, el error de reproyección L_p debe ser bajo (ver figura 3.2). Si para el cálculo de L_p se promediara el *pe* de todas las imágenes origen, se obligaría a la red a hacer coincidir los píxeles ocluidos resultando en una estimación de profundidad borrosa. Mientras que si se coge para cada píxel el *pe* mínimo (ecuación 3.1), se obliga a hacer coincidir cada píxel con la vista en la que es visible, lo que genera resultados más nítidos, bordes más afilados y mejor precisión (ver figura 3.3). La imagen reconstruida se forma con los píxeles de aquellas imágenes origen donde el *pe* respecto a la imagen objetivo es menor.

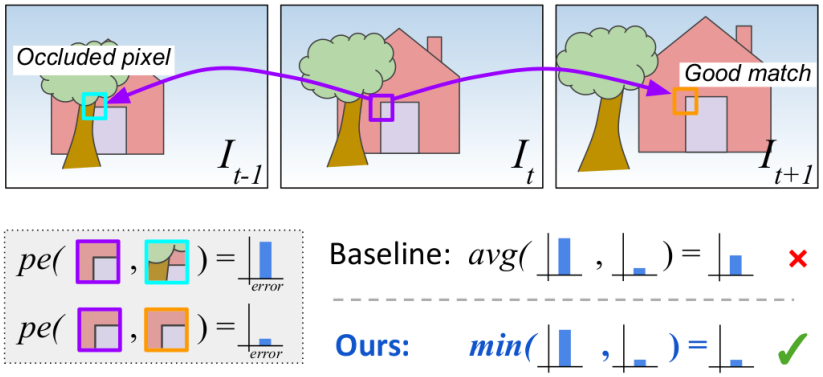

Figura 3.2: Error de reconstrucción fotométrico *pe* para entrenamiento monocular cuando algunos píxeles en la imagen objetivo I_t son visibles en la imagen del instante posterior I_{t+1}, pero no en la imagen anterior I_{t-1}. En este ejemplo, la profundidad predicha es correcta porque el emparejamiento de los píxeles entre imágenes es correcto. Por tanto, el error de reproyección L_p debería ser bajo. Si para el cálculo de L_p se hace la media de los errores fotométricos de ambos píxeles, L_p tendrá un valor alto cuando debería ser bajo. Quedándose L_p con el *pe* mínimo, L_p es bajo y, por tanto, no se penaliza negativamente a la red. Fuente: Modificado de [37] Copyright © 2019 IEEE.

Aparte del error de reproyección fotométrico L_p, otro componente de la función de coste total encargado del suavizado de píxeles es

$$L_s = |\delta_x d_t^*| e^{-|\delta_x I_t|} + |\delta_y d_t^*| e^{-|\delta_y I_t|} \tag{3.6}$$

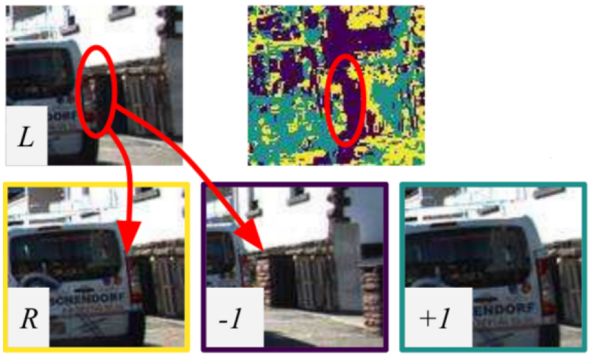

Figura 3.3: Beneficio de usar el error de reproyección mínimo en entrenamiento estéreo-monocular. Los píxeles en la región redondeada de la imagen objetivo I_L se encuentran ocluidos en I_R, así que nos se aplica error entre I_L y I_R. En su lugar, los píxeles son relacionados a I_{t-1} donde sí que son visibles. Los colores en la imagen superior derecha indican de cual de las imágenes origen de la parte inferior proviene cada píxel seleccionado por la ecuación 3.1 para contribuir al valor final de la función de coste. Fuente: Modificado de [37] Copyright © 2019 IEEE.

donde $d_t^* = d_t/\overline{d_t}$ es la profundidad inversa media normalizada [43], $\delta_x d_t^*$ y $\delta_y d_t^*$ son los gradientes de d_t^* en las direcciones x e y, y $\delta_x I_t$ y $\delta_y I_t$ los gradientes de color en las direcciones x e y, respectivamente. L_s impone que la reconstrucción sea suave a excepción de los bordes. Los bordes son cambios de profundidad bruscos que suelen ir acompañados de un fuerte gradiente de color. Este error es mayor, es decir, penaliza cuando la red predice un gradiente de profundidad elevado en unos píxeles que no presentan un cambio importante en la intensidad de sus colores en la imagen objetivo I_t. Si no se normalizara d_t, dado un mapa de profundidad inversa, siempre se podría construir un nuevo mapa de profundidad inversa reduciendo su escala (y actualizando la posición en consecuencia), lo cual resultaría en un valor de función de coste menor hasta saturar en cero la profundidad inversa predicha.

En entrenamiento estéreo, la imagen origen I_t' es el otro punto de vista de la pareja estéreo que hace con la imagen objetivo I_t, de la cual se conoce su posición. Sin embargo, en entrenamiento monocular la posición relativa entre imágenes origen y objetivo no es conocida. Por eso se entrena una segunda red neuronal encargada de predecir las posiciones relativas $\boldsymbol{E}_{t\to t'}$ usadas en la función de reproyección $proj()$. Durante el entrenamiento, las redes de posición de cámara y de profundidad son procesadas simultáneamente para minimizar Lp. Durante el entrenamiento monocular, las imágenes origen son las imágenes temporalmente adyacentes a la imagen objetivo I_t, es decir, $I_{t'} \in \{I_{t-1}, I_{t+1}\}$. En entrenamiento estéreo-monocular, $I_{t'}$ son las imágenes temporalmente adyacentes a I_t y su punto de vista opuesto del par estéreo.

El entrenamiento monocular no supervisado asume que en un vídeo la cámara está en constante movimiento y el entorno es estático. Cuando esto no ocurre, como

cuando la cámara se queda quieta o hay un objeto moviéndose, la red no funciona correctamente. Si esto ocurre, la red puede, por ejemplo, predecir mapas con zonas de píxeles que tengan infinita profundidad para aquellos objetos que durante el entrenamiento solían estar en movimiento. Para solventar este problema, se utiliza una máscara que filtra los píxeles cuya apariencia no cambia entre imágenes consecutivas. Esto evita que los píxeles de objetos que se mueven a la misma velocidad que la cámara, de imágenes donde la cámara permanece quieta o de zonas con baja textura se tengan en cuenta en el cálculo de la función de coste (ver figura 3.4). En la práctica, consiste en la aplicación de una máscara binaria $\mu \in \{0, 1\}$ que se multiplica a L_p para que solo contribuyan a la función de coste aquellos píxeles donde el error de reproyección L_p de la imagen reconstruida $I_{t \to t'}$ respecto la imagen objetivo I_t sea menor que el de la imagen origen $I_{t'}$ respecto a la imagen objetivo I_t, es decir

$$\mu = [\min_{t'} pe(I_t, I_{t \to t'}) < \min_{t'} pe(I_t, I_{t'})] \tag{3.7}$$

siendo [] el corchete de Iverson, una generalización de la delta de Kronecker que devuelve 1 si su argumento es verdadero y 0 si es falso.

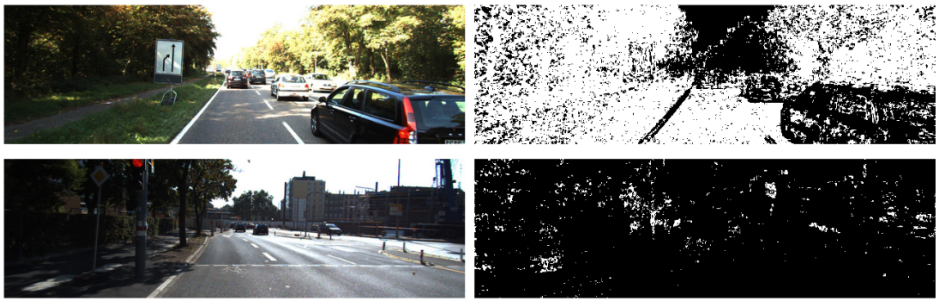

Figura 3.4: Funcionamiento de la máscara de píxeles estacionarios. Los píxeles negros son eliminados del cálculo de la función de coste, es decir, $\mu = 0$. La máscara previene que los objetos moviéndose a velocidades similares a la cámara (arriba) y todas las imágenes donde la cámara permanece estática (abajo) contaminen la función de coste. Fuente: [37] Copyright © 2019 IEEE.

Para evitar que el entrenamiento se estanque en un mínimo local, la función de coste total es la combinación de costes individuales localizados en cada escala del *decoder* de la red de predicción de profundidad. Primero se aumenta la resolución de los mapas de profundidad predichos en las capas intermedias de baja resolución para hacerlos coincidir con el tamaño de la imagen de entrada. Entonces, se reconstruye la imagen objetivo y se obtiene el error fotométrico *pe* (ver figura 3.5). Este procedimiento restringe los mapas de profundidad de cada capa a trabajar para conseguir el mismo objetivo, es decir, reconstruir la imagen objetivo a la resolución de entrada de la manera más precisa posible.

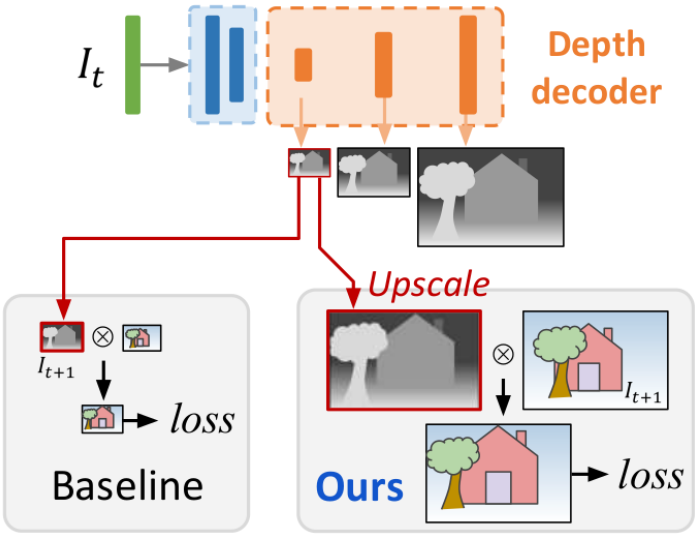

Figura 3.5: Proceso de aumento de la resolución de los mapas de profundidad predichos en capas intermedias del *decoder* antes de reconstruir junto a la imagen origen (aquí I_{t+1}) la imagen reproyectada que será comparada con la imagen objetivo (I_t) mediante el error de reproyección que contribuye a la función de coste o *loss* (ours), frente al método tradicional en el que no se aumenta la resolución (baseline). Fuente: modificado de [37] Copyright © 2019 IEEE.

Combinando el error de suavizado de píxeles y el error enmascarado de reproyección forométrica, la función de coste es

$$L = \mu L_p + \lambda L_s \tag{3.8}$$

donde λ es una constante empírica de suavizado de valor 10^{-3}.

Para mejorar la robustez de la red ante pequeñas variaciones en las imágenes de entrada y para reducir la aparición de sobreajuste durante el entrenamiento, se aumenta la cantidad de datos de entrada aplicando una serie de cambios a las imágenes de entrenamiento. En concreto, se realizan variaciones aleatorias de brillo, contraste, saturación y color a una de las capas RGB de la imagen.

Arquitectura

La red de predicción de profundidad se basa en la arquitectura general de U-Net (subsección 2.3.6). Como *encoder* se usa una ResNet18 como la de la figura 2.14 pero sin la primera capa convolucional y la de agrupación máxima (*pool*), ni última capa totalmente conectada. La red usada tiene 11 millones de parámetros, funciones de activación sigmoidales a la salida de las capas donde se extrae el mapa de profundidad predicho para usarlo en la función de coste y ELU a la salida del resto de capas. La red aprende a predecir disparidad, que puede asemejarse a profundidad inversa escalada

según la ecuación 2.15. Para convertir la salida sigmoidal de disparidad de la red σ en profundidad escalada a un rango de 0.1 a 100 unidades D se aplica la siguiente fórmula

$$D = \frac{1}{a\sigma + b} \tag{3.9}$$

siendo a el rango de disparidad $(1/0{,}1 - 1/100)$ y b la disparidad mínima $(1/100)$.

La red encargada de estimar la posición durante el entrenamiento monocular y estéreo-monocular consiste en una ResNet18 como la del *encoder* de predicción de profundidad pero modificado para admitir dos imágenes (6 canales de color) como entrada en lugar de una y devolver a la salida la posición relativa entre ambas. Esta modificación consiste en una primera capa convolucional cuyos pesos tienen una forma de 6x64x3x3, en lugar de 3x64x3x3 que tiene el *encoder* de la red de predicción de profundidad que es el estándar de la ResNet18. El 6 viene del número de canales de color entre ambas imágenes de entrada, 64 es el número de mapas de características que tiene esa primera capa, y el 3x3 es el tamaño del kernel. Todos los pesos de esta primera capa añadida se dividen por dos para tener la misma forma que la ResNet18 por defecto. Sus capas utilizan ReLU como función de activación.

La red de Monodepth2 tal y como está diseñada requiere que todas sus imágenes de entrenamiento tengan la misma resolución y los mismos parámetros de cámara intrínsecos y extrínsecos. Como la base de datos *Hamlyn* que aquí se ha utilizado contiene vídeos con diferentes resoluciones y calibraciones, se ha tenido que rediseñar el código para poder entrenar con todas las secuencias. Los vídeos que tuvieran resolución similar y parámetros de cámara parecidos se han agrupado. Si toda la base de datos tuviera imágenes iguales, la forma de entrenar sería tal que se irían seleccionando imágenes aleatorias de cualquiera de los vídeos y cuando se hubieran usado todas las imágenes de la base de datos se habría completado una época. Los parámetros de cámara y la resolución de las imágenes se cargan antes de comenzar el entrenamiento y, por tanto, no se pueden modificar dinámicamente en medio del entrenamiento. Para poder entrenar con los cinco grupos de vídeos diferentes, se ha hecho que la red se entrene primero en un único grupo de vídeos y que al terminar de ver todas sus imágenes se guarde el modelo. Después, se carga el modelo guardado pero con los parámetros del nuevo grupo de vídeos en el que va a entrenarse la red. Cuando se han usado todos los grupos se dice que la red ha completado una época.

Capítulo 4

Validación experimental

En este capítulo se van a detallar todos los experimentos realizados con la red modificada a partir de Monodepth2 (de aquí en adelante EndoDepth para diferenciarla de la red de Monodepth2 original), con el objetivo de mejorar lo máximo posible la estimación de profundidad sobre los vídeos de endoscopias del dataset *Hamlyn*. En la sección de modelos entrenados se exponen todas los modelos que han resultado relevantes en la investigación. En la sección de experimentos se comentan las conclusiones extraídas a raíz de todos los modelos entrenados y se comparan sus errores de estimación de profundidad con los del estado del arte.

4.1. Modelos entrenados

Antes de entrenar un modelo con la base de datos *Hamlyn*, ésta ha sido dividida en tres subconjuntos: entrenamiento, validación y test. El subconjunto de test está formado por todas las imágenes de un vídeo en concreto que será usado para evaluar los errores de estimación del modelo. El vídeo de test puede variar entre un modelo y otro para así evaluar el rendimiento de la red de una manera más objetiva. Los cinco vídeos de endoscopias del *Hamlyn* usados aquí para test son: una secuencia estática con un corazón en movimiento con intrusión de herramientas (*corazón* o vídeo 4 del *Hamlyn*) y cuatro secuencias en movimiento (*abdomen*, *órganos*, *exploración* y *víscera*, o vídeos 1, 19, 20 y 21 del *Hamlyn*, respectivamente). Los subconjuntos de entrenamiento y validación están formados por las imágenes del resto de vídeos, habiendo sido escogidas de manera aleatoria y conteniendo el 90 % y el 10 % de las imágenes, respectivamente. El cometido de los datos de validación es indicarnos el desempeño del modelo mientras se entrena con los datos de entrenamiento y así saber en qué época parar. Conforme se va entrenando durante más épocas, llega un momento en el que el desempeño del modelo no mejora e incluso empeora (*overfitting*), indicando la época en la que detenerse.

En total se han entrenado 17 modelos diferentes, los cuales aparecen reflejados en

las tablas 4.1 y 4.2. Los primeros 10 modelos de la tabla 4.1 y los 3 de la tabla 4.2 han sido entrenados de maneras diferentes, pero utilizan el mismo vídeo *órganos* como test para poder evaluar la influencia de cada parámetro durante el entrenamiento y, de esta manera, extraer las conclusiones de los experimentos que se comentarán en la siguiente subsección 4.2. Los modelos de la tabla 4.1 tienen una serie de variables que hacen que el entrenamiento sea diferente. El primer parámetro que se puede variar es la forma de entrenamiento no supervisado de la red. Como ya hemos comentado en 3.4.2, EndoDepth puede entrenarse de manera estéreo, monocular y estéreo-monocular. También se puede variar la resolución del modelo, es decir, la resolución a la que se reescalan las imágenes de entrenamiento antes de procesarse por la red. La forma en que la red reconstruye la imagen objetivo a partir de las imágenes origen es otra variable que se puede modificar. Cuando se reproyectan los píxeles de una imagen origen sobre la cámara objetivo siempre ocurre que algunos caen fuera del plano imagen porque el punto de vista es diferente. Los píxeles que quedan vacíos pueden rellenarse con ceros (relleno *ceros*) o con el valor del píxel más cercano que sí tiene color (relleno *borde*). El *encoder* de la red de predicción de profundidad y el de la red de estimación de la posición pueden entrenarse con los pesos inicializados aleatoriamente o con los pesos de una red ResNet-18 preentrenada en *ImageNet*. Debido a la inmensidad de *ImageNet*, en lugar de entrenar la ResNet-18, ha sido ha sido descargada ya preentrenada del repositorio oficial de PyTorch [44]. Los modelos pueden entrenarse con las bases de datos *KITTI* y *Hamlyn*. Si un modelo de la tabla 4.1 tiene marcadas las casillas de *KITTI* y *Hamlyn*, primero se ha entrenado en *KITTI* y luego en *Hamlyn*. Por último, cada modelo se ha entrenado durante un número de épocas diferente.

Los modelos de la tabla 4.2 han sido todos entrenados con el vídeo *órganos* de manera monocular durante una sola época, con la misma resolución y tipo de relleno. Los tres modelos habían sido previamente entrenados en el *Hamlyn* de manera estéreo, monocular o estéreo-monocular, pero sin usar el vídeo de *órganos*. A este proceso de entrenar con pocos datos una red que había sido preentrenada se denomina tuneo[1].

La evaluación numérica es posible porque se utiliza la profundidad calculada con *Libelas* (en mm) a modo de *ground truth* con el que comparar la estimación de los modelos entrenados. Hay que tener en cuenta que Libelas no es perfecto, por lo que la profundidad estimada en verdad es *pseudo-ground truth*. Libelas suele dejar píxeles sin profundidad en los bordes de las imágenes e incluso dentro de ellas porque no es capaz de emparejar ciertos puntos estéreo. Además, aplica un suavizado a sus mapas de profundidad, por lo que puede que se tenga en cuenta de manera negativa los

[1]La palabra tuneo es un anglicismo de la palabra *fine-tuning*. Tuneo viene del verbo tunear, aceptado por la RAE, que significa 'adaptar algo a los gustos o intereses personales'.

mapas de profundidad de EndoDepth si estos tienen un nivel de detalle mayor aunque sea correcto. La profundidad estimada por los modelos de EndoDepth se encuentra escalada a un rango de 0.1 a 100 unidades no métricas. Para conseguir escala métrica hay que multiplicar las predicciones por un factor de escala. Este factor de escala es individual para cada mapa de profundidad estimado y se calcula como

$$factor_{local} = \frac{mediana(gt)}{mediana(pred)} \tag{4.1}$$

siendo $mediana(gt)$ y $mediana(pred)$ la mediana del mapa de profundidad *ground truth* (gt) y del mapa de profundidad estimado ($pred$), respectivamente. Este factor de escala local ha sido usado para obtener los mapas de profundidad estimada en escala métrica (mm), necesarios para poder comparar con los del *ground truth* que están en mm y así extraer las métricas de rendimiento de las tablas 4.1, 4.2 y de la columna de EndoDepth local de la tabla 4.4. En el caso de haber entrenado en modo estéreo o estéreo-monocular, la red asume una distancia entre cámaras fija de 0.1 unidades. Entonces, una buena aproximación del factor de escala global que hay que aplicar es la separación real entre las cámaras usadas durante el entrenamiento, multiplicada por 10 para compensar el 0.1. Como cada vídeo del *Hamlyn* tiene unos parámetros de cámara ligeramente diferentes, se ha calculado una media ponderada que tiene en cuenta el número de imágenes de cada vídeo. Esta distancia entre cámaras promedio tiene un valor de 5.2864 mm, y al multiplicarlo por 10 obtenemos el factor de escala global con el que se puede obtener profundidad en escala métrica (en mm concretamente al estar la separación entre cámaras en esas unidades) sin hacer uso del *ground truth*. La columna de EndoDepth global de la tabla 4.4 hace uso de este factor.

Siete son las métricas utilizadas para evaluar cuantitativamente el rendimiento de los modelos entrenados al comparar todos los mapas de profundidad predichos a partir de las imágenes del vídeo de test con los mapas del *ground truth*.

- Error absoluto relativo medio

$$Abs\ rel = media\left(\frac{|gt - pred|}{gt}\right) \tag{4.2}$$

- Error cuadrático relativo medio (mm)

$$Sq\ Rel = media\left(\frac{(gt - pred)^2}{gt}\right) \tag{4.3}$$

- Raíz del error cuadrático medio (mm)

$$RMSE = \sqrt{media((gt - pred)^2)} \tag{4.4}$$

– Raíz del error logarítmico cuadrático medio (log(mm))

$$RMSE\ log = \sqrt{media((log(gt) - log(pred))^2)} \tag{4.5}$$

– Tanto por uno de puntos por debajo de un determinado umbral de error

$$umbral = \text{máx}\left(\frac{gt}{pred}, \frac{pred}{gt}\right) \tag{4.6}$$

$$\delta < 1{,}25\ = media(umbral < 1{,}25) \tag{4.7}$$

$$\delta < 1{,}25^2\ = media(umbral < 1{,}25^2) \tag{4.8}$$

$$\delta < 1{,}25^3\ = media(umbral < 1{,}25^3) \tag{4.9}$$

De entre estos siete errores los más útiles son el $RMSE$ y $\delta < 1{,}25$. El $RMSE$ da una idea de cuantos mm suele diferir la profundidad predicha respecto a su valor correcto, perjudicando si hay píxeles con errores atípicos, mucho mayores que la media (*outliers*). $\delta < 1{,}25$ devuelve el tanto por uno de píxeles cuya profundidad predicha entra dentro del rango de unos $\pm 25\,\%$ respecto a su profundidad *ground truth*.

4.2. Experimentos

Comparación con profundidad promedio

Para comprobar que las redes neuronales entrenadas consiguen aprender patrones, se ha diseñado un algoritmo muy básico que siempre predice la profundidad promedio del *Hamlyn*, independientemente de la imagen de entrada. Los errores métricos de este algoritmo respecto a los cinco vídeos de test del *Hamlyn* pueden encontrarse en la tabla 4.3. El error $\delta < 1{,}25$ es prácticamente cero en todos los vídeos de test porque este algoritmo produce un mapa de profundidad plano que no se ajusta absolutamente nada a la realidad. Todos los modelos entrenados tienen un $\delta < 1{,}25$ mayor al $60\,\%$, lo que indica que todos nuestros modelos son capaces de aprender patrones en la imagen que están relacionados con la profundidad. El $RMSE$ obtenido con este algoritmo sirve de referencia para saber cuan bueno es el $RMSE$ de un modelo usando un vídeo de test en concreto, ya que un valor de $RMSE$ que es considerado bueno para un vídeo de test puede no serlo si se usa otro. Por ejemplo, este efecto puede observarse entre los modelos 8 y 14 que han sido entrenados de la misma manera pero con diferente vídeo de test (*órganos* y *víscera*) y que tienen un $RMSE$ de 4.90mm y de 15.01mm, respectivamente. Podría parecer que entrenar los modelos de esta manera funciona bien en vídeos del tipo de *órganos* pero no tan bien si son del tipo de *víscera*. Por eso es mejor extraer conclusiones del rendimiento comparando respecto al $RMSE$ del algoritmo aquí presentado.

nº	Modo	Resolución	Relleno	ImgNet	KITTI	Hamlyn	Épocas	Vídeo test	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1{,}25$	$\delta < 1{,}25^2$	$\delta < 1{,}25^3$
1	E	640x192	borde		✓		20	Órganos	0.18	2.457	10.424	0.222	0.673	0.943	0.997
2	E	640x192	borde	✓	✓		20	Órganos	0.152	1.801	8.961	0.185	0.763	0.981	0.999
3	E	352x128	borde	✓	✓	✓	1	Órganos	0.079	0.553	4.935	0.099	0.959	0.998	1
4	E	352x128	borde			✓	2	Órganos	0.076	0.529	4.853	0.099	0.957	0.998	1
5	E	352x128	borde	✓		✓	1	Órganos	0.085	0.68	5.376	0.107	0.94	0.997	1
6	E	352x128	ceros	✓		✓	5	Órganos	0.092	0.808	5.834	0.116	0.927	0.995	1
7	E	704x288	borde			✓	5	Órganos	0.089	0.771	5.686	0.112	0.937	0.994	1
8	E	704x288	borde	✓		✓	2	Órganos	0.077	0.576	4.902	0.098	0.957	0.996	1
9	M	704x288	borde	✓		✓	2	Órganos	0.1	0.857	6.26	0.128	0.907	0.997	1
10	EM	704x288	borde	✓		✓	9	Órganos	0.09	0.746	5.81	0.115	0.938	0.996	1
11	E	640x480	borde	✓		✓	4	Abdomen	0.104	1.625	11.194	0.135	0.897	0.989	0.998
12	E	352x288	borde	✓		✓	2	Corazón	0.04	0.169	2.416	0.054	0.987	0.999	1
13	E	704x288	borde	✓		✓	1	Exploración	0.134	2.429	10.903	0.172	0.851	0.961	0.993
14	E	704x288	borde	✓		✓	2	Víscera	0.171	3.611	15.01	0.215	0.721	0.945	0.99

Tabla 4.1: Evaluación de modelos de EndoDepth entrenados de múltiples formas. Los modelos 1 y 2 usan la red Monodepth2 original. *Modo* indica el tipo de entrenamiento: estéreo (E), monocular (M) o estéreo-monocular (EM). Los modelos entrenados también varían en resolución, tipo de relleno durante la reproyección, bases de datos utilizadas durante el entrenamiento (*ImageNet*, *KITTI* y *Hamlyn*), número de épocas de entrenamiento y vídeo de test sobre el que se evalúan todas las métricas, usando un factor de escala individual para cada imagen.

nº	Modo previo	Épocas	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1{,}25$	$\delta < 1{,}25^2$	$\delta < 1{,}25^3$
15	E	1	0.113	1.051	6.818	0.142	0.874	0.991	1
16	M	1	0.06	0.316	3.797	0.078	0.982	1	1
17	EM	1	0.084	0.682	5.543	0.111	0.943	0.996	1

Tabla 4.2: Evaluación de modelos de EndoDepth 8, 9 y 10 de la tabla 4.1, entrenados de manera estéreo (E), monocular (M) y estéreo-monocular (EM), y posteriormente tuneados de manera monocular con el vídeo *órganos* durante una sola época. Todos los modelos tienen la misma resolución 704×288, mismo modo de relleno (borde) y usan el mismo vídeo de test (*órganos*). Evaluación realizada con factor de escala local para cada imagen.

Vídeo test	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1{,}25$	$\delta < 1{,}25^2$	$\delta < 1{,}25^3$
Abdomen	0.629	33.874	52.915	1.03	0	0.005	0.102
Corazón	0.35	5.747	15.919	0.442	0.006	0.55	0.979
Exploración	0.509	17.129	32.734	0.748	0.008	0.07	0.376
Víscera	0.572	27.262	46.996	0.926	0	0.036	0.277
Órganos	0.396	8.437	20.431	0.531	0.028	0.322	0.809

Tabla 4.3: Rendimiento de un algoritmo que siempre predice la profundidad media del *Hamlyn* para todos los píxeles, evaluado sobre cinco secuencias de test del *Hamlyn*.

Efecto de entrenar con datos parecidos a los de test

La base de datos *KITTI* poco tiene que ver con la del *Hamlyn*, porque la primera contiene imágenes de calles y la segunda de endoscopias. Aunque el objetivo común sea el mismo, estimar profundidad, si la red neuronal ha sido entrenada con *KITTI* habrá aprendido los patrones visuales correspondientes a las calles. La red, al enfrentarse a una imagen de test, tendrá una predisposición a situar una carretera en la parte inferior con edificios a los lados que aumentan su profundidad hacia un punto de fuga colocado en el centro de la imagen y un cielo arriba con una profundidad monótona muy lejana (ver figura 4.1). Sin embargo, como las imágenes de test aquí utilizadas son de secuencias médicas, todas esas características intrínsecas de las calles aprendidas con *KITTI* no solo no sirven para estimar la profundidad de imágenes del *Hamlyn*, sino que además pueden confundir a la red y desencadenar un resultado desastroso.

Figura 4.1: Imagen de la base de datos *KITTI*. Fuente: *KITTI Vision Benchmark* [38].

Cualitativamente se puede comprobar con los mapas estimados por los modelos 1 y 2 de la tabla 4.2 que nada tienen que ver con el *ground truth*. Con las imágenes del modelo 1 puede verse como la red devuelve siempre una zona más profunda en la zona superior central de la imagen, como si de cielo se tratara, y profundidad más cercana a los lados y en la parte baja, ya que esperaba ver edificios y una carretera. El resto de modelos que sí han sido entrenados en el *Hamlyn* puede verse como consiguen captar, de un mejor o peor modo, la profundidad de la imagen.

Podemos evaluar este efecto fijándonos en los modelos 2 y 3 en la tabla 4.1. El modelo 2 fue entrenado con el *KITTI* y el 3 partió de ese modelo para reentrenar en el *Hamlyn*, consiguiendo así un rendimiento mucho mayor. Curiosamente al comparar el rendimiento de los modelos 3 y 5, puede verse como preentrenar en *KITTI* antes de entrenar en *Hamlyn* ha resultado beneficioso. Este resultado tiene relación con los siguientes dos experimentos.

Sobreajuste

Al fijarse en el número de épocas que ha durado el entrenamiento de cada modelo de EndoDepth que aparecen en la tabla 4.1, queda claro que EndoDepth necesita muy pocas épocas para converger antes de producirse un sobreajuste negativo (*overfitting).*

Las redes neuronales diseñadas para trabajar con el *KITTI* suelen entrenarse durante decenas e incluso centenas de épocas. Esta es una de las razones por la que en el modelo 3 aun habiendo entrenado en *KITTI*, como luego se entrena en *Hamlyn*, el rendimiento no es malo respecto a no haber sido preentrenado en *KITTI* (modelo 5). Esto es debido a que EndoDepth sobreajusta rápidamente a los datos de entrenamiento, olvidando su prejuicio de calles y adaptándose a las imágenes médicas. Esta característica tiene dos consecuencias positivas y una negativa. La parte buena es que la red necesita pocos datos para ser entrenada y requiere de poco tiempo de procesamiento. Para contextualizar, un modelo estéreo con resolución 704x288 entrenado durante 10 épocas dura entorno a las 12 horas, si es monocular unas 15 horas y en el caso estéreo-monocular puede alcanzar las 35 horas, en una tarjeta gráfica Nvidia RTX 2080 Ti. La parte negativa de este rápido sobreajuste es que al entrenar en una gran base de datos las últimas imágenes con las que se entrena durante una época tienen mayor influencia que las primeras imágenes con la que se entreno. Es decir, la red tiende perder la información previamente aprendida cuando nueva información es aprendida. A este efecto se le conoce como *olvido catastrófico* [45] y hay que tratar de evitarlo en la medida de lo posible. El uso de la arquitectura de red tipo U-Net, en lugar de autocodificador (figura 2.15), ayuda a minimizarlo.

Inicialización aleatoria del optimizador

El algoritmo optimizador de las redes neuronales, aquí Adam, se inicializa con unos valores aleatorios. Esto causa que cada vez que se entrena un nuevo modelo, aunque sea con los mismos parámetros y datos, el rendimiento final tanto cualitativo como cuantitativo va a ser diferente. Esta variación suele ser ligera, en los modelos que aquí han sido reentrenados, normalmente el *RMSE* no varía más de 1mm. Pero puede darse el extraño caso de que una mala inicialización conduzca a la red a un mínimo local y la red no consiga converger adecuadamente, resultando en valores de *RMSE* notablemente más altos.

Efecto de preentrenar en ImageNet

Los autores de Monodepth2 y de trabajos previos ([46], [47], [48]) defienden inicializar los *encoder* con los pesos preentrenados en *ImageNet*. Sus redes neuronales siempre consiguen unos resultados ligeramente mejores. Aquí también se ha querido experimentar para ver si se conseguían mejores rendimientos. Al fijarse en los errores métricos de los modelos 1 y 2 en los que la única diferencia es que el 2 ha sido preentrenado en *ImageNet*, a pesar de no haber sido entrenados en *Hamlyn*, se aprecia una ligera mejora del error al preentrenar en *ImageNet*. Esto no ocurre cuando en lugar

de entrenar en *KITTI* se hace en *ImageNet*. Comparando los modelos 4 y 5, donde el modelo 4 no ha sido preentrenado con *ImageNet*, se muestra un rendimiento un poco superior al no preentrenar. Para comprobar que esto no era fruto de una mala inicialización por parte del modelo 5, se ha reentrenado tres veces bajo las mismas condiciones y el resultado siempre ha sido similar. La explicación de por qué preentrenar en *ImageNet* mejora el resultado si la red ha sido entrenada en *KITTI*, pero no si lo ha sido en *Hamlyn*, puede deberse al hecho de que las imágenes de *ImageNet* son más parecidas a las del vídeo de test *órganos* que las del *KITTI* y las del *Hamlyn* son más parecidas que las de *ImageNet*. Esto se traduce en que *ImageNet* ayuda al modelo 2 a ser mejor que el 1, pero confunde a la red del modelo 5 respecto al 4. Sin embargo, si nos fijamos en los modelos 7 y 8, preentrenar en *ImageNet* sí que resulta fructífero. La única diferencia con los anteriores modelos 4 y 5 es que aquellos tienen una resolución menor. Podría extrapolarse la conclusión de que al usar imágenes con una resolución mayor, *ImageNet* ayuda a las redes a converger mejor. Lo que queda claro es que las redes neuronales tienen un comportamiento incierto, la mayoría de las conclusiones se extraen de manera empírica y a veces resulta difícil generalizar. Otros autores diferentes a los de Monodepth2 prefieren no preentrenar en *ImageNet* en sus trabajos y aquí se ha comprobado que no siempre se traduce en un mejor rendimiento. Tendrá que estudiarse cada caso en particular para ver si resulta beneficiosa su aplicación.

Efecto del tipo de relleno

Los autores de Monodepth2 defienden el uso de relleno tipo borde porque reduce las distorsiones visuales en los bordes de las imágenes. Con EndoDepth esto parece no ocurrir. Si se comparan las imágenes de los modelos 5 (relleno borde) y 6 (relleno ceros) de la figura 4.2, los bordes son similares con ambos modelos. Pero sí que se observa que al usar relleno con ceros, la red no captura tan bien la forma de algunos bultos de la escena como sí hace con relleno de borde, sobretodo en las imágenes de la segunda y tercera columna. Además, al comparar ambos modelos en la tabla 4.1, se comprueba que el uso de relleno tipo borde devuelve unos errores métricos mejores.

Efecto de la resolución

En la tabla 4.1 puede verse que la única diferencia entre los modelos 5 y 8 es la resolución, siendo la de éste último mayor. Se comprueba que al usar imágenes con una resolución mayor se obtienen mejores métricas, como defienden en [49]. El efecto de usar una resolución mayor también puede observarse en las imágenes predichas por los modelos 5 y 8 que aparecen en la figura 4.2. Con ambas resoluciones se recupera la forma de la escena pero es solo en mayores resoluciones en las que la red es capaz

de aprender a detectar los detalles más pequeños y producir imágenes más nítidas. La razón por la que los bordes aparecen más definidos al usar mayor resolución es porque la función de error L_s (ecuación 3.6) impone un suavizado a todas las zonas del mapa de profundidad menos en las que detecta un cambio brusco de color. Esta característica resulta beneficiosa en las imágenes del *KITTI*, pero no siempre en las médicas del *Hamlyn*. Esta L_s permite delimitar mejor los bordes de los órganos, pero resulta perjudicial donde lo tejidos contienen venas en su subsuperficie. Estas venas engañan a la red porque presentan un cambio de color bien definido pero no aportan relieve adicional al tejido. Al emplear imágenes de alta resolución, todas estas finas venas son apreciables, la red puede fijarse en ellas durante el entrenamiento y entender que debe darles una profundidad adicional. En cambio, si se usa baja resolución, las imágenes pierden nivel de detalle y las venas pasan más desapercibidas.

Efecto del modo de entrenamiento

El parámetro más importante que se puede variar es la forma en la que la red no supervisada es entrenada. Monodepth2, siendo entrenada y evaluada en el *KITTI*, tiene mejores métricas en modo estéreo-monocular, después en modo estéreo y finalmente los peores resultados se dan en modo monocular [37]. Esto no ocurre con EndoDepth que, como se observa entre los modelos 8, 9 y 10 en la tabla 4.1, presenta errores menores cuando se entrena de manera estéreo, luego estéreo-monocular y el peor caso es el monocular. Esto puede deberse al hecho de que, cuando se entrena monocular o estéreo-monocular, la red utiliza la consistencia temporal entre tres imágenes adyacentes y necesita estimar el cambio de posición entre las tres vistas. En *KITTI*, como el entorno es mayoritariamente estático y los objetos tienen buena textura, la red de posición funciona realmente bien. Sin embargo, en escenas intracorpóreas ocurre todo lo contrario, lo que provoca que la red de posición tenga que enfrentarse a un problema mucho mayor. Al entrenar en modo estéreo este problema no existe, teniéndolo EndoDepth mucho más fácil para converger a una buena solución, por eso funciona mejor. La red de posición en EndoDepth funciona de manera imprecisa, y por eso al usarse en el modo estéreo-monocular afecta de manera negativa confundiendo a la red y dando peor resultado que en modo estéreo, pero no tan malo como en modo solo monocular.

Efecto del tuneo monocular

Un experimento relevante para el trabajo futuro con estos métodos es comprobar la precisión de un modelo de EndoDepth ya entrenado en *Hamlyn*, y posteriormente tuneado con las imágenes del vídeo de test del *Hamlyn* que no vio durante el primer

entreno y que serán usadas para evaluarlo. La motivación es hacer SLAM visual para reconstruir tridimensionalmente las secuencias médicas. El experimento aquí expuesto simula lo que ocurriría en el problema real donde se tendría toda la secuencia monocular del endoscopio disponible para usarse en el tuneo de una red ya entrenada. Se ha probado a tunear con el vídeo *órganos* los modelos 8, 9 y 10, que están preentrenados de manera estéreo, monocular y estéreo-monocular, respectivamente. Los resultados pueden encontrarse en la tabla 4.2. Puede verse como tunear de manera monocular sobre el vídeo que servirá de test hace que los errores métricos mejoren cuando se parte de un modelo preentrenado de manera monocular o estéreo-monocular, pero no cuando se parte de estéreo. Esto es debido al hecho de que en los dos primeros casos para el tuneo monocular se parte de un modelo que ya tenía una red de posición entrenada sobre el *Hamlyn* y, aunque el tuneo sea solo durante una época, la red es capaz de ajustarse. Este es otro caso en el que el alto sobreajuste de EndoDepth resulta beneficioso.

Comparativa con el estado del arte

Tras realizar todos los experimentos anteriores se ha llegado a la conclusión de que la mejor manera de entrenar EndoDepth con los vídeos del *Hamlyn* es de forma estéreo, usando una alta resolución lo más cercana posible a la resolución original de las imágenes, usando un relleno de tipo borde y preentrenar los *encoders* en *ImageNet* antes de entrenar en *Hamlyn*. Por eso se ha usado esta configuración para entrenar modelos similares pero que usan los otros cuatro vídeos de test *abdomen*, *corazón*, *exploración* y *víscera* (modelos 11, 12, 13 y 14, respectivamente). Para el caso del vídeo de test *órganos*, el modelo 4 que fue entrenado a menor resolución y sin preentreno en ImageNet funciona un poco mejor que el 8 que fue entrenado con alta resolución y con *ImageNet*. Por esta razón se han usado los modelos 4, 11, 12 y 13 para compararse con los mejores métodos de estimación de profundidad en secuencias médicas a día de hoy, DefSLAM y SD-DefSLAM, que también han sido evaluados sobre los mismo cuatro vídeos de test. Los resultados aparecen en la tabla 4.4. Los modelos de EndoDepth han sido evaluados usando un factor de escala global (EndoDepth global), aprovechando que han sido entrenados de manera estéreo, y con un factor de escala local (EndoDepth local) como usan DefSLAM y SD-DefSLAM. Puede comprobarse como el rendimiento de EndoDepth local supera con creces a DefSLAM en los cuatro vídeos de test y a SD-DefSLAM en todos menos uno. EndoDepth global funciona ligeramente peor que EndoDepth local ya que su factor de escala es único para todos los modelos y no se ha obtenido comparando con el *ground truth*. Aún así, mejora a DefSLAM en tres de las cuatro secuencias y a SD-DefSLAM en dos de ellas. Es importante remarcar que

solo se ha comparado en estas cuatro secuencias del *Hamlyn* porque tanto DefSLAM como SD-DefSLAM fallan en el resto. Una posible causa de que SD-DefSLAM tenga mejores resultados que EndoDepth en la secuencia *corazón* es porque SD-DefSLAM (y también DefSLAM) parten de una malla plana 2D que van ajustando para adaptarse al entorno observado en cada imagen, y resulta que la secuencia *corazón* es muy plana y estática. Probablemente por este motivo les resulta muy complicado capturar las formas con cambios de profundidad bruscos y no consigan funcionar en muchas de las secuencias del *Hamlyn*. EndoDepth les sobrepasa en robustez, puesto que funciona en todos los vídeos y además proporciona mapas de profundidad densos capturando también la forma de las herramientas cuando DefSLAM y SD-DefSLAM las enmascaran para no verlas. Otro aspecto a destacar es que los modelos de EndoDepth comparados con DefSLAM y SD-DefSLAM han sido entrenados de manera estéreo sin aprovechar la consistencia temporal entre imágenes adyacentes que ellos sí utilizan. Con la red de estimación de posición actual, EndoDepth funciona peor al entrenarse de manera estéreo-monocular o monocular que con solo estéreo. Aún así, con los tres tipos de entrenamiento se observa un rendimiento superior que DefSLAM y SD-DefSLAM en el vídeo *órganos* (no se ha probado a entrenar de manera estéreo-monocular y monocular con el resto de vídeos de test). Con trabajo futuro se espera que el rendimiento de estos entrenamientos con estimación de posición mejoren.

La estimación de profundidad en los cinco vídeos de test del *Hamlyn* con los modelos estéreo de EndoDepth, su representación 3D y las imágenes de profundidad *ground truth* a manos de Libelas pueden encontrarse en el siguiente link: https://youtu.be/V3Be2W3iomI.

	DefSLAM	SD-DefSLAM	EndoDepth global	EndoDepth local
Abdomen	23.98	22.2	45.29	11.19
Corazón	3.84	1.17	3.48	2.42
Exploración	17.02	12.56	11.59	10.9
Órganos	13.02	6.63	6.08	4.85

Tabla 4.4: Comparación de errores por píxel en DefSLAM, SD-DefSLAM y EndoDepth sobre cuatro secuencias de *Hamlyn*, usando como métrica la raíz del error cuadrático medio (RMSE) en mm. DefSLAM, SD-DefSLAM y EndoDepth local utilizan un factor de escala individual para cada imagen, mientras que EndoDepth global usa un solo factor de escala para todas las imágenes. Los modelos de EndoDepth han sido todos entrenados de manera estéreo (modelos 11, 12, 13 y 4 de la tabla 4.1, respectivamente).

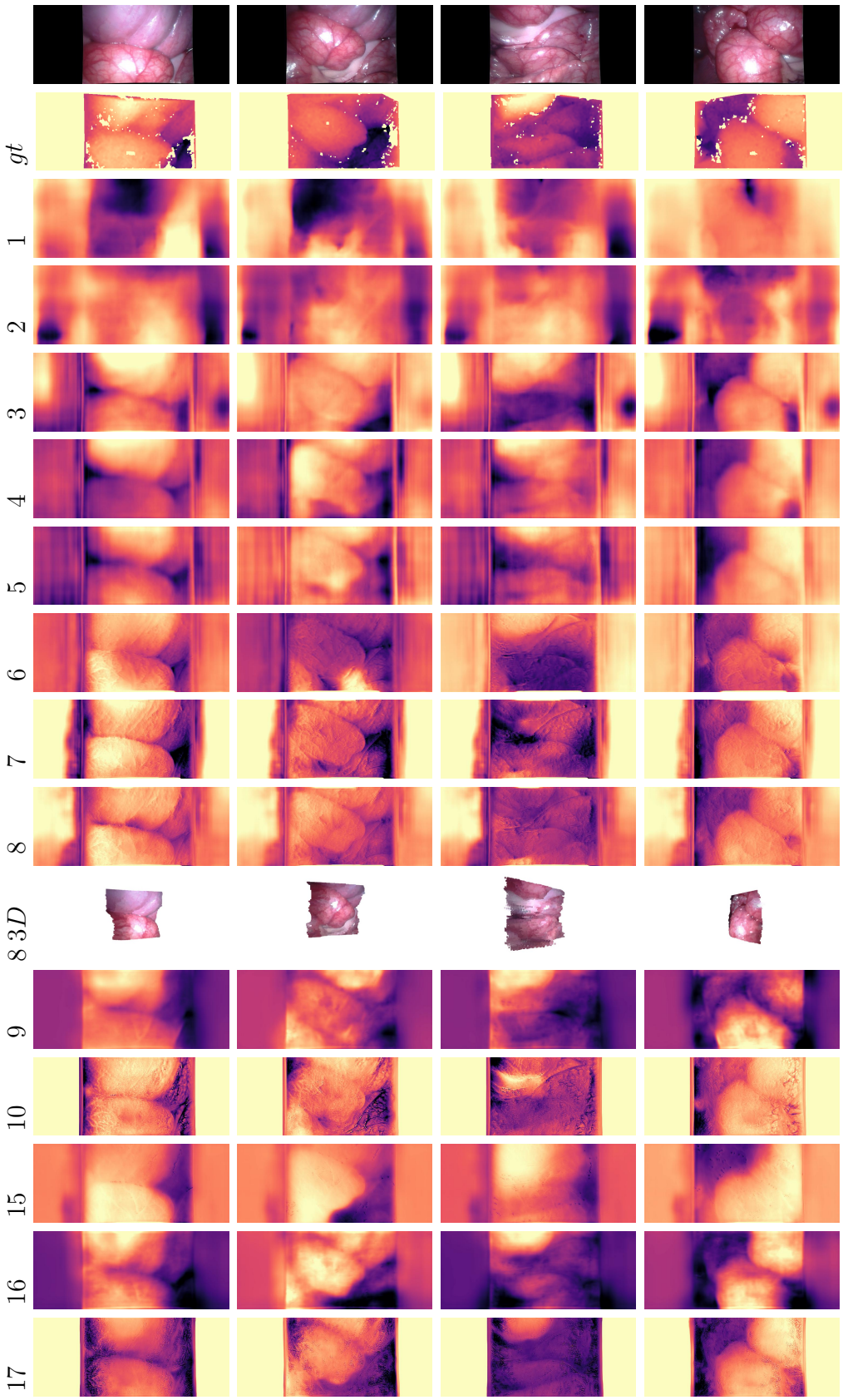

Figura 4.2: Resultados cualitativos de profundidad de los 17 modelos entrenados, con la misma numeración que en las tablas 4.1 y 4.2. Un tono más oscuro significa profundidad mayor. Las imágenes de la primera fila son las imágenes de entrada, *gt* es la profundidad *pseudo-ground truth* de *Libelas* y *8 3D* es la representación tridimensional usando la profundidad predicha por 8.

Capítulo 5

Conclusiones y trabajo futuro

En este trabajo presentamos EndoDepth, una red neuronal profunda no supervisada para estimación robusta de profundidad monocular en vídeos de endoscopias. EndoDepth consigue mejorar el estado del arte al reportar unos errores medios (*RMSE*) notablemente bajos, de hasta 2mm en una de las secuencias, y lo consigue mientras produce mapas densos de profundidad capturando todos los objetos de la escena, incluso las herramientas. Además, es importante remarcar que los modelos desarrollados operan en situación de desventaja: la profundidad de Endodepth se estima a partir de una imagen, sin explotar la consistencia temporal entre las imágenes consecutivas de un vídeo. Se espera que con el trabajo futuro se pueda aprovechar la línea temporal de las imágenes, haciendo que los errores obtenidos en este trabajo se reduzcan todavía más.

En cuanto a trabajo futuro, a corto plazo hemos planificado mejorar la técnica de estimación de profundidad en vídeos de endoscopias con el objetivo de elevar a EndoDepth a un escalón superior. Continuaremos los experimentos con preentrenamiento en *ImageNet* y se desarrollará una versión optimizada para imágenes médicas de la función de suavizado. Con la ayuda de una estación de trabajo mucho más potente se podrán entrenar muchos más modelos en menos tiempo, haciendo accesible realizar una validación cruzada sobre todos y cada unos de los vídeos de *Hamlyn*, para así evitar posibles errores de inicialización y que las métricas obtenidas sean lo más fiables posibles. El siguiente objetivo, más ambicioso, consistirá en adaptar EndoDepth para que sea capaz de realizar SLAM visual [50], o lo que es lo mismo, ser capaz de reconstruir tridimensionalmente y autolocalizar el endoscopìo en el interior del cuerpo haciendo uso únicamente de su grabación monocular. Para ello será necesario mejorar la estimación de posición, porque la original de Monodepth2, como hemos comprobado en este trabajo, tiene resultados adversos en nuestras imágenes de endoscopias.

Capítulo 6

Bibliografía

[1] Las cifras del cáncer en españa 2020. *Sociedad Española de Oncología Médica (SEOM)*, 2020.

[2] Mercè Comas, Joan Mendivil, Montserrat Andreu, Cristina Hernández, and Xavier Castellsl. Long-term prediction of the demand of colonoscopies generated by a population-based colorectal cancer screening program. *PLOS ONE*, 11(10):1–13, 10 2020.

[3] J. Lee, S. W. Park, Y. S. Kim, K. J. Lee, H. Sung, P. H. Song, W. J. Yoon, and J. S. Moon. Risk factors of missed colorectal lesions after colonoscopy. *Medicine, 96:27, e7468*, 2017.

[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[5] Carlos Campos, Richard Elvira, Juan J. Gomez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *arXiv preprint arXiv:2007.11898*, 2020.

[6] Davison. Real-time simultaneous localisation and mapping with a single camera. pages 1403–1410 vol.2, 2003.

[7] Jose Lamarca, Shaifali Parashar, Adrien Bartoli, and JMM Montiel. Defslam: Tracking and mapping of deforming scenes from monocular sequences. *IEEE Transactions on Robotics*, 2020.

[8] Juan J. Gómez Rodríguez, José Lamarca, Javier Morlana, Juan D. Tardós, and José M.M. Montiel. Sd-defslam: Semi-direct monocular slam for deformable and intracorporeal scenes. *arXiv:2010.09409v1 [cs.CV]*, 2020.

[9] Endomapper: Real-time mapping from endoscopic video. *CORDIS*, 2019.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[12] Jose M. Facil, Benjamin Ummenhofer, Huizhong Zhou, Luis Montesano, Thomas Brox, and Javier Civera. CAM-Convs: Camera-Aware Multi-Scale Convolutions for Single-View Depth. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[13] Frederik Warburg, Soren Hauberg, Manuel Lopez-Antequera, Pau Gargallo, Yubin Kuang, and Javier Civera. Mapillary street-level sequences: A dataset for lifelong place recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[14] Berta Bescós, José M. Fácil, Javier Civera, and José Neira. Dynslam: Tracking, mapping and inpainting in dynamic scenes. *CoRR*, abs/1806.05620, 2018.

[15] Nhs endoscopy. https://www.nhs.uk/conditions/endoscopy/. Accedido: 04-11-2020.

[16] JMM Montiel. Endomapper: Real-time mapping from endoscopic video. https://sites.google.com/unizar.es/endomapper/home. Accedido: 04-11-2020.

[17] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[18] Kenji Hata and Silvio Savarese. Lecture notes in cs231a: Computer vision, from 3d reconstruction to recognition, Winter 2018.

[19] Oculus VR LLC. *Oculus Developer Guide*.

[20] Bastian Leibe. Lecture notes in machine learning, Winter 2019.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning:An Introduction*. The MIT Press. A Bradford Book, 2014,2015.

[22] D. Michie, D.J. Spiegelhalter, and C.C. Taylor (eds). *Machine Learning, Neural and Statistical Classification*. 1994.

[23] Electronic 'brain' teaches itself. *The New York Times*, 1958.

[24] Alejandro Cartas. Diagrama de un perceptrón con cinco señales de entrada. https://commons.wikimedia.org/wiki/File:Perceptr%C3%B3n_5_unidades.svg. Creado: 12 de julio de 2015.

[25] Stanford Vision Lab. Imagenet large scale visual recognition challenge (ilsvrc). http://www.image-net.org/challenges/LSVRC/. Accedido: 08-11-2020.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Marc'Aurelio Ranzato. Lecture notes in an introduction to deep learning. deeplearn summer school - bilbao, July 2017.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in neural information processing systems*, pages 550–558, 2016.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[32] Tawsifur Rahman, Muhammad E. H. Chowdhury, Amith Khandakar, Khandaker R. Islam, Khandaker F. Islam, Zaid B. Mahbub, Muhammad A. Kadir, and Saad Kashem. Transfer learning with deep convolutional neural network (cnn) for pneumonia detection using chest x-ray. *Applied Sciences*, 10(9):3233, May 2020.

[33] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

[34] DongHo Hong, Wallapak Tavanapong, Johnny Wong, JungHwan Oh, and Piet C. de Groen. 3d reconstruction of virtual colon structures from colonoscopy images. *Computerized Medical Imaging and Graphics*, 38(1):22 – 33, 2014.

[35] Anita Rau, PJ Eddie Edwards, Omer F Ahmad, Paul Riordan, Mirek Janatka, Laurence B Lovat, and Danail Stoyanov. Implicit domain adaptation with conditional generative adversarial networks for depth prediction in endoscopy. *International journal of computer assisted radiology and surgery*, 14(7):1167–1176, 2019.

[36] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[37] Clement Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[38] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[39] S. Giannarou, M. Visentini-Scarzanella, and G. Yang. Probabilistic tracking of affine-invariant anisotropic regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):130–143, 2013.

[40] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[41] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Asian Conference on Computer Vision (ACCV)*, 2010.

[42] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[43] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2022–2030, 2018.

[44] PyTorch. Torchvision models. https://pytorch.org/docs/stable/torchvision/models.html.

[45] Benedikt Pfülb, Alexander Gepperth, S Abdullah, and A Kilian. Catastrophic forgetting: still a problem for dnns. In *International Conference on Artificial Neural Networks*, pages 487–497. Springer, 2018.

[46] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[47] Yevhen Kuznietsov, Jorg Stuckler, and Bastian Leibe. Semi-supervised deep learning for monocular depth map prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6647–6655, 2017.

[48] Xiaoyang Guo, Hongsheng Li, Shuai Yi, Jimmy Ren, and Xiaogang Wang. Learning monocular depth by distilling cross-domain stereo networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 484–500, 2018.

[49] Sudeep Pillai, Rareş Ambruş, and Adrien Gaidon. Superdepth: Self-supervised, super-resolved monocular depth estimation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9250–9256. IEEE, 2019.

[50] Lokender Tiwari, Pan Ji, Quoc-Huy Tran, Bingbing Zhuang, Saket Anand, and Manmohan Chandraker. Pseudo rgb-d for self-improving monocular slam and depth prediction. *arXiv preprint arXiv:2004.10681*, 2020.

[51] Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969.

Lista de Figuras

Lista de Tablas