UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería Informática

Estimación de la profundidad de una escena mediante cámaras RGB-D y modelos de estimación de profundidad monocular

Scene depth estimation using RGB-D cameras and monocular depth estimation models

Realizado por

Rubén Jiménez Reina

Tutorizado por

Javier González Jiménez

José Raúl Ruiz Sarmiento

Departamento

Sistemas y Automática

MÁLAGA, SEPTIEMBRE DE 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

Estimación de la profundidad de una escena mediante cámaras RGB-D y modelos de estimación de profundidad monocular

Scene depth estimation using RGB-D cameras and monocular depth estimation models

Realizado por
Rubén Jiménez Reina

Tutorizado por
Javier González Jiménez
José Raúl Ruiz Sarmiento

Departamento
Sistemas y Automática

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE de 2025

Fecha defensa: Septiembre de 2025

Resumen

La percepción de la profundidad es un requisito fundamental para una interacción segura y eficaz con el entorno en campos como la robótica y la conducción autónoma. Los enfoques existentes para la adquisición de profundidad, como LiDAR, visión estereoscópica, sensores RGB-D y estimación de profundidad monocular, ofrecen ventajas complementarias, pero se ven invariablemente afectados por la incertidumbre de la medición. Este trabajo aborda el problema de estimar y refinar tanto la profundidad como su incertidumbre asociada, combinando los resultados de un sensor RGB-D con un modelo de estimación de profundidad monocular. Proponemos modelar la incertidumbre como una distribución gaussiana por píxel, lo que permite la fusión de múltiples fuentes de profundidad mediante la multiplicación probabilística. Para lograrlo, desarrollamos un modelo de aprendizaje automático capaz de predecir mapas de incertidumbre a partir de imágenes RGB emparejadas y estimaciones de profundidad, capturando tanto las fuentes de error aleatorias como las sistemáticas. Los experimentos se llevan a cabo utilizando el conjunto de datos sintéticos HyperSim, lo que garantiza el acceso a la profundidad real para el entrenamiento supervisado. El método propuesto produce mapas de profundidad refinados con menor incertidumbre, lo que ofrece una representación más fiable de la escena. Entre las posibles aplicaciones se incluyen la robótica, la navegación autónoma y otros ámbitos que requieren una percepción de profundidad robusta.

Palabras clave: Estimación de profundidad, Estimación de incertidumbre, Visión por computador

Abstract

Depth perception is a fundamental requirement for safe and effective interaction with the environment in fields such as robotics and autonomous driving. Existing approaches to depth acquisition—such as LiDAR, stereo vision, RGB-D sensors, and monocular depth estimation—offer complementary advantages but are invariably affected by measurement uncertainty. This work addresses the problem of estimating and refining both depth and its associated uncertainty by combining the outputs of an RGB-D sensor with a monocular depth estimation model. We propose modeling uncertainty as a per-pixel Gaussian distribution, enabling the fusion of multiple depth sources through probabilistic multiplication. To achieve this, we develop a machine learning model capable of predicting uncertainty maps from paired RGB images and depth estimates, capturing both aleatoric and systematic sources of error. Experiments are conducted using the HyperSim synthetic dataset, ensuring access to ground-truth depth for supervised training. The proposed method produces refined depth maps with lower uncertainty, offering a more reliable scene representation. Potential applications include robotics, autonomous navigation, and other domains requiring robust depth perception.

Keywords: Depth estimation, Uncertainty estimation, Computer vision

Índice

1

Introducción

1.1. Motivación

La habilidad para percibir la profundidad de una escena resulta esencial para el entendimiento y la capacidad de interactuar con la misma. Esto es especialmente cierto en ámbitos como el de la robótica, para garantizar que sistemas autónomos operen con seguridad en escenarios tan variados como puedan ser la robótica doméstica o la conducción autónoma.

Existen diversas metodologías para medir o aproximar la profundidad en una escena. Entre las más destacadas se encuentran los sensores LIDAR, sistemas de ultrasonidos, los sistemas de visión estéreo (Saxena et al., 2007), cámaras RGB-D, y los modelos – esto es, redes neuronales – de aprendizaje automático de estimación de profundidad monocular.

Cada método tiene sus ventajas e inconvenientes. Por ejemplo, los sensores LIDAR suelen ofrecen mediciones de distancia muy precisas, pero son inherentemente dispersas, lo que significa que solo proporcionan puntos de profundidad en ubicaciones discretas, dejando áreas de la escena sin información directa, además de ser, por lo general, muy costosos. Los sistemas de visión estéreo necesitan dos cámaras separadas a una distancia conocida y una calibración muy precisa, esto se debe a que estos sistemas calculan la profundidad mediante el principio de triangulación: al identificar puntos correspondientes en las imágenes de ambas cámaras y conocer la geometría relativa de las cámaras, es posible determinar la posición tridimensional de ese punto en el espacio. Las cámaras RGB-D proporcionan medidas densas, pero suelen tener un alcance máximo limitado, a partir del cual la calidad de la medida disminuye significativamente (Rustler et al., 2025). Los modelos de estimación de profundidad monocular producen resultados densos y verosímiles, pero no tienen base física, y pueden ser resultado de alucinaciones del modelo.

Sin embargo, todos estos métodos se ven afectados por un inconveniente común: la incertidumbre de la medición. Esta incertidumbre se refiere a la discrepancia esperada entre el valor de profundidad real y el observado. Si bien los fabricantes de sensores suelen proporcionar una especificación de la incertidumbre de sus productos, esta se refiere a la incertidumbre en condiciones ideales. Por ejemplo, un sensor podría especificar una precisión de ± 1 milímetro a una distancia de 1 metro en una superficie uniforme y bien iluminada. No obstante, el mundo real introduce desafíos que dificultan la obtención de medidas precisas, tales como las condiciones de iluminación y los materiales presentes en la escena. Por ejemplo, el haz de luz infrarroja emitido por una cámara RGB-D podría ser absorbido o reflejado en función del material, ángulo y porosidad de la superficie en el que impacte. Estos fenómenos se pueden traducir en medidas inconsistentes, erróneas, e incluso en la ausencia de datos.

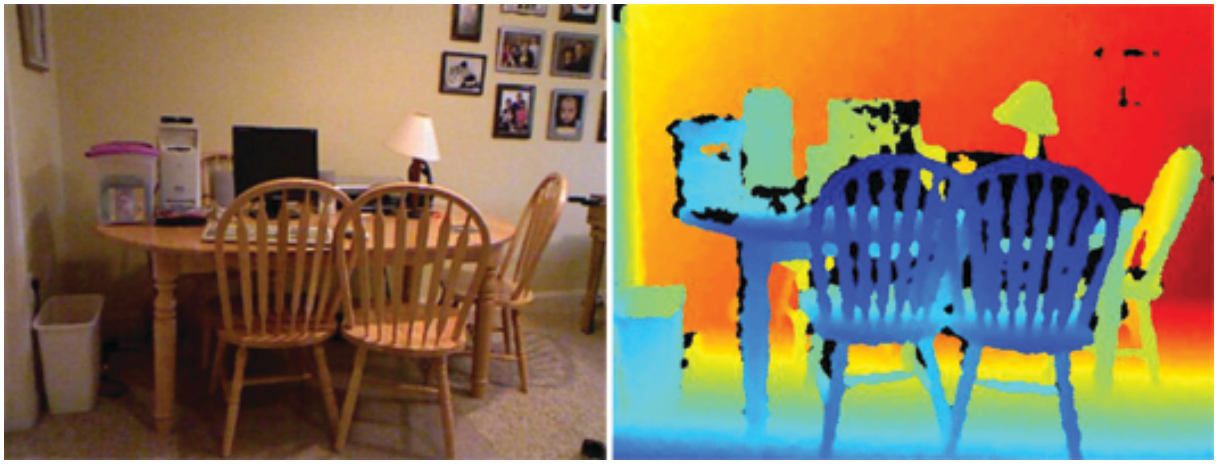

Figura 1: Muestra del conjunto de datos NYU Depth Dataset v2
(Nathan Silberman & Fergus, 2012)

La capacidad de conocer o poder aproximar esta incertidumbre nos podría permitir cuantificar la calidad y robustez de las medidas obtenidas, lo que podría mejorar nuestro entendimiento de la escena. Además, esto podría permitir combinar las medidas de varios métodos de captura o estimación de profundidad de forma inteligente, dando más relevancia a las mediciones más confiables.

1.2. Objetivo

En este trabajo se pretende desarrollar una técnica para la estimación de un mapa – esto es, una matriz – de profundidad y su correspondiente mapa de incertidumbre asociados a una escena a partir de la información proveniente de una cámara RGB-D y un modelo de estimación de profundidad monocular.

1.2.1. Estimación de la incertidumbre

Para esto, comenzaremos asumiendo que la incertidumbre de un mapa de profundidad asociado a una escena se puede modelar como una distribución gaussiana, de forma que, disponiendo de un mapa de profundidad, podemos interpretarlo como una matriz de distribuciones normales $N(\mu_{ij}, \sigma_{ij}^2)$, donde μ_{ij} y σ_{ij}^2 son el valor de la profundidad y la incertidumbre del mismo en el punto (i, j), respectivamente. Llamaremos a esta matriz **mapa de distribuciones**.

Dado que el producto de dos funciones de densidad de probabilidad gaussianas es otra función de densidad de probabilidad gaussiana cuya varianza es inferior o igual a la de las distribuciones originales (Smith, 2025), si disponemos de dos distribuciones $N_1(\mu_1, \sigma_1^2)$ y $N_2(\mu_2, \sigma_2^2)$, podemos calcular su producto como:

$$N(\mu_3, \sigma_3^2) = N(\mu_1, \sigma_1^2)N(\mu_2, \sigma_2^2) = N\left(\frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}, \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\right) \quad (1)$$

Obsérvese que la contribución de cada media está ponderada por la varianza asociada a la otra, de modo que, a mayor la varianza asociada a una media, mayor será la contribución de la otra. Si multiplicamos elemento a elemento dos mapas de distribuciones asociados a los mapas de profundidad de una escena determinada obtendremos otro mapa de distribuciones al que llamaremos **mapa refinado**, donde μ_{ij} y σ_{ij}^2 son la **medida refinada** y la **incertidumbre refinada** en el punto (i, j).

Estimar la incertidumbre asociada a un mapa de medidas de profundidad producida por un sensor con base física tal como pueda ser una cámara RGB-D podría lograrse utilizando un modelo de error cuadrático:

$$\sigma^2 = \alpha_0 + \alpha_1 x + \alpha_2 x^2 \quad (2)$$

Este método tiene varias limitaciones, siendo la principal la ausencia de contexto: para calcular la incertidumbre en un punto tan solo se tiene en cuenta el mismo, y no se consideran los puntos vecinos, ni tampoco otros aspectos de la escena como la iluminación, texturas, etc. Además, calcular la incertidumbre resulta más complicado con métodos de obtención de medidas que no tienen base física, tales como los modelos de aprendizaje automático de estimación de profundidad. No obstante, supongamos la existencia de una función $F(x, d)$ tal que, dados una imagen x y un mapa de profundidad d asociados a una escena, estime un **mapa de incertidumbres** donde cada elemento σ_{ij}^2 es la incertidumbre asociada a la profundidad en el punto d_{ij}.

Una vez disponemos de una forma de estimar estos mapas de distribuciones, podríamos tomar varios mapas de profundidad d_k para misma imagen x de una escena y estimar los mapas de profundidad e incertidumbre refinados como:

$$N(\mu, \sigma^2) = \prod_{i}^{k} N(d_i, F(x, d_i)) \tag{3}$$

En este trabajo tomaremos el canal de profundidad de una cámara RGB-D como el primer mapa de profundidad, y a partir de los canales RGB emplearemos un modelo de estimación de profundidad monocular tal como *Depth Anything* (Yang et al., 2024) o *Depth Pro* (Bochkovskii et al., 2025) para estimar un segundo mapa de profundidad.

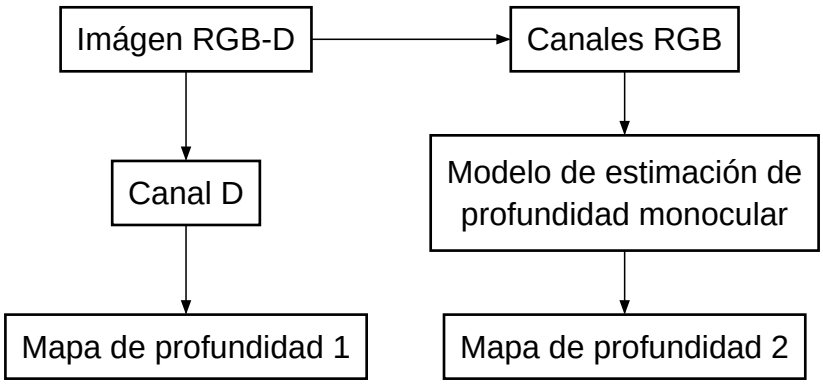

Figura 2: Método para la obtención de dos mapas de profundidad a partir de una imagen RGB-D

1.2.2. Enfoque propuesto

En este trabajo desarrollaremos un **modelo de aprendizaje automático de estimación de incertidumbre** que actue como la función $F(x, d)$ mencionada anteriormente. Este modelo podría aprender relaciones complejas entre los elementos de la imagen y el mapa de profundidad asociado a esta, teniendo en cuenta detalles como estructuras locales, texturas, etc. Además, esto nos podría permitir modelar tanto la incertidumbre aleatórica como la sistémica (Kiureghian & Ditlevsen, 2009), esto es, la incertidumbre asociada a errores aleatorios introducidos por la cámara RGB-D y la incertidumbre asociada a errores sistémicos que podría introducir el modelo de estimación de profundidad monocular.

De este modo, finalmente, dispondremos de los mapas de incertidumbre asociados a cada mapa de profundidad, un mapa de profundidad refinado y un mapa de incertidumbre refinado asociado a este último. Esto proporcionaría información sobre la profundidad de la escena más allá de los propios valores de profundidad.

1.3. Trabajos relacionados

De forma previa al desarrollo de este trabajo, se estudiaron métodos para la estimación de la profundidad de una escena mediante redes neuronales, y posteriormente métodos para la estimación de la incertidumbre de medidas y estimaciones de profundidad.

En el ámbito de la estimación de profundidad, destacan los trabajo de Yang et al. (2024), quienes proponen *Depth Anything*, y de Bochkovskii et al. (2025), quienes proponen *Depth Pro*. Estos modelos de estimación de profundidad monocular producen resultados de gran calidad en entornos muy variados. Por ello, ambos se consideraron para el desarrollo de este trabajo como modelo de estimación base.

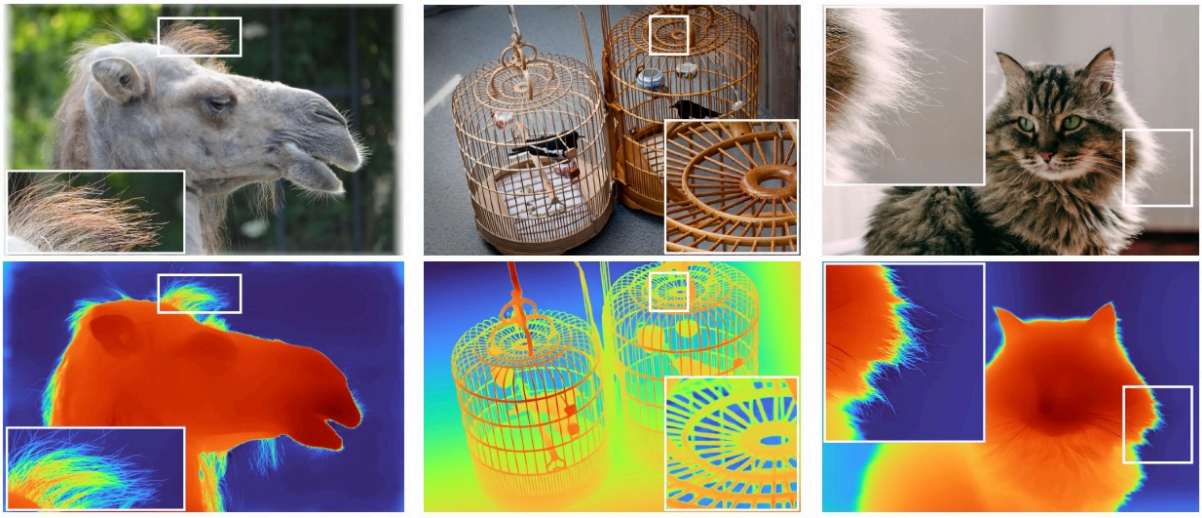

Figura 3: Imagen de presentación del modelo *Depth Pro* (Bochkovskii et al., 2025)

Por otro lado, en el ámbito de la estimación de incertidumbre más notable es Baumann et al. (2023), donde se estima tanto la profundidad como el error de la misma en una escena mediante una familia de redes neuronales llamadas «MIMO». A diferencia del trabajo de Baumann et al. (2023), en este trabajo no se pretende estimar la profundidad de una escena mediante una red neuronal, sino el desarrollo de una red neuronal para estimar la incertidumbre de un par imagen-profundidad, para más adelante combinar la información procedente de una cámara RGB-D y un modelo de estimación de profundidad. No obstante, en este trabajo se implementan algunas de las ideas de Baumann et al. (2023) referentes a la estimación de incertidumbre, que serán mencionadas más adelante en el capítulo referente al desarrollo del modelo de estimación de incertidumbre.

1.4. Tecnologías utilizadas

Este trabajo ha sido elaborado con el lenguaje de programación *Python* debido a su presencia en la industria del aprendizaje automático y la gran cantidad de librerías disponibles, junto al gestor de proyectos y dependencias *UV*. Para la elaboración y entrenamiento del modelo se ha optado por la librería *Pytorch* (Ansel et al., 2024) y *Lightning* (Falcon & The PyTorch Lightning team, 2019). Estas librerías proporcionan una amplia gama de herramientas y funcionalidades para el desarrollo y entrenamiento de modelos de aprendizaje automático, y son muy populares en la industria de la visión artificial. Además, el modelo de estimación de profundidad monocular escogido para el desarrollo de este trabajo, *Depth Pro*, fue desarrollado usando *Pytorch*, por lo que la elección de estas librerías fue una decisión natural, al ofrecer un altísimo grado de compatibilidad.

También se ha empleado la librería *Optuna* (Akiba et al., 2019) para la búsqueda inicial de hiperparámetros de forma previa al entrenamiento del modelo. El proceso de búsqueda de hiperparámetros se explicará en la Sección 3.4.

Los entrenamientos se llevaron a cabo en la plataforma CSAR del equipo de investigación MAPIR de la Universidad de Málaga, usando la tecnología de contenedores LXC. Dada la potencia del hardware disponible en la plataforma CSAR, fue posible iterar rápidamente sobre diferentes configuraciones de hiperparámetros y modelos.

Para monitorizar y evaluar los entrenamientos se optó por la plataforma *Neptune* y la herramienta *Rerun* (Rerun Development Team, s. f.). Mientras que *Neptune* nos permite realizar un seguimiento de los experimentos en línea y en tiempo real, *Rerun* nos permite visualizar y explorar los datos de los experimentos de forma interactiva y fácil de usar, lo que resultó muy útil en las fases tempranas del desarrollo del modelo.

2
Datos

2.1. Conjunto de datos

Dado que pretendemos entrenar un modelo que estime la varianza entre la profundidad real de una escena y la estimada por otro modelo, será necesario disponer de un conjunto de datos que contenga imágenes y sus profundidades asociadas. Además, estas profundidades deben ser perfectamente conocidas, sin ruido ni sesgos que puedan afectar la calidad del modelo entrenado. Los conjuntos de datos capturados en el mundo real tales como NYU Depth Dataset v2 (Nathan Silberman & Fergus, 2012) contienen imágenes reales y profundidades capturadas por sensores de profundidad. Pero estas profundidades están afectadas por la incertidumbre intrínseca del sensor con el que se capturaron y por los errores de medición comentados anteriormente (vease Figura 1), de modo que pueden discrepar sustancialmente de la profundidad real, y por lo tanto no suponen una buena base para el entrenamiento.

Por esta razón, se ha decidido usar un conjunto de datos sintéticos, es decir, un conjunto de datos donde las muestras no han sido capturadas del mundo real, sino generadas de forma artificial, con técnicas tales como renderizados 3D. La mayor ventaja de usar un conjunto de datos sintético es que es posible conocer la profundidad real de una escena, y esta no estaría afectada por sesgos ni ruido. Por otro lado, es necesario que las imágenes sean fotorrealistas, esto es, lo más verosímiles posible, para que los modelos involucrados en el proceso dispongan de muestras que se asimilen a las del mundo real. El conjunto de datos seleccionado para llevar a cabo el entrenamiento es HyperSim de Apple (Susskind, 2021), ya que proporciona una gran cantidad de datos consistiendo en fotogramas de videos en una variedad de entornos fotorrealistas donde la profundidad de la escena representada en cada fotograma es perfectamente conocida.

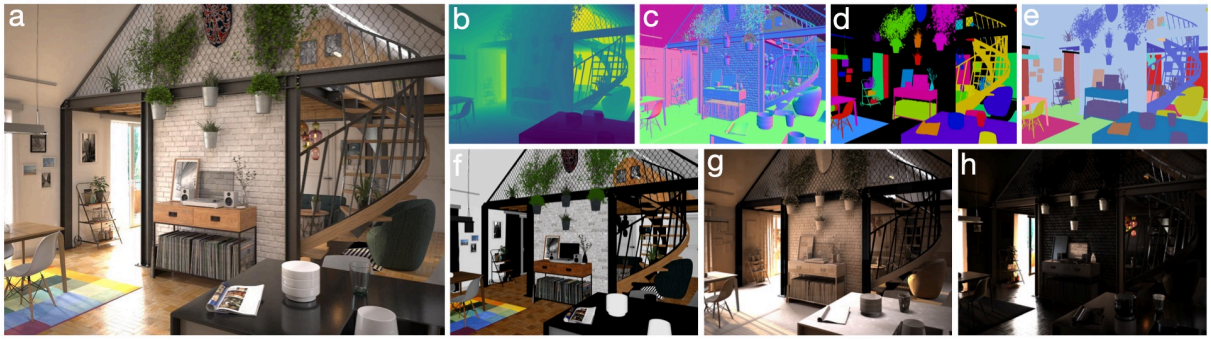

Figura 4: Muestra del conjunto de datos HyperSim (Susskind, 2021)

2.2. Preprocesado

HyperSim contiene multitud de datos asociados a cada fotograma, incluyendo mapas de normales, máscaras semánticas, y otros datos que no resultan necesarios para el entrenamiento, ya que en el mundo real los modelos no disponen de esta información. Estos datos son eliminados con un script (ver Apéndice A) para reducir el uso de espacio en disco del conjunto de datos. De este modo, conservamos las imágenes y los mapas de profundidad originales.

Aunque el conjunto de datos ofrece 77.400 imagenes, se ha decidido utilizar aproximadamente un 20% del mismo para reducir la duración del entrenamiento y el uso de recursos computacionales. Estos datos fueron posteriormente limpiados, eliminando entradas con valores anómalos, con lo que finalmente quedaron alrededor de 13.000 entradas; esto es aproximadamente el 17% del conjunto original y un 75% del subconjunto escogido. De estos datos, 8.026 serán usados durante el entrenamiento, con una división de 80% para entrenamiento y 20% para validación; El resto (4337) será usado como datos de prueba. Se ha puesto especial cuidado en que los datos de prueba sean de escenas no presentes en el conjunto de entrenamiento y validación, con el objetivo de evaluar el rendimiento del modelo en situaciones no vistas anteriormente.

Una vez limpiados los datos, y con el objetivo de acelerar el proceso de entrenamiento, se han precalculado todas las estimaciones de profundidad con DepthPro. Esto nos permite acceder rápidamente a las mismas – a las cuales nos referiremos de aqui en adelante como «mapas de profundidad estimados» – durante el entrenamiento sin tener que calcularlas nuevamente. Además, también calculamos los bordes – con el operador Sobel – y el Laplaciano de las imagenes, los mapas de profundidad originales, y los mapas de profundidad estimados. La motivación es que

estas representaciones resaltan las transiciones bruscas y las estructuras locales, lo que puede aportar información adicional al modelo de estimación de incertidumbre que pretendemos desarrollar. En un escenario real únicamente dispondríamos de la imagen y de la estimación de profundidad, de modo que los bordes y Laplacianos calculados a partir de ellos serían las únicas características adicionales accesibles. Sin embargo, en nuestro conjunto de datos sintético sí contamos con la profundidad real, lo que nos permite calcular sus bordes y Laplaciano de la misma. Aunque esta información no estaría disponible en el mundo real, la aprovecharemos más adelante durante el entrenamiento como un recurso auxiliar para reforzar el aprendizaje del modelo. Finalmente, los valores de las imagenes y los bordes se escalan al rango [0, 1] para aportar estabilidad al entrenamiento, mientras que los mapas de profundidad originales y estimados mantienen su escala original. Esto se hace para mantener la información de profundidad sin perder detalles importantes.

Este preprocesamiento se realiza una única vez de forma previa al entrenamiento, almacenando cada entrada en un archivo hdf5 (The HDF Group, 2025) para facilitar su posterior acceso durante el entrenamiento. El preprocesado se realiza en paralelo para cada entrada del conjunto de datos original, ya que cada entrada puede ser procesada de forma independiente sin afectar a las demás. Esto nos permite reducir considerablemente el tiempo de preprocesamiento.

2.3. Aumento de datos

Para compensar la reducción de datos resultante de usar un subconjunto del conjunto de datos original y la posterior limpieza del mismo, se aplicaron técnicas de aumento de datos para generar muestras adicionales a partir de las existentes a medida que se entrena el modelo. El aumento de datos consiste en aplicar transformaciones tales como rotaciones, distorsiones, alteraciones en la luminosidad y color de la imagen, entre otras. Estas transformaciones pueden ser aplicadas de forma consistente (es decir, siempre) o aleatoria con una cierta probabilidad, de modo que a partir de los datos originales se generan nuevas muestras que amplían la diversidad de los datos de entrenamiento. Esto ayuda al modelo a aprender mejor y a generalizar mejor a nuevos datos (Khoshgoftaar, 2019).

Un aspecto importante es que no todas las transformaciones aplicadas a la imagen original preservan la validez del mapa de profundidad previamente estimado. Decimos que un mapa de profundidad es válido o consistente respecto a una imagen

transformada cuando sigue representando la misma profundidad que el modelo produciría si se recalculase a partir de esa nueva imagen. En otras palabras, el mapa sigue siendo una descripción correcta de la escena bajo la transformación aplicada. Este concepto de consistencia está directamente relacionado con la conmutatividad entre la transformación y el estimador de profundidad. Si una transformación T conmuta con el estimador E, entonces el orden en que se aplican no altera el resultado:

$$E(T(I)) = T(E(I)) \tag{4}$$

donde I es la imagen original, E el estimador de profundidad (en nuestro caso, DepthPro) y T una transformación o conjunto de transformaciones. En estos casos –como ocurre con las transformaciones geométricas tales como rotaciones, traslaciones, y volteos– basta con aplicar la misma transformación al mapa precalculado para que continúe siendo válido, ya que mantiene la coherencia con lo que el estimador produciría. Por el contrario, cuando la transformación y el estimador no conmutan, la igualdad se rompe:

$$E(T(I)) \neq T(E(I)) \tag{5}$$

Esto ocurre con las transformaciones que modifican las características visuales de la imagen, como cambios en la iluminación, el contraste, el color, la introducción de ruido o distorsiones ópticas. Estas alteraciones afectan directamente a las señales que el modelo utiliza para inferir la profundidad, de modo que el mapa precalculado deja de ser consistente con la imagen transformada. En resumen, la validez de un mapa de profundidad tras aplicar una transformación depende precisamente de si la transformación conserva o rompe esta conmutatividad con el estimador de profundidad. Por este motivo, se decidió usar transformaciones que no pudiesen afectar al mapa de profundidad estimado, con el objetivo de poder reutilizar las estimaciones precalculadas durante el preprocesamiento.

El procedimiento implementado para transformar las imágenes es el siguiente: se reducen la imagen, el mapa de profundidad original, y el mapa de profundidad estimado de su tamaño original a 256x256. A continuación, se voltean horizontalmente con una probabilidad del 50%. Finalmente, se escoge un ángulo uniformemente distribuido entre -8 y 8 grados y se aplica una rotación a la imagen y a los mapas de profundidad.

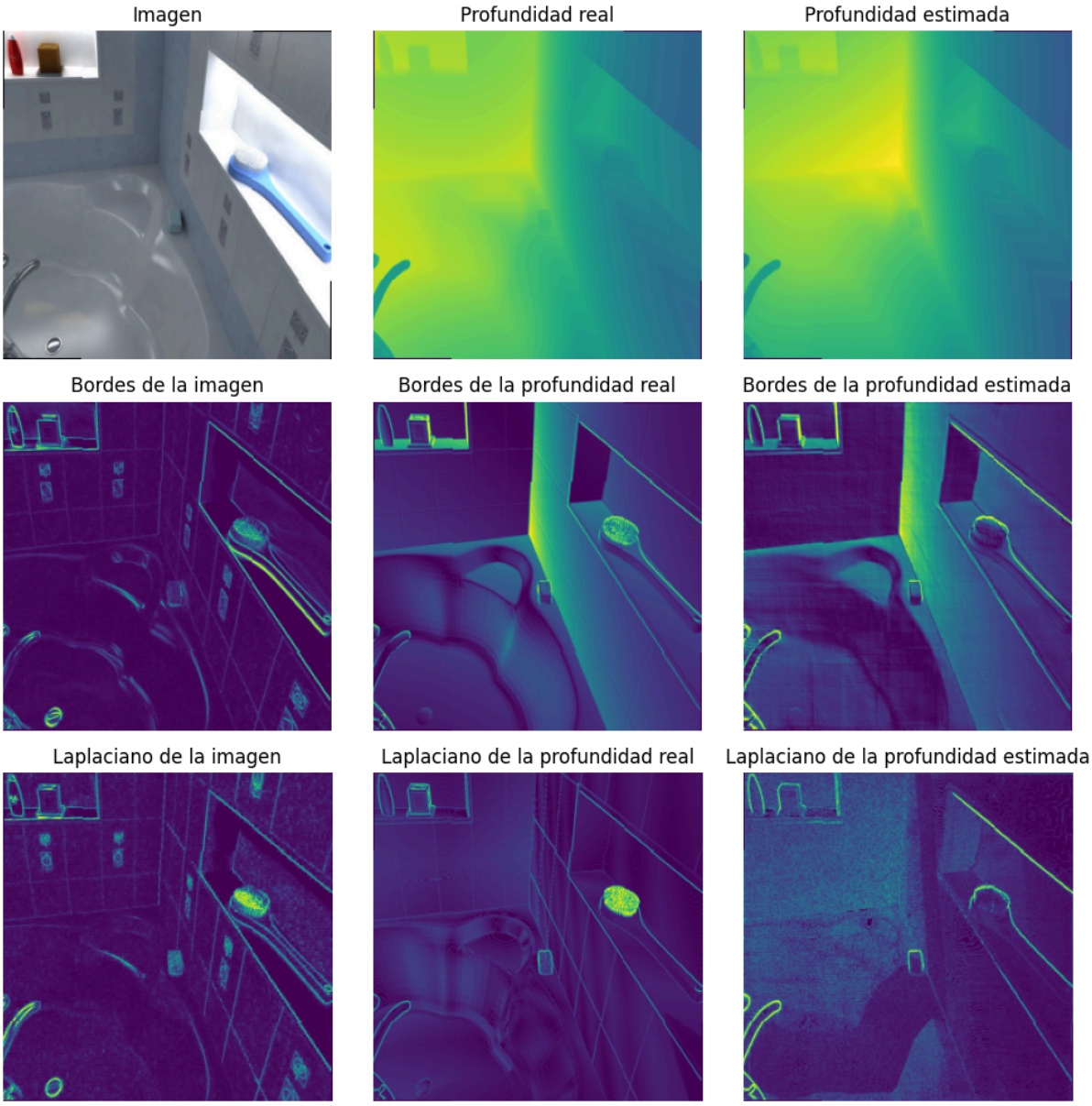

Figura 5: Muestra de los datos empleados en el entrenamiento con transformaciones aplicadas

3
Modelo

3.1. Arquitectura Base

La tarea principal del modelo de aprendizaje automático propuesto es una de regresión, es decir, predecir una variable continua. Para esto, se consideraron diversas arquitecturas base tales como las redes neuronales convolucionales (CNN), redes U-NET (Ronneberger et al., 2015), autocodificadores variacionales (VAE) (Kingma & Welling, 2022) y *vision transformers* (ViT) (Dosovitskiy et al., 2021).

Los *vision transformers* son el resultado de aplicar la arquitectura de transformadores (Vaswani et al., 2023) a la visión por computador. Aunque muy populares, y con gran potencial en tareas de clasificación y generación de imágenes, tales como *DLSS 4* de *Nvidia* (NVIDIA Corporation, 2025), son difíciles de implementar y computacionalmente costosos. Se consideró utilizar modelos preentrenados, o alternativas más ligeras y fáciles de implementar tales como *ConvMix* (Trockman & Kolter, 2022), pero finalmente se decidió no utilizar esta arquitectura.

3.1.1. U-NET

Las redes U-Net son una arquitectura de red neuronal convolucional basada en la estructura de una red neuronal convolucional tradicional, pero con la adición de capas de *upsampling* (aumento de resolución) para permitir la reconstrucción de la entrada original, además de conexiones *skip* (saltos) que permiten propagar la información entre distintos niveles de la red. De este modo, la red primero reduce la resolución de los datos a la vez que aumenta el número de dimensiones de los mismos a medida que se propagan por la red «hacia abajo». Una vez que se alcanza la resolución mínima, decimos que hemos llegado al «cuello de botella», donde se aplica una convolución adicional. Después, la red comienza a aumentar la resolución de los datos a medida que se propagan por la red «hacia arriba», mientras que se disminuye el número de dimensiones de los mismos.

Estas redes son muy populares en tareas de segmentación de imágenes (Ronneberger et al., 2015), clasificación, y generación de imágenes (Mei, 2024).

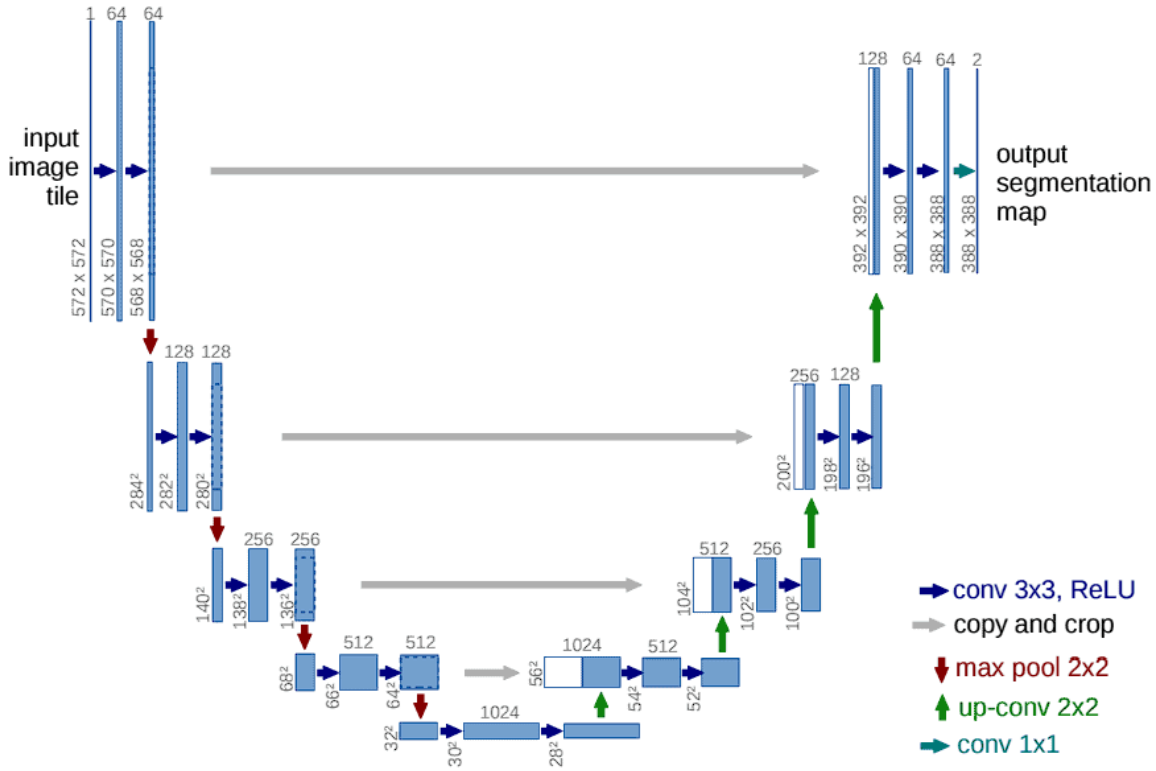

Figura 6: Arquitectura U-Net (Ronneberger et al., 2015)

3.1.2. Autocodificadores Variacionales

Los autocodificadores variacionales (VAE) son una arquitectura de red neuronal basada en la teoría de la probabilidad y la estadística que permite generar nuevas muestras a partir de un espacio latente aprendido durante el entrenamiento. Normalmente, se pretende que las muestras generadas sean similares a las muestras originales, pero en este trabajo se busca generar muestras que actuen como un mapa de varianzas. Siguen una arquitectura muy similar a la de las redes U-Net, con la diferencia de que no suelen usar conexiones *skip* y que, una vez alcanzada la resolución mínima, los datos son pasados por dos capas que generan la varianza y la media de una distribución normal. Posteriormente, se puede muestrear esta distribución para obtener muestras que son pasadas por capas de *upsampling* para aumentar su resolución hasta producir una salida de tamaño similar al de la entrada original.

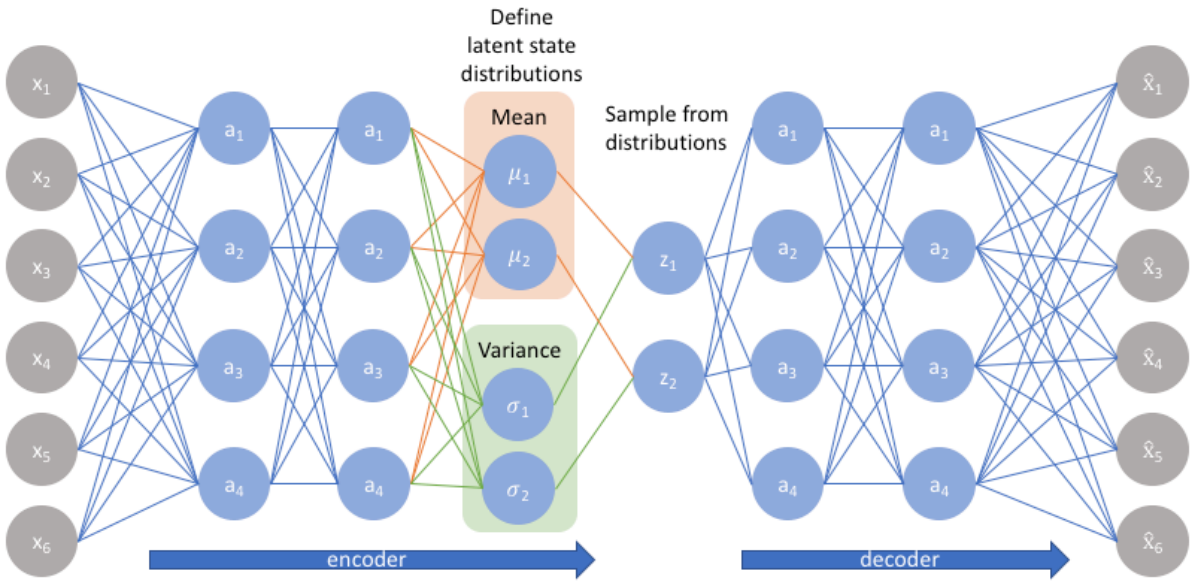

Figura 7: Arquitectura VAE (Jordan, 2018)

No obstante, experimentos inciales mostraron que los resultados producidos por una arquitectura VAE en el contexto de nuestro trabajo no eran del todo satisfactorios, siendo superados por la arquitectura U-Net. Dado esto, y su dificultad de implementación en comparación con la arquitectura U-Net, se ha optado por utilizar una arquitectura U-Net.

Nuestro modelo tomará como entrada una imagen y un mapa de profundidad. Opcionalmente se pueden proporcionar los bordes y laplaciano de la imagen y del mapa de profundidad, aunque de no ser proporcionados, el modelo los calculará internamente.

Estos datos de entrada son matrices $B, C \times 256 \times 256$, donde B es el tamaño del lote y C representa el número de dimensiones (canales), que será 3 en el caso de la imagen y 1 en el caso del resto. Los valores de estos datos estarán en el rango $[0, 1]$, a excepción de la profundidad, que mantendrá sus valores originales.

3.2. Módulos del modelo

Para el desarrollo del modelo, se han utilizado las siguientes técnicas en la mayoría de los módulos que constituyen el mismo:

Convolución linealmente separable Estas convoluciones, introducidas en Chollet (2017), son muy eficientes a la hora de procesar datos de alta dimensionalidad, ya que reducen la cantidad de parámetros y parámetros de aprendizaje necesarios. Esto se consigue utilizando una combinación de convoluciones punto a punto (*point-wise*) y en profundidad (*depth-wise*).

Modulo de atención espacial Este módulo, basado en el trabajo de Woo et al. (2018), permite al modelo aprender a dar importancia a ciertas regiones de los datos. Esto resulta de especial importancia tanto en las primeras capas, tales como los codificadores que describiremos más adelante, como en las capas más profundas, tales como el cuello de botella.

Dropout en 2 dimensiones La técnica de *dropout* en dos dimensiones (Tompson et al., 2015), es una extensión de la técnica original de *dropout* (Srivastava et al., 2014) para datos con estructura espacial, como imágenes o videos. Esta técnica ayuda a prevenir el sobreajuste del modelo y mejora su capacidad de generalización. Esto se consigue convirtiendo en cero aleatoriamente una fracción de las entradas de las capas, lo que ayuda a evitar que el modelo memorice patrones específicos en los datos de entrenamiento y mejora su capacidad de generalización.

3.2.1. Activaciones

El objetivo de una función de activación es introducir no linealidad en las capas del modelo, permitiendo que el modelo aprenda representaciones más complejas y no lineales de los datos de entrada. Una función de activación muy popular es la sigmoide, que produce salidas en el rango $[0, 1]$:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{6}$$

No obstante, esta función no resulta adecuada en las capas intermedias debido a un problema conocido como «desvanecimiento del gradiente», que ocurre cuando los gradientes se van volviendo muy pequeños a medida que se propagan hacia atrás en la red neuronal. Esto puede llevar a que el modelo no aprenda de manera efectiva.

Otra función de activación muy popular, en especial para los problemas de visión por computación, es la función de activación ReLU (Howard et al., 2017):

$$\mathrm{ReLU}(x) = \max(0, x) \tag{7}$$

Esta función, a pesar de su simplicidad, resulta ser sorprendentemente efectiva en muchas aplicaciones, por lo que es nuestra primera elección en este trabajo. Sin embargo, también tiene algunas limitaciones. Por ejemplo, la función ReLU no tiene una derivada continua en $x = 0$, lo que puede llevar a problemas de estabilidad durante el entrenamiento.

Existen múltiples variantes de la función ReLU, como la función *PReLU*, que introduce un parámetro aprendido para controlar la pendiente negativa, o la función *Swish* (Ramachandran et al., 2017):

$$\begin{aligned} \mathrm{PReLU}(x) &= \max(\alpha x, x) \\ \mathrm{Swish}(x) &= x\sigma(x) \end{aligned} \tag{8}$$

De entre todas las variantes y evoluciones de la función *ReLu*, la que consideraremos en este trabajo es la función *GELU* (Hendrycks & Gimpel, 2023), que ha demostrado superar a la función *ReLU* en términos de precisión y velocidad de convergencia en algunas tareas, con el inconveniente de ser más costosa computacionalmente.

$$\mathrm{GELU}(x) = xP(X \leq x), X \sim N(0, 1) \tag{9}$$

Por último, otra función de activación que consideraremos en este trabajo es la función *Siren* (Sitzmann et al., 2020), la cual, gracias a sus parámetros y naturaleza periódica, ha demostrado ser capaz de modelar funciones complejas y no lineales de manera eficiente.

$$\mathrm{Siren}(x) = \sin(w_0 x + w_1) \tag{10}$$

3.2.2. Codificadores

Dado que la naturaleza de estos datos es muy diferente, y en el caso de la profundidad, está en un rango distinto al de los otros elementos de la entrada, se ha implementado un módulo *Encoder* (codificador) que, una vez inicializado para un número de canales C, realiza una serie de operaciones en su entrada y produce una salida con 8 canales. Estas operaciones consisten en una serie de capas convolucionales linealmente separables (Chollet, 2017) normalizadas con normalización de instancias (Ulyanov et al., 2017), seguidas de la función de activación configurada. Finalmente, aplicamos *dropout* en 2 dimensiones (Tompson et al., 2015) con una probabilidad de 0.2 para ayudar a reducir la posibilidad de sobreajuste (Srivastava et al., 2014) y aplicamos un mecanismo de atención espacial basado en *CBAM* (*Convolutional Block Attention Module*) (Woo et al., 2018) para mejorar la capacidad de captura de información en las regiones más importantes.

El modelo dispone de 4 codificadores, uno para la imagen con 3 canales de entrada, uno para el par de bordes y laplaciano de la imagen con 2 canales de entrada, uno para el mapa de profundidad con 1 canal de entrada, y uno para el par de bordes y laplaciano del mapa de profundidad con 2 canales de entrada.

Una vez pasados los datos de entrada por su codificador correspondiente, disponemos de 4 matrices con 8 canales cada una, que concatenamos en una única matriz de 32 canales. Esta matriz pasará a través de una serie de bloques convolucionales para reducir su tamaño y capturar información progresivamente más compleja.

3.2.3. Bloques convolucionales

Otro de los módulos implementados es el *ConvBlock* (bloque convolucional), del cual podemos configurar dimensiones de entrada y salida, así como la función de activación a utilizar y una probabilidad p.

Una vez inicializado, los datos de entrada pasan por un atajo: una convolución con un tamaño de *kernel* de 1x1 que transforma la entrada al número de canales de salida especificado. Si la probabilidad p es mayor que 0, determinamos con esa misma probabilidad si devolver el resultado de aplicar el atajo. Si la probabilidad p es 0, o si se determina no aplicar el atajo, el bloque convolucional realiza una serie de operaciones en su entrada y produce una salida con las dimensiones de salida configuradas.

Estas operaciones consisten en una capa convolucional linealmente separable normalizadas con normalización de lote (Ioffe & Szegedy, 2015), seguida de la función de activación configurada, y *dropout* en dos dimensiones con una probabilidad de 0.2 para ayudar a reducir la posibilidad de sobreajuste. Finalmente, aplicamos una conexión residual sumando el resultado del atajo con la salida del bloque convolucional.

La salida de cada bloque pasa por una función de *max pooling* con un *kernel* de tamaño 2×2. Esta función escoje el valor máximo en cada ventana de tamaño 2×2, de modo que el tamaño de la salida se reduce a la mitad en ambas dimensiones. Estos datos se almacenan en memoria para su posterior uso como conexión *skip* con los datos a medida que se escalan.

Una vez aplicados todos los bloques convolucionales y llegados al cuello de botella de la red, aplicamos una convolución linealmente separable, atención espacial y *dropout* en dos dimensiones con una probabilidad de 0.2. Finalmente, comenzamos a escalar la resolución de los datos usando bloques de escalado.

3.2.4. Bloques de escalado

Este modulo nos permite duplicar la resolución de su entrada mediante la aplicación de la técnica de *Pixel Shuffling* (Shi et al., 2016), la cual consiste en reorganizar los elementos de una matriz de entrada en una matriz de salida con una resolución mayor y una dimensionalidad menor. En concreto, el factor de escalado r determina la cantidad de veces que se aumenta la resolución de la matriz de entrada, a la vez que el número de dimensiones de salida se reduce en un factor de r^2.

Tomando inspiración de la tecnica *Bilineal Additive Upsampling* (BAU) propuesta en Wojna et al. (2019), utilizamos interpolación bilineal para aumentar la resolución de la matriz de entrada tras reducir su dimensionalidad para que esta sea igual a la de la salida de la técnica de *Pixel Shuffling*, y finalmente combinamos la salida de esta operación con la salida de la técnica de *Pixel Shuffling* con un parámetro aprendido α:

$$\mathrm{up}(x) = \alpha \ \mathrm{PixelShuffle}(x) + (1 - \alpha) \ \mathrm{BilinealUp}(\mathrm{DownSample}(x)) \qquad (11)$$

Finalmente, aplicamos normalización de lote y la función de activación configurada.

3.2.5. Pirámide de características

A medida que escalamos los datos y aplicamos la conexión *skip* correspondiente, pasamos la salida de cada bloque de escalado por una convolución linealmente separable que reduce la dimensionalidad de la matriz de entrada a 1, y posteriormente escalamos este resultado nuevamente utilizando interpolación bilineal para que coincida con la resolución de la entrada original. Esto da lugar a una colección de matrices de tamaño $1 \times H \times W$ a la que llamamos **pirámide de características**.

En esta pirámide, los primeros elementos son los que han sufrido el escalado más agresivo, dando lugar a una pirámide de características donde los elementos en la base – es decir, los primeros elementos – capturan detalles más generales, y los elementos en la parte superior – los últimos elementos – capturan detalles más finos. En total, se forma una pirámide de 4 elementos.

Utilizando un parámetro aprendido del modelo w, un vector de 4 pesos w_i inicializados como $[0.1, 0.2, 0.3, 0.4]$ para dar más relevancia a los detalles más finos. Esto permite al modelo aprender que niveles de detalle son más importantes para la generación del mapa de varianza. Podemos calcular la suma ponderada de los elementos de la pirámide con el resultado de aplicar la función *softmax* a los pesos de w. La función *soft max* hace que la suma de los pesos sea igual a 1:

$$\mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_j^N e^{x_j}} \tag{12}$$

3.3. Salida y estructura

Finalmente, recortamos los valores de salida al rango $[-6, 6]$ e interpretamos la salida no como una matriz de varianzas, sino como una matriz del logaritmo de las mismas, de modo que para obtener el mapa de varianza deseado debemos aplicar la función exponencial a la salida:

$$\exp(x) = e^x \tag{13}$$

Esto aumenta la estabilidad numérica del modelo, ya que la función exponencial ayuda a suavizar los gradientes. Además, los valores de las varianzas estarán entre $[e^{-6}, e^6]$ (aproximadamente $[0.0025, 403.4288]$), dando al modelo flexibilidad a la hora de estimar varianzas, sin permitir que las mismas sean negativas o extremadamente grandes.

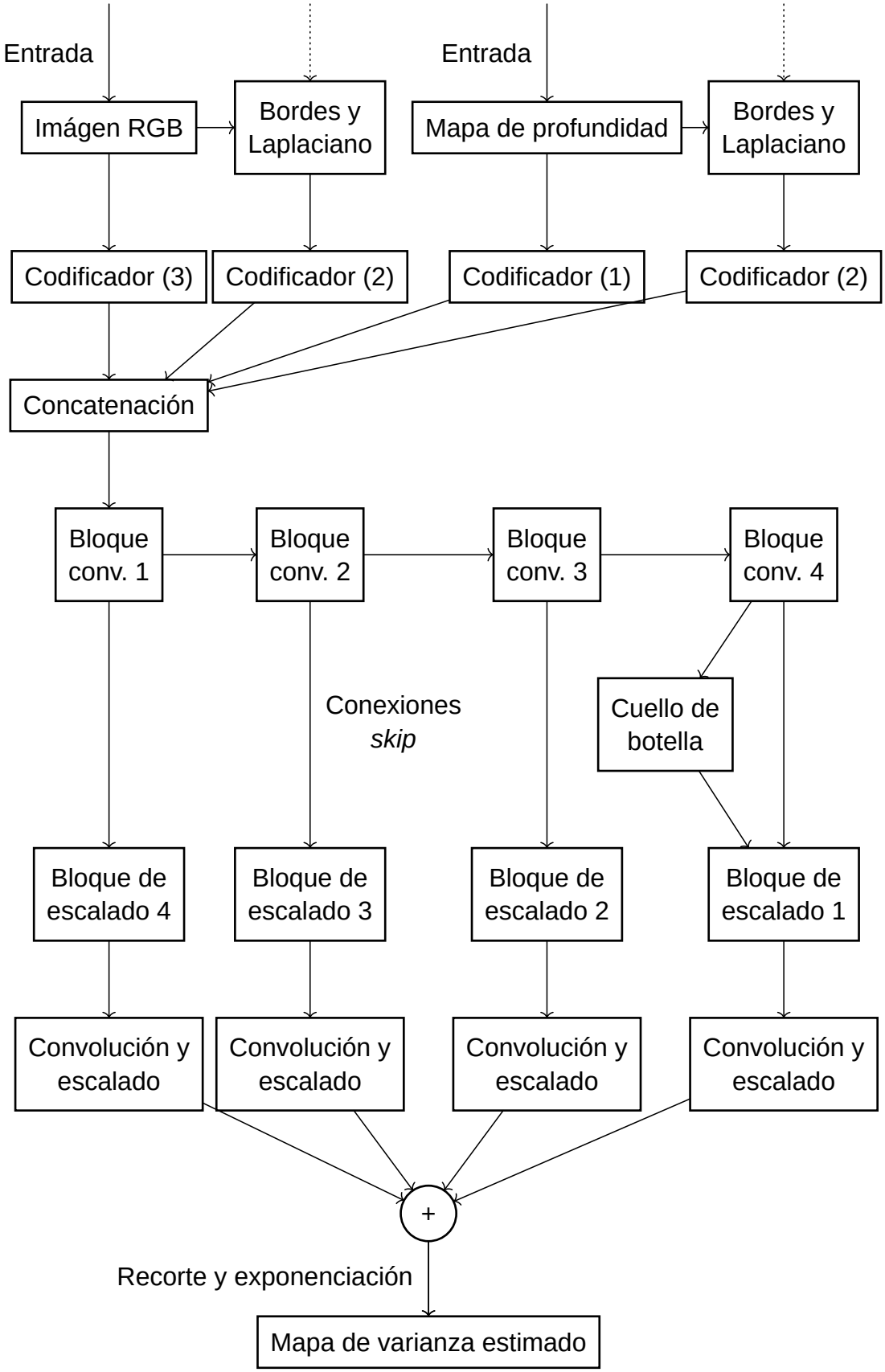

Figura 8: Estructura general del modelo

3.4. Búsqueda de hiperparámetros

Un hiperparámetro es una variable que nos permite configurar el comportamiento del modelo. Estos parámetros no son aprendidos por el modelo, sino que se establecen antes de comenzar el entrenamiento. Algunos hiperparámetros comunes son el tamaño de lote, la tasa de aprendizaje y la regularización L2 (*weight decay*).

En este trabajo, decidimos fijar el tamaño de lote a 32 elementos y la regularización L2 a 0.03. Nos centraremos en la búsqueda de la tasa de aprendizaje, optimizador, función de activación y peso de la pérdida de referencia – introducida en la Sección 3.5.1 – que maximicen el rendimiento del modelo.

Existen múltiples métodos para buscar los hiperparámetros óptimos. Estos incluyen búsqueda combinatoria, búsqueda aleatoria y búsqueda bayesiana. La búsqueda combinatoria prueba combinaciones de hiperparámetros predefinidas, lo cual puede resultar útil en espacios de búsqueda reducidos, pero crece de manera exponencial con el número de hiperparámetros y sus valores posibles. La búsqueda aleatoria prueba valores aleatorios de los hiperparámetros, lo cual puede ser útil en espacios de búsqueda grandes, pero puede resultar en valores subóptimos, ya que no sigue una estrategia sistemática. La búsqueda bayesiana utiliza una estrategia de búsqueda basada en la teoría de la probabilidad, lo cual puede ser útil en espacios de búsqueda grandes y complejos. En este trabajo utilizamos la librería *Optuna* con el algoritmo de optimización bayesiana *Tree-structured Parzen Estimator* (TPE) para encontrar los hiperparámetros óptimos.

Los principales hiperparámetros son la función de activación, el optimizador y el valor de la tasa de aprendizaje. En nuestro caso, también introducimos el peso de la pérdida de referencia, el cual es introducido en la Sección 3.5.1. Durante los experimentos se usará un 30% de los datos, un tamaño de lote de 32 muestras y se entrenará durante un máximo de 5 épocas. Si *Optuna* detecta que el experimento no dará buenos resultados, se detendrá antes de completar las 5 épocas y pasará a ejecutar el siguiente experimento. La justificación para este número de épocas es que pretendemos determinar los valores que dan buenos resultados frente a los que no lo hacen, sin necesidad de simular un entrenamiento completo.

Para determinar el rendimiento – es decir, la calidad de la salida del modelo – utilizaremos la siguiente métrica durante las fases de evaluación y prueba:

$$Q(x, d, D) = \frac{(d - D)^2}{F(x, d)} \tag{14}$$

Cuanto más se aproxime esta métrica a 1, más precisas serán las estimaciones producidas por el modelo. No obstante, dado que *Optuna* necesita una métrica a maximizar o minimizar, podemos realizar la siguiente transformación:

$$Q_m(x, d, D) = |1 - Q(x, d, D)| = \left|1 - \frac{(d - D)^2}{F(x, d)}\right| \tag{15}$$

De este modo, cuando la proporción entre la varianza real $(d - D)^2$ y la estimada $F(x, d)$ se aproxima a 1, la métrica $Q_m(x, d, D)$ es mínima en 0, y podemos permitir a *Optuna* buscar valores de los hiperparámetros que minimicen esta métrica.

3.4.1. Barrido

Para las funciones de activación, consideramos usar *ReLu*, *GELU*, y *Siren*. Para los optimizadores, consideramos usar *AdamW* y *Prodigy* (Mishchenko & Defazio, 2024). Los valores de la tasa de aprendizaje dependen del optimizador: para *AdamW*, consideramos usar valores en el rango $[10^{-6}, 10^{-2}]$, distribuidos de forma logarítmica. En el caso de *Prodigy*, el valor se fija en 1. Finalmente, consideramos usar un peso de la pérdida de referencia en el rango $[10^{-6}, 1]$, distribuido de forma logarítmica. Realizamos un barrido – esto es, la ejecución de los experimentos – con 25 experimentos, tras el cual *Optuna* nos indica la importancia de cada hiperparámetro en el proceso de entrenamiento a la hora de minimizar el valor objetivo:

Función de activación	Optimizador	Peso de la pérdida de referencia
3.283×10^{-3}	0.282	0.714

Tabla 1: Importancias de los hiperparámetros determinados durante la búsqueda

Como podemos observar, el peso de la pérdida de referencia resulta de crucial importancia en el proceso de entrenamiento, mientras que la elección de la función de activación parece no tener una importancia significativa. No resulta posible ver la importancia de la elección de la tasa de aprendizaje dado que es un parámetro condicional.

Una vez observados los resultados del barrido de hiperparámetros, decidimos utilizar los siguientes valores para los hiperparámetros durante el entrenamiento ya que, según *Optuna*, son los que mejor minimizan el valor objetivo:

Función de activación	Optimizador	Tasa de aprendizaje	Peso de la pérdida de referencia
ReLu	*AdamW*	$1.189 \cdot 10^{-3}$	$1.318 \cdot 10^{-4}$

Tabla 2: Valores de los hiperparámetros recomendados por *Optuna*

3.5. Entrenamiento

En un lote de datos dado, cada muestra contiene una imagen a la que denotamos como x, sus bordes x_e y laplaciano x_l, un mapa de profundidad original D, sus bordes D_e y laplaciano D_l, un mapa de profundidad estimada d y sus bordes d_e y laplaciano d_l. No obstante, dado que el modelo puede cálcular los bordes y laplacianos internamente, asumiremos que los pasamos como entradas, pero no los escribiremos para ahorrar espacio.

Por un lado, obtenemos un mapa de **varianzas estimadas**, el cual denotamos como $\hat{z_e}$:

$$\hat{z_e} = F(x, d) \tag{16}$$

Donde $\hat{z}_{e_{ij}}$ es la incertidumbre estimada para el valor de profundidad d_{ij} estimado por el modelo de estimación de profundidad en el punto (i, j).

Por otro lado, obtenemos un mapa de **varianzas de referencia**, el cual denotamos como $\hat{z_r}$:

$$\hat{z_r} = F(x, D) \tag{17}$$

Esto es, la incertidumbre que estima el modelo al evaluarse con la profundidad real. Este término permite al modelo aprender a producir varianzas pequeñas cuando la profundidad con la que se evalúa es de alta calidad, es decir, cercana a la real.

3.5.1. Funciones de pérdida

Tal como se propone en Baumann et al. (2023), calculamos la pérdida de la varianza estimada utilizando una función de pérdida derivada de la distribución de Laplace, ya que ofrece mejores resultados que su contraparte gaussiana en tareas de visión por computador (Kendall & Gal, 2017):

$$\mathcal{L}_{\text{Lap}}(x, d, D) = \mathbb{E}\left[\ln(F(x, d)) + \frac{|d - D|}{F(x, d)}\right] \tag{18}$$

Un aspecto importante a considerar es que, aunque usemos una función de pérdida derivada de la distribución de Laplace durante el entrenamiento, seguimos interpretando los valores de $\hat{z}_e$ como varianzas de una distribución gaussiana, y la función de pérdida utilizada durante la fase de evaluación es la derivada de la distribución gaussiana, ya que es más coherente con nuestra asunción de que la profundidad sigue una distribución gaussiana:

$$\mathcal{L}_{\text{Gauss}}(x, d, D) = \mathbb{E}\left[\ln(F(x, d)) + \frac{(d - D)^2}{2F(x, d)^2}\right] \tag{19}$$

Ya que la varianza entre la profundidad real y si misma debería ser cero, la emplearemos como término de penalización llamado **pérdida de referencia** escalada por un hiperparámetro al que llamaremos **peso de la pérdida de referencia**, denotado por λ. Es importante acotar este hiperparámetro para evitar que el modelo dé demasiada importancia a la varianza de referencia, haciendo que devuelva ceros ante cualquier entrada.

Finalmente, la función que buscamos minimizar es:

$$\mathcal{L}(x, d, D) = \mathbb{E}\left[\ln(F(x, d)) + \frac{|d - D|}{F(x, d)}\right] + \lambda\mathbb{E}[F(x, D)] \tag{20}$$

3.5.2. Técnicas de entrenamiento

De cara a mejorar la estabilidad del entrenamiento, se ha empleado la técnica de *gradient clipping* (recorte de gradientes) (Zhang et al., 2020) para evitar que los gradientes exploten durante el entrenamiento. Para ello, se ha establecido un límite máximo para la magnitud del gradiente igual a 1, y cualquier gradiente cuya magnitud supere este límite se recorta a este valor.

También usamos *weight decay* (regularización L2) (Loshchilov & Hutter, 2019) en el optimizador *AdamW* para penalizar valores grandes de los parámetros, lo que ayuda a evitar el sobreajuste. Decidimos fijar este valor en 10^{-3} en lugar del valor por defecto de 10^{-2} para permitir una mayor flexibilidad en la optimización.

Por otro lado, empleamos una técnica conocida como *Cosine Annealing Learning Rate Scheduling* (Loshchilov & Hutter, 2017), la cual ajusta gradualmente la tasa de aprendizaje durante el entrenamiento, lo que puede mejorar la convergencia y la generalización del modelo. La tasa de aprendizaje se reduce desde el valor inicial η_0 hasta un valor final η_T de 0 a lo largo de T pasos:

$$\eta_t = \eta_T + \frac{1}{2}(\eta_0 - \eta_T)\left(1 + \cos\left(\frac{t\pi}{T}\right)\right) = \frac{\eta_0}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right) \tag{21}$$

3.5.3. Proceso de entrenamiento

Entrenamos el modelo durante 50 épocas con todos los datos disponibles, un tamaño de lote de 32 muestras y los mejores hiperparámetros encontrados por *Optuna*: la función de activación *ReLU*, el optimizador *AdamW* con una tasa de aprendizaje de $1.189 \cdot 10^{-3}$ y un peso de la pérdida de referencia de $1.318 \cdot 10^{-4}$.

A medida que avanza el entrenamiento, cada vez que se mejora la pérdida en el conjunto de validación, se guarda el modelo en discom, de modo que, incluso si en un momento dado la pérdida comienza a aumentar, el modelo mejorado se conserva y se puede utilizar para realizar predicciones más precisas en el futuro.

A continuación se muestran las evoluciones de las pérdidas durante el entrenamiento, así como una muestra de las predicciones del modelo en el conjunto de prueba.

3.5.4. Pérdidas durante el entrenamiento

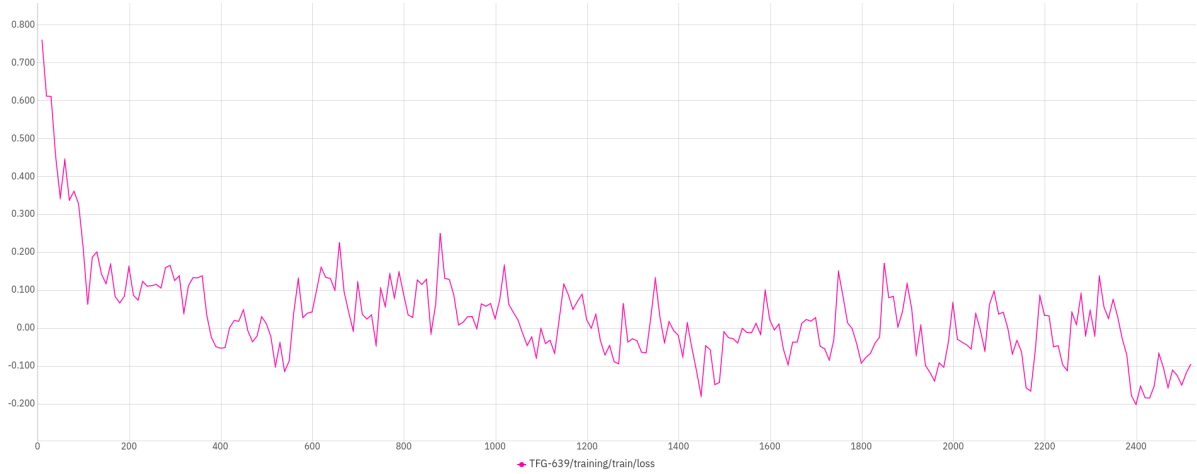

Figura 9: Pérdida durante el entrenamiento

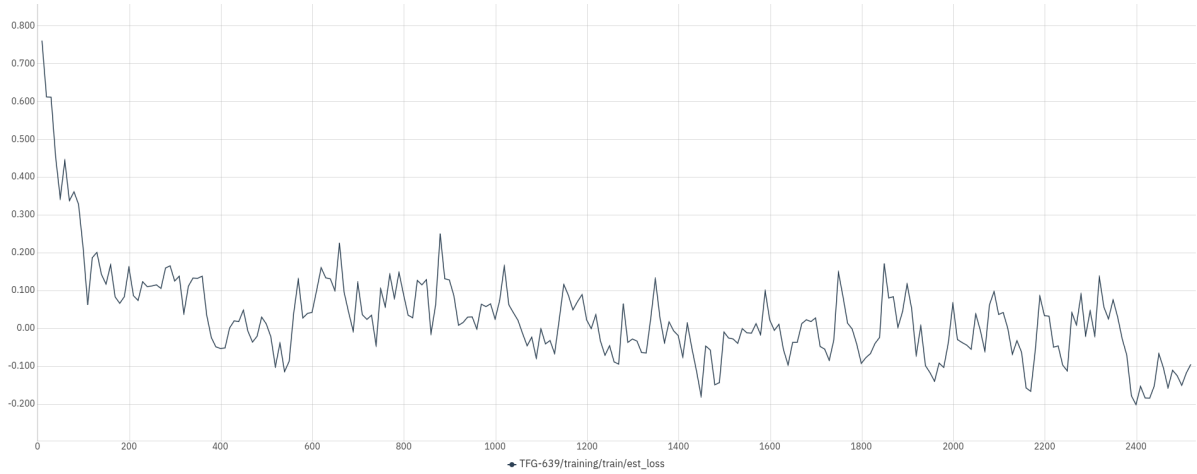

Figura 10: Pérdida de estimación durante el entrenamiento

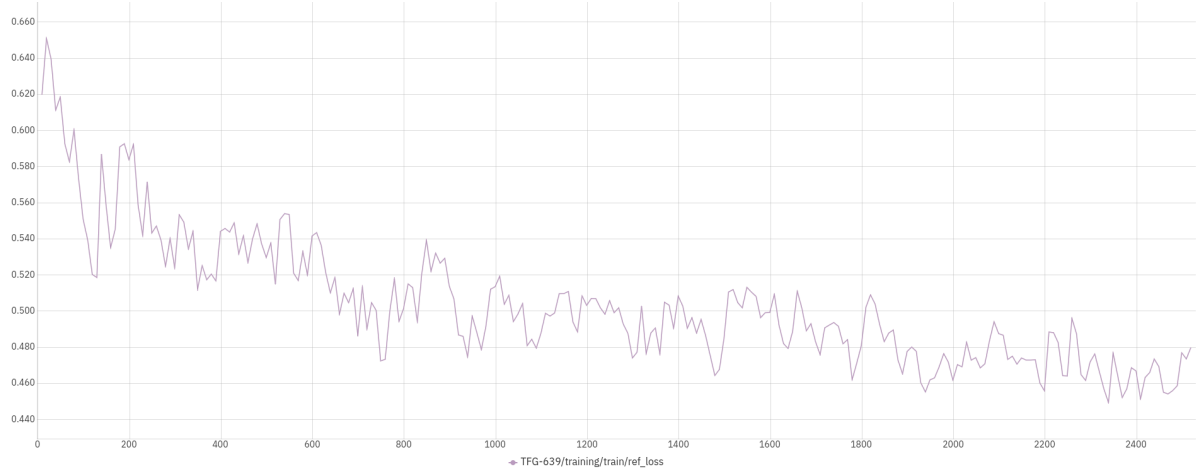

Figura 11: Pérdida de referencia durante el entrenamiento

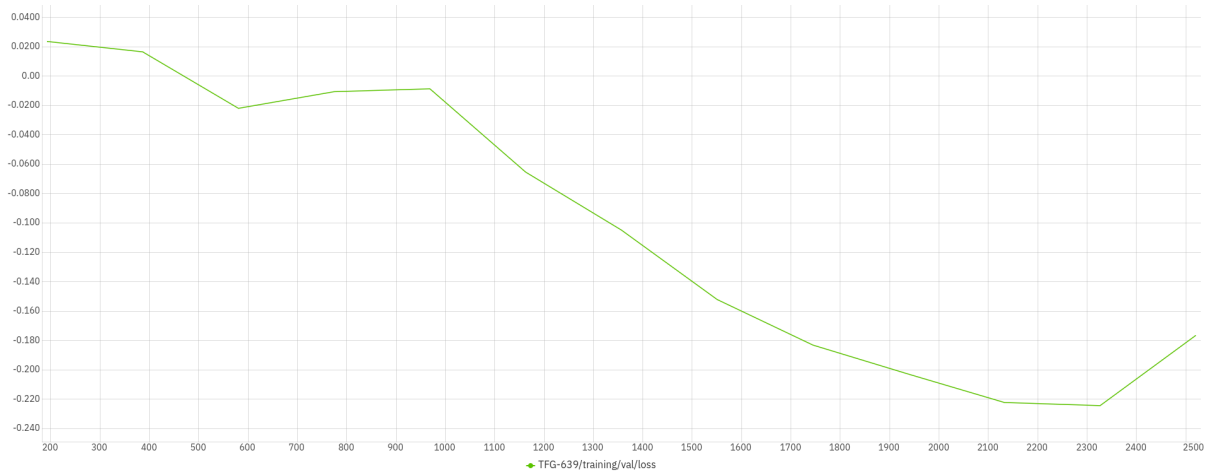

Figura 12: Pérdida de validación durante el entrenamiento

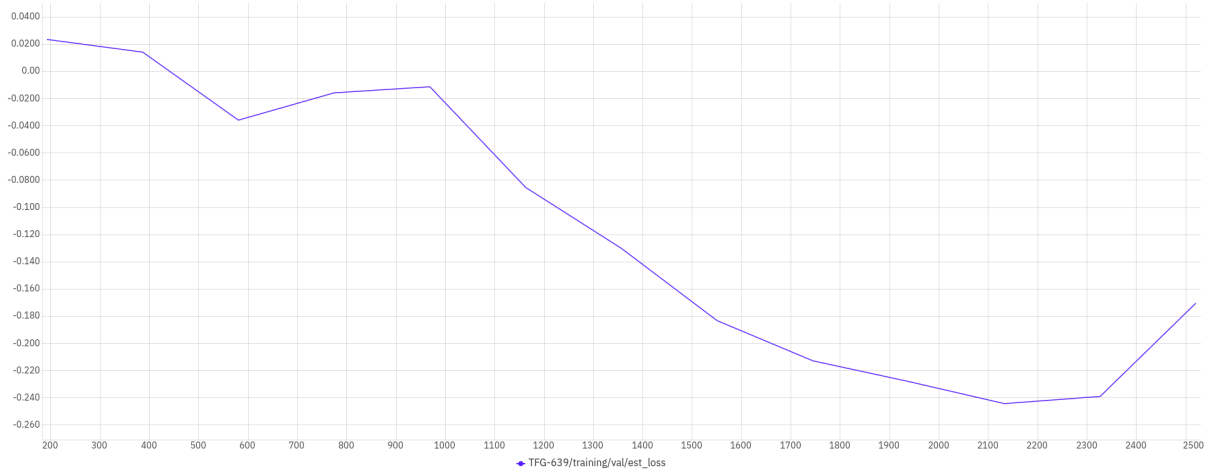

Figura 13: Pérdida de estimación de validación durante el entrenamiento

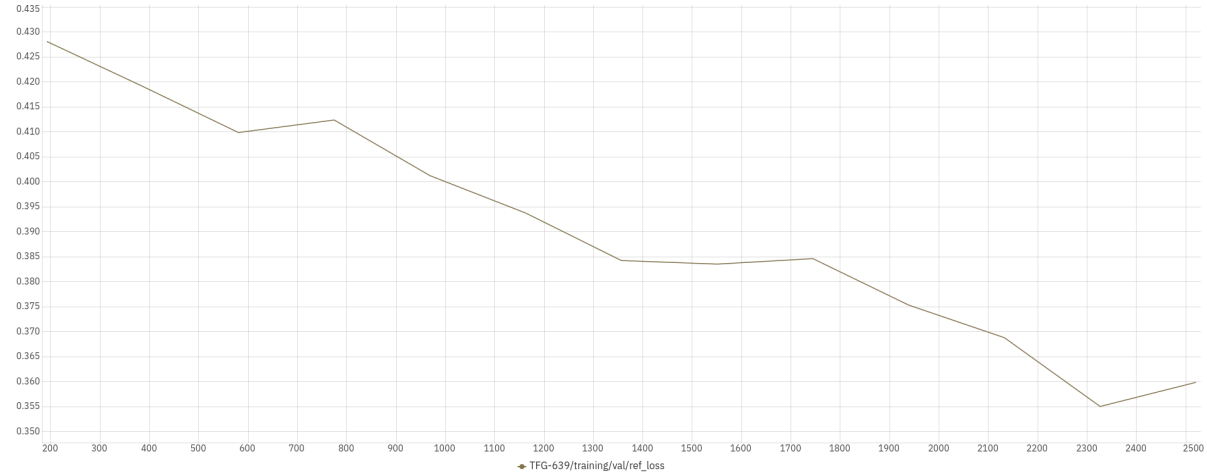

Figura 14: Pérdida de referencia de validación durante el entrenamiento

Tras el repunte en la pérdida de validación, el modelo comienza a sobreajustar, por lo que el entrenamiento se detiene de forma anticipada.

3.5.5. Predicciones del modelo con datos de validación

Figura 15: Imagen de muestra del conjunto de validación

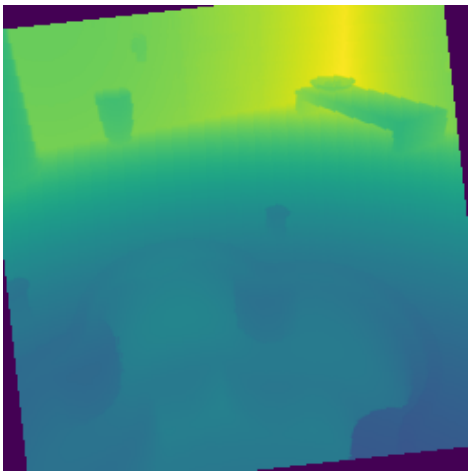

Figura 16: Profundidad real de la muestra del conjunto de validación

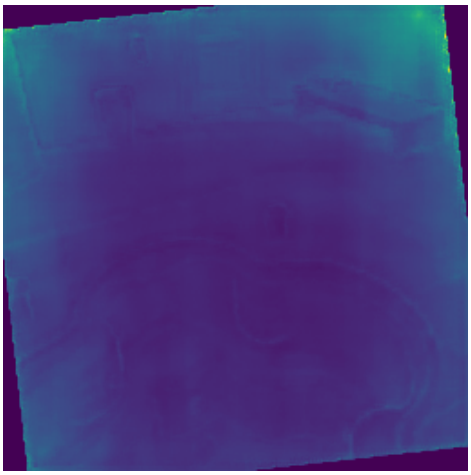

Figura 17: Varianza de referencia de la muestra del conjunto de validación

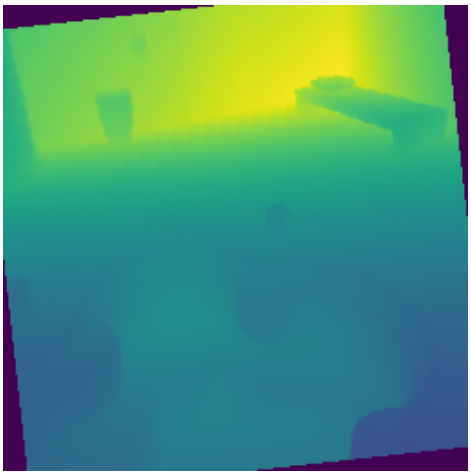

Figura 18: Profundidad estimada para la imagen del conjunto de validación

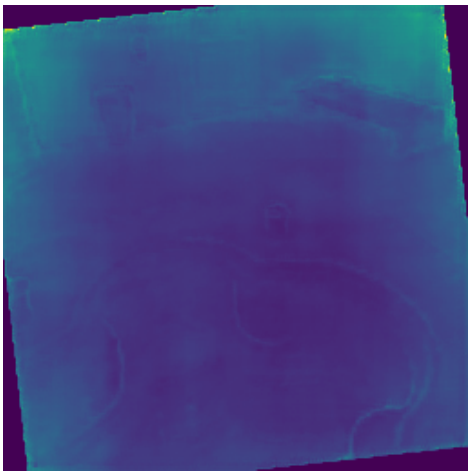

Figura 19: Varianza estimada en el conjunto de validación

3.5.6. Predicciones del modelo con datos de prueba

Figura 20: Imagen de muestra del conjunto de prueba

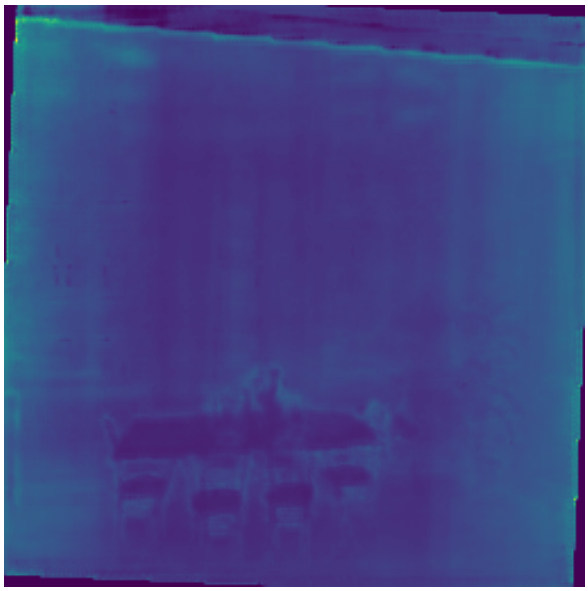

Figura 22: Varianza estimada del conjunto de prueba

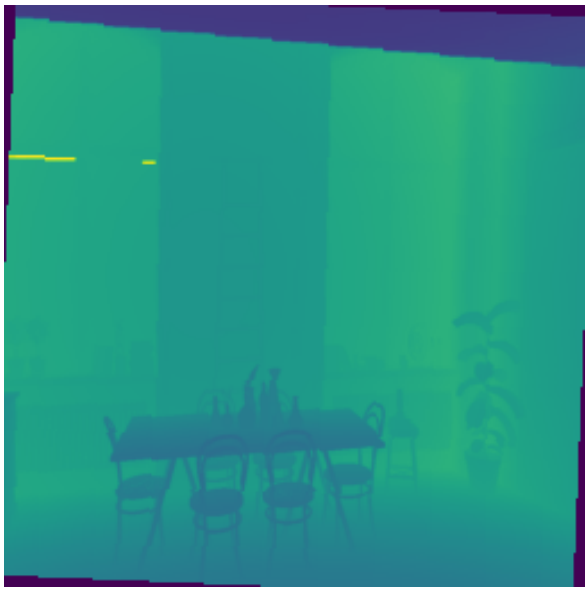

Figura 21: Profundidad real de la muestra del conjunto de prueba

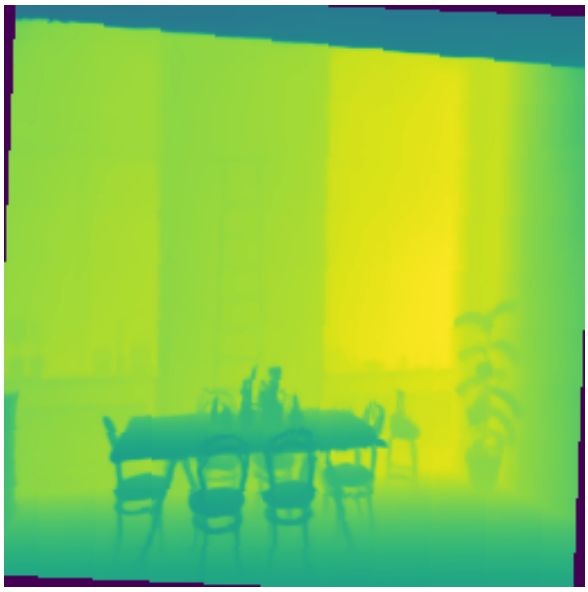

Figura 23: Profundidad estimada para la imagen del conjunto de prueba

Una vez evaluado el conjunto de prueba, obtenemos una pérdida de estimación relativamente mayor a las del conjunto de validación, mientras que la de referencia se mantiene en un rango similar al las del mismo, indicando que el peso de la pérdida de referencia podría ser demasiado elevado, y podría necesitar de ajustes para mejorar el rendimiento del modelo en el conjunto de prueba.

4

Cadena de procesamiento

Finalmente, para implementar el proceso de estimación de incertidumbre y refinado de profundidad decidimos implementar una cadena de procesamiento (*Pipeline*) que se encargará de tomar la imagen capturada por una cámara RGB-D como entrada, separarla en sus canales RGB y de profundidad y pasar los canales RGB a *Depth Pro* para obtener el mapa de profundidad estimado, tal como se describe en la Figura 2.

A continuación, se calcularan los mapas de varianza del mapa de profundidad capturado por la cámara y del mapa de profundidad estimado. Dado que los mapas de profundidad son matrices $1 \times 256 \times 256$, y que los mapas de varianza son matrices $1 \times 256 \times 256$, podemos concatenarlas en una matriz $2 \times 256 \times 256$ donde la primera dimensión (la de la profundidad) será interpretada como una media, y la segunda dimensión será la de la varianza. De este modo, cada matriz $2 \times 256 \times 256$ es un mapa de distribuciones.

Dado que disponemos de dos de estos mapas de distribuciones: el asociado a la profundidad capturada por la cámara y el asociado a la profundidad estimada. Podemos multiplicarlos elemento a elemento para obtener un mapa refinado:

$$N(\mu_{ij}, \sigma_{ij}^2) = N\left(D_{ij}, \hat{z}_{D_{ij}}\right) N\left(d_{ij}, \hat{z}_{d_{ij}}\right) \tag{22}$$

Donde D es la profundidad capturada por la cámara, d es la profundidad estimada, $\widehat{z_D}$ es la varianza estimada de la profundidad capturada por la cámara, y $\hat{z_d}$ es la varianza estimada de la profundidad estimada.

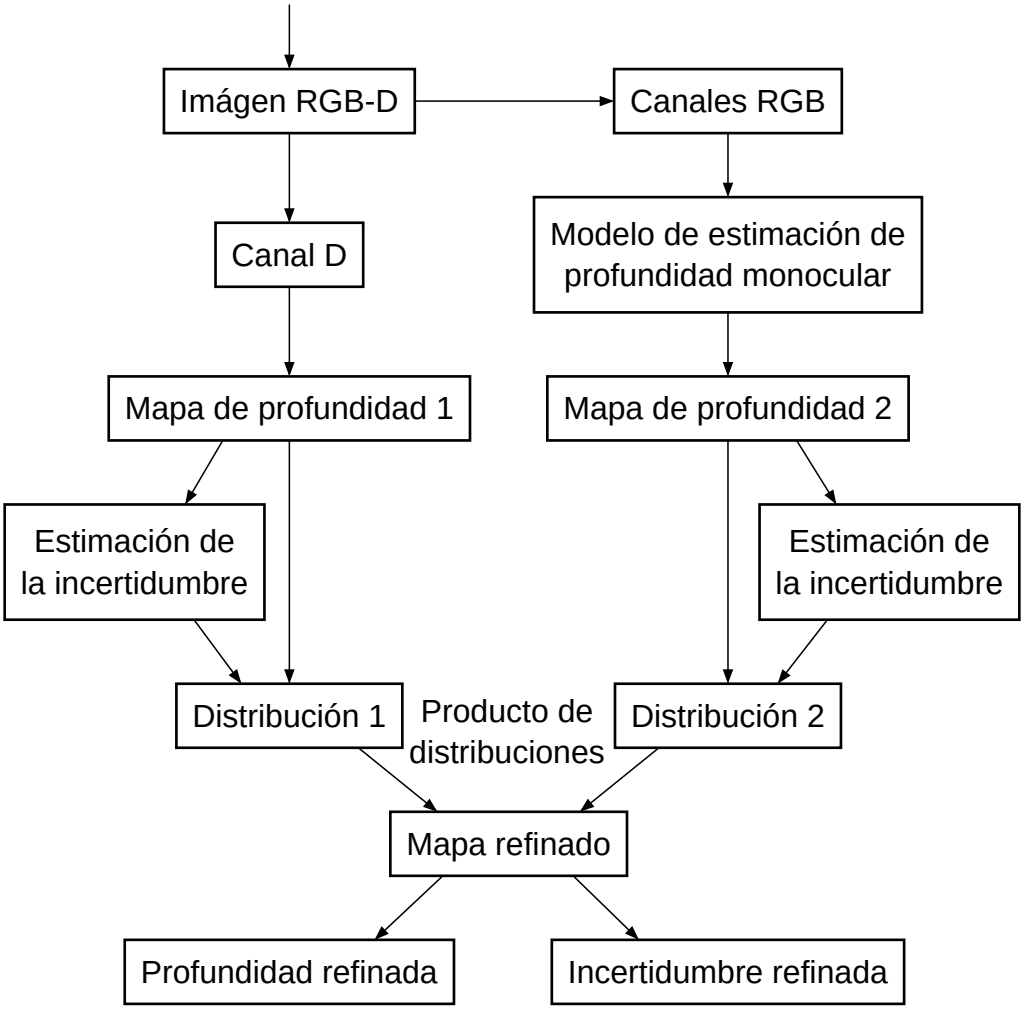

Figura 24: Vista general de la cadena de procesamiento

5

Resultados y Conclusiones

En este trabajo se ha propuesto un método para la estimación de la profundidad de una escena mediante cámaras RGB-D y modelos de estimación de profundidad monocular mediante la estimación de incertidumbre de la profundidad de ambas fuentes de información. Además, se ha propuesto incluir la incertidumbre asociada a los valores de profundidad a la hora de considerar la profundidad de una escena, ya que esto puede resultar fundamental en el entendimiento de la misma. El modelo propuesto para la estimación de la varianza y las técnicas utilizadas para entrenar el mismo se basan en el estado del arte de la visión por computador y otras técnicas que han demostrado su eficacia a lo largo de los años.

5.1. Aplicaciones

Las aplicaciones del modelo desarrollado se extienden a diversos ámbitos donde la fiabilidad de las estimaciones de profundidad es crítica:

- **Integración inteligente de sensores:** combinar medidas de diferentes dispositivos en función de la incertidumbre asociada a cada una.
- **Re-adquisición de datos bajo condiciones poco fiables:** si el sistema detecta que la incertidumbre en una zona determinada supera un umbral, se puede decidir volver a capturar datos desde otra posición o ángulo. Esto puede resultar útil en aplicaciones de inspección, cartografía o exploración, donde es importante garantizar la calidad de la información obtenida.
- **Planificación de movimientos segura:** al disponer no solo de un mapa de profundidad, sino también de la confianza asociada, un robot o vehículo autónomo puede evitar zonas con alta incertidumbre y escoger trayectorias más seguras. Esto es especialmente relevante en navegación en entornos desconocidos, o en la manipulación de objetos en situaciones donde un error de cálculo en la profundidad puede provocar colisiones.
- **Aplicaciones en visión aumentada y mixta:** en contextos de realidad aumentada o mixta, disponer de información fiable sobre la profundidad es crucial para la correcta superposición de elementos virtuales. La gestión explícita de la incertidumbre puede ayudar a mejorar la estabilidad y el realismo de estas aplicaciones.

5.2. Trabajo futuro

Existen varios caminos para continuar y mejorar el trabajo realizado:

- **Ejecución en tiempo real:** aunque el modelo es relativamente ligero, su uso en entornos prácticos depende en gran medida del rendimiento del modelo de estimación de profundidad monocular empleado. Un paso natural sería optimizar el sistema para alcanzar velocidades cercanas al tiempo real, lo que abriría la puerta a aplicaciones robóticas y de navegación en línea.
- **Exploración de modelos probabilísticos alternativos:** en este trabajo se asumió que la incertidumbre podía modelarse con distribuciones gaussianas. Sin embargo, otras distribuciones —como Laplace o mezclas de gaussianas— podrían representar mejor ciertos tipos de error y mejorar la calidad de la fusión de mapas.
- **Mejora de la optimización e hiperparámetros:** actualmente, el ajuste de parámetros como el peso de la pérdida de referencia se realiza de manera fija. Una posible mejora sería implementar mecanismos adaptativos que ajusten estos valores durante el entrenamiento en función del comportamiento del modelo, lo que podría reducir problemas de sobreajuste.
- **Aumento de datos más variado:** en los experimentos se usaron transformaciones relativamente simples para no comprometer la consistencia entre imagen y mapa de profundidad. Una extensión natural sería incorporar transformaciones más diversas — como cambios de iluminación o distorsiones— que, aunque dificultan el uso de mapas precalculados, pueden mejorar la capacidad de generalización del modelo en entornos reales.
- **Evaluación en datos del mundo real:** hasta ahora, los experimentos se han centrado en un conjunto de datos sintético con profundidad de referencia perfecta. Un siguiente paso importante es comprobar el rendimiento en escenarios reales, donde las medidas están afectadas por ruido y errores propios de los sensores. Esto permitiría evaluar hasta qué punto los mapas de incertidumbre ayudan a mejorar la fiabilidad en condiciones reales.
- **Extensión a otras tareas de visión por computador:** aunque el trabajo se centra en la estimación de profundidad, el mismo enfoque podría aplicarse a problemas relacionados, como la segmentación semántica o la detección de objetos, donde también resulta útil disponer de una medida explícita de incertidumbre.

Bibliografía

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-Generation Hyperparameter Optimization Framework. *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2623-2631.

Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., … Chintala, S. (2024, abril). PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. https://doi.org/10.1145/3620665.3640366

Baumann, A., Roßberg, T., & Schmitt, M. (2023,). *Probabilistic MIMO U-Net: Efficient and Accurate Uncertainty Estimation for Pixel-wise Regression*. https://arxiv.org/abs/2308.07477

Bochkovskii, A., Delaunoy, A., Germain, H., Santos, M., Zhou, Y., Richter, S. R., & Koltun, V. (2025,). Depth Pro: Sharp Monocular Metric Depth in Less Than a Second. *International Conference on Learning Representations*. https://arxiv.org/abs/2410.02073

Chollet, F. (2017, julio). Xception: Deep Learning With Depthwise Separable Convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021,). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. https://arxiv.org/abs/2010.11929

Falcon, W., & The PyTorch Lightning team. (2019, marzo). *PyTorch Lightning*. https://doi.org/10.5281/zenodo.3828935

The HDF Group. (2025,). *HDF5 for AI/ML: Unlocking the Power of Efficient Data Management*. https://www.hdfgroup.org/solutions/hdf5-for-ai-ml/

Hendrycks, D., & Gimpel, K. (2023,). *Gaussian Error Linear Units (GELUs)*. https://arxiv.org/abs/1606.08415

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017,). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. https://arxiv.org/abs/1704.04861

Ioffe, S., & Szegedy, C. (2015,). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. https://arxiv.org/abs/1502.03167

Jordan, J. (2018,). *Variational Autoencoders*.

Kendall, A., & Gal, Y. (2017,). *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?*. https://arxiv.org/abs/1703.04977

Khoshgoftaar, C. S. A. T. M. (2019,). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data 6, 60*. https://doi.org/10.1186/s40537-019-0197-0

Kingma, D. P., & Welling, M. (2022,). *Auto-Encoding Variational Bayes*. https://arxiv.org/abs/1312.6114

Kiureghian, A. D., & Ditlevsen, O. (2009). Aleatory or epistemic? Does it matter?. *Structural Safety*, *31*(2), 105-112. https://doi.org/https://doi.org/10.1016/j.strusafe.2008.06.020

Loshchilov, I., & Hutter, F. (2017,). *SGDR: Stochastic Gradient Descent with Warm Restarts*. https://arxiv.org/abs/1608.03983

Loshchilov, I., & Hutter, F. (2019,). *Decoupled Weight Decay Regularization*. https://arxiv.org/abs/1711.05101

Mei, S. (2024,). *U-Nets as Belief Propagation: Efficient Classification, Denoising, and Diffusion in Generative Hierarchical Models*. https://arxiv.org/abs/2404.18444

Mishchenko, K., & Defazio, A. (2024,). Prodigy: An Expeditiously Adaptive Parameter-Free Learner. *Forty-first International Conference on Machine Learning*. https://openreview.net/forum?id=JJpOssn0uP

Nathan Silberman, P. K., Derek Hoiem, & Fergus, R. (2012,). Indoor Segmentation and Support Inference from RGBD Images. *ECCV*.

NVIDIA Corporation. (2025,). *DLSS 4: Transforming Real-Time Graphics with AI*.

Ramachandran, P., Zoph, B., & Le, Q. V. (2017,). *Searching for Activation Functions*. https://arxiv.org/abs/1710.05941

Rerun Development Team. *Rerun: A Visualization SDK for Multimodal Data*. Rerun Technologies AB. https://www.rerun.io/

Ronneberger, O., Fischer, P., & Brox, T. (2015,). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. https://arxiv.org/abs/1505.04597

Rustler, L., Volprecht, V., & Hoffmann, M. (2025). *Empirical Comparison of Four Stereoscopic Depth Sensing Cameras for Robotics Applications*. https://doi.org/https://doi.org/10.1109/ACCESS.2025.3560810

Saxena, A., Schulte, J., & Ng, A. Y. (2007). Depth estimation using monocular and stereo cues. *International Joint Conference on Artificial Intelligence*, 2197-2203. http://ijcai.org/Proceedings/07/Papers/354.pdf

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., & Wang, Z. (2016,). *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network*. https://arxiv.org/abs/1609.05158

Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., & Wetzstein, G. (2020,). *Implicit Neural Representations with Periodic Activation Functions*. https://arxiv.org/abs/2006.09661

Smith, J. O. (2025). *Spectral Audio Signal Processing*. https://ccrma.stanford.edu/~jos/sasp/Product_Two_Gaussian_PDFs.html

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929-1958. http://jmlr.org/papers/v15/srivastava14a.html

Susskind, M. R. A. J. R. A. A. R. A. A. K. A. M. A. B. A. N. P. A. R. W. A. J. M. (2021,). Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding. *International Conference on Computer Vision (ICCV) 2021*.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., & Bregler, C. (2015,). *Efficient Object Localization Using Convolutional Networks*. https://arxiv.org/abs/1411.4280

Trockman, A., & Kolter, J. Z. (2022,). *Patches Are All You Need?*. https://arxiv.org/abs/2201.09792

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2017,). *Instance Normalization: The Missing Ingredient for Fast Stylization*. https://arxiv.org/abs/1607.08022

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023,). *Attention Is All You Need*. https://arxiv.org/abs/1706.03762

Wojna, Z., Ferrari, V., Guadarrama, S., Silberman, N., Chen, L.-C., Fathi, A., & Uijlings, J. (2019,). *The Devil is in the Decoder: Classification, Regression and GANs*. https://arxiv.org/abs/1707.05847

Woo, S., Park, J., Lee, J.-Y., & Kweon, I. S. (2018,). *CBAM: Convolutional Block Attention Module*. https://arxiv.org/abs/1807.06521

Yang, L., Kang, B., Huang, Z., Xu, X., Feng, J., & Zhao, H. (2024,). Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data. *CVPR*.

Zhang, J., He, T., Sra, S., & Jadbabaie, A. (2020,). *Why gradient clipping accelerates training: A theoretical justification for adaptivity*. https://arxiv.org/abs/1905.11881

Apéndice A
Script de inicio

Este *script* descarga los datos especificados en los ficheros `data/train.txt` y `data/test.txt`, elimina los archivos no necesarios y ejecuta la tarea de preprocesamiento.

```
#!/bin/bash

# Instalar dependencias en el entorno virtual
uv sync

# Descargar pesos de DepthPro
if [ -d "checkpoints" ]; then
    echo "El directorio checkpoints ya existe, omitiendo descarga de
pesos";
else
    mkdir -p checkpoints;
    wget https://ml-site.cdn-apple.com/models/depth-pro/depth_pro.pt -
P checkpoints;
fi

# Descargar datos
mkdir -p data/train data/test
cat data/train.txt | xargs -n 1 -P 8 wget -P data/train;
cat data/test.txt | xargs -n 1 -P 8 wget -P data/test;
unzip "data/train/*.zip" -d data/train;
unzip "data/test/*.zip" -d data/test;

# Eliminar archivos no necesarios
find data -type f ! -name "*.color.jpg" ! -name "*.est.npy" ! -name
"*.depth_meters.hdf5" ! -name "*.txt" -print0 | xargs -0 rm --

# Ejecutar tarea de preprocesamiento
uv run src/uncertainty_estimation/utils/preprocess.py
```

Apéndice B
Manual de uso

Instalación de dependencias y descarga de datos

```
bash ./init.sh
```

Preprocesado

```
uv run src/uncertainty_estimation/utils/preprocess.py
```

Búsqueda de hiperparámetros

```
uv run src/uncertainty_estimation/sweep.py
```

Entrenamiento

```
uv run src/uncertainty_estimation/train.py --config trainer_config.yaml
```

Evaluación

```
uv run src/uncertainty_estimation/utils/pipeline.py
```