Redes Neuronales Convolucionales Aplicadas a Demosaicing y Denoising

Gonzalo Balduvino, Matías Lorenzo

Supervisores: Dr Eduardo Fernández (InCo), Dr Mauricio Delbracio (IIE),
Dr José Lezama (IIE)

Informe de Proyecto de Grado presentado al
Tribunal Evaluador como requisito de
graduación de la carrera Ingeniería en Computación

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Instituto de Computación
Facultad de Ingeniería de UdelaR
Montevideo, Uruguay
Octubre, 2019

Resumen

Los procesos realizados por las cámaras digitales usualmente son propietarios y adaptados exclusivamente a los modelos de cada fabricante. Sin embargo, muchas de las etapas de la cadena de procesamiento son similares en la mayoría de las cámaras. Entre estas etapas se encuentran las de *demosaicing* y *denoising*, que se encargan de estimar colores faltantes no capturados por los sensores y eliminar el ruido de la imagen respectivamente.

Con el advenimiento del aprendizaje automático, y en particular del aprendizaje profundo, muchos trabajos se han presentado con el objetivo de implementar estas etapas de la cadena de procesamiento de las cámaras digitales. En este trabajo se presentan dos redes convolucionales para resolver el problema de demosaicing y denoising conjuntamente tomando en cuenta el estado del arte y buscando la optimización de estas redes. Además, junto con este trabajo se publica el código de un algoritmo que permite generar ejemplos para el entrenamiento de dichas redes.

Palabras claves: cámara digital, demosaicing, denoising, redes neuronales, redes convolucionales

Índice general

Capítulo 1

Introducción

1.1. Motivación

Desde su invención en el año 1975 la fotografía digital ha revolucionado la industria de la tecnología. En menos de cincuenta años se ha evolucionado de una cámara grande, pesada y con muy poca resolución a, entre otras cosas, dispositivos móviles capaces de tomar imágenes 1200 veces más detalladas que esta primer cámara.

Rápidamente los usuarios y los mercados adoptaron a la cámara digital como el nuevo protagonista de la fotografía. Este dispositivo no requería de película fotográfica. Las fotografías podían ser vistas al instante, y descartadas sin costo alguno en caso de no ser satisfactorias. Estos factores generaron un efecto disruptivo sobre el mercado, cuyos proveedores comenzaron a fabricar y a desarrollar nuevas tecnologías para ganar terreno. Tan abrupto fue el cambio que ya para el año 1997 la venta de cámaras digitales tendía a aumentar un 75 % por año, mientras que las ventas de cámaras análogas solamente aumentaba un 3 % [1].

La intensa competencia en el mercado por parte de los fabricantes generó una oferta de dispositivos que cumplían la misma misión, pero cuyo hardware y software era propietario, y por ende heterogéneo. El proceso realizado por una cámara para obtener la imagen final a partir de la señal capturada por el sensor es conocido como la cadena de procesamiento de imágenes (*Image Processing Pipeline* en inglés). Este proceso es complejo y consta de una cantidad variable de pasos según el fabricante. Si bien algunos de estos pasos están definidos, cada fabricante mantiene en secreto el exacto funcionamiento de su *pipeline* [2][3].

Estos factores han generado un ecosistema donde los pasos que una cámara debe seguir para lograr generar la imagen digital están definidos y son conocidos, pero cuya implementación varía entre fabricantes, y está adaptada al propio hardware. Dos de las etapas de este *pipeline* son las conocidas como *demosaicing* y *denoising*, y son, de alguna manera, el punto central del proceso. Si bien cada fabricante dispone de su implementación, las transformaciones realizadas en estas dos etapas se encuentran bien documentadas y definidas, y probablemente todas las cámaras digitales sigan un proceso muy similar.

Una de las ventajas de una cadena de procesamiento es que sus etapas son (o deberían ser) lo más independiente posible de las demás. Surge, entonces, la posibilidad de implementar estas etapas de forma genérica y agnóstica al hardware utilizado. Con el pasar de los años se han publicado trabajos de investigación que presentan implementaciones de las etapas de la cadena, en particular de *demosaicing* y *denoising*. Entre ellos se encuentran algunos trabajos que buscan implementar de forma conjunta estas dos tareas utilizando aprendizaje automático [4][5][6].

Con el advenimiento del aprendizaje automático, y en particular del aprendizaje profundo, muchas

de las tareas previamente realizadas por otro tipo de algoritmos (por ejemplo aquellos que utilizan heurísticas para llegar a un resultado aceptable en un tiempo razonable) comenzaron a ser implementadas utilizando modelos de este tipo. Como es de esperar, en el campo de la fotografía se ha experimentado con redes neuronales en varios tipos de aplicaciones. Por desgracia, en general este tipo de modelos, si bien son eficaces en las tareas para las que fueron entrenados, son poco eficientes en cuanto a recursos de cómputo y tiempo de ejecución, lo que hace complicada su aplicación en dispositivos de bajas prestaciones o móviles.

En el presente proyecto se analiza una posible implementación de una red convolucional que pretende disminuir el tiempo de ejecución y el poder computacional necesario para realizar las tareas de *demosaicing* y *denoising*, comprometiendo lo menos posible la precisión. Para lograr este objetivo, se explora la posibilidad de mejorar un modelo ya existente [5], alterando su arquitectura y analizando el costo en precisión de este cambio.

A lo largo de este informe se analizan los aspectos claves del diseño de la solución y las decisiones tomadas al respecto. Además, se detallan los principales problemas enfrentados. Uno de los principales, y que limitó considerablemente las posibilidades de exploración de las soluciones, fue la falta de poder computacional disponible para realizar el entrenamiento de las redes. Las etapas de entrenamiento de las redes consistían, en muchos casos, en un cuello de botella para la continuación del proyecto, dado que se requerían los resultados para seguir avanzando (que en algunos casos tomaban más de un día).

1.2. Alcance del proyecto

En este proyecto se aborda la construcción de la cadena de procesamiento digital de bajo nivel (*demosaicing* + *denoising*) utilizando técnicas de aprendizaje automático.

Dado que una cámara digital no adquiere directamente una imagen de color RGB, sino una imagen (mosaico de Bayer) que tiene un valor único de color en cada píxel (rojo, verde o azul), resulta imprescindible interpolar los colores faltantes en cada píxel de manera de completar la información faltante. A este proceso se le conoce como demosaicing.

Además es imprescindible realizar una etapa de eliminación de ruido (denoising), dado que por la propia naturaleza de la luz (ruido fotónico) y los componentes electrónicos presentes en el dispositivo digital, la imagen adquirida presenta ruido aleatorio.

En resumen, el objetivo del proyecto es desarrollar un sistema que procesa la imagen *raw* (mosaico de Bayer) capturada por una cámara digital para convertirla en una imagen de color RGB. Esto implica el desarrollo de algoritmos de demosaicing y denoising, aprendidos por redes convolucionales entrenadas especialmente para este trabajo, la evaluación y la publicación de los resultados.

1.3. Organización del documento

El presente documento se divide en cuatro capítulos. En el primero de ellos se detalla la introducción al proyecto, donde se explica su motivación. Luego, en el segundo capitulo se describe la historia y las partes que componen una cámara digital, el estado del arte y se detallan los avances realizados por otros investigadores en el área trabajada.

En el tercer capítulo se detalla la solución propuesta, definiendo el problema en términos formales y detallando las decisiones más importantes a la hora de diseñar y entrenar un modelo de aprendizaje profundo: el dataset utilizado, la función de pérdida elegida, la arquitectura de la red y el entrenamiento de la misma.

Por último se presentan los resultados experimentales, comparando los modelos entrenados con

el estado del arte referenciado en el segundo capítulo. Posteriormente se detallan las conclusiones y la bibliografía.

Capítulo 2

Fotografía digital

2.1. Historia de la fotografía digital

La fotografía se remonta hacia finales del siglo XVIII y a principios del siglo XIX. Los primeros prototipos inventados se basaron en el fenómeno de *camera obscura*, que ocurre al proyectar una escena por un pequeño agujero. Este fenómeno combinado con productos naturales que cambian de color con la influencia lumínica llevó a algunos inventores a proponer una forma innovadora de capturar un momento en papel.

Con el pasar de los años la fotografía se tornó en un fenómeno popular, y surgieron compañías enteras dedicadas a la fabricación y venta de dispositivos fotográficos. Entre estas se encontraba Eastman Kodak Co., fundada en 1888.

En el año 1975 los investigadores de Kodak Gareth A. Lloyd y Steven J. Sasson crean el primer prototipo de cámara digital en la historia [7]. La tecnología utilizada era extremadamente lenta y de baja calidad: la cámara tardaba 23 segundos en tomar una imagen, con una resolución de 0.01 mega píxeles. La información lumínica se almacenaba en una cinta magnética, que luego podía ser reproducida por un dispositivo especial. La figura 2.1 muestra el prototipo antes mencionado.

Lloyd y Sasson fueron capaces de crear su prototipo en base a la invención de los Dispositivos de Carga Acoplada (*Charge-Coupled Devices*, o *CCD*). Estos dispositivos se usan actualmente para tomar imágenes digitales, y son clave en el funcionamiento de una cámara digital.

Con el pasar de los años, la cámara digital se popularizó y surgieron varias empresas que fabricaron sus propias versiones. Si bien la arquitectura base de las cámaras digitales es la misma, la diferencia en detalles de implementación generó un ecosistema relativamente heterogéneo.

La cámara digital también causó el nacimiento de aristas de la ciencia de la computación enfocadas específicamente al tratamiento de imágenes digitales, tanto de alto nivel como de bajo nivel. El tratamiento de imágenes de alto nivel ha gozado de popularidad en los últimos años, con grandes avances en las áreas de imagenología médica y visión artificial. Sin embargo, a lo largo de la historia se han presentado avances muy importantes en el tratamiento de imágenes de bajo nivel, como lo son las técnicas modernas de demosaicing y denoising. Este informe se enfoca principalmente en el tratamiento de imágenes de bajo nivel, atacando las tareas mencionadas.

Figura 2.1: Imagen de la primer cámara digital

2.2. Funcionamiento de las cámaras digitales

Las cámaras digitales de hoy en día funcionan en base a un fuerte componente de hardware, encargado de captar la señal lumínica y de traducirla a un formato digital. Luego, la imagen digitalizada pasa por un proceso serial de transformaciones que permiten obtener las imágenes a las que estamos acostumbrados los humanos.

El componente de la cámara que se encarga de realizar estas tareas es conocido como Procesador de la Señal de Imagen (o *Image Signal Processor* en inglés). A lo largo de los años los fabricantes de cámaras se han puesto de acuerdo en las etapas que debe tener el procesamiento de imágenes, pero en general la implementación es propietaria.

El proceso comienza en el momento que la señal lumínica arriba al sensor de luz. Estos sensores consisten la primer parte del pipeline digital de la cámara fotográfica.

2.3. Sensores Lumínicos

A lo largo de la historia, dos tipos de sensores han predominado en las cámaras digitales para captar la intensidad lumínica: los sensores basados en un CCD y los sensores CMOS.

2.3.1. Captura de la señal

Esencialmente la captura de la señal lumínica es realizable gracias al efecto fotoeléctrico. Este fenómeno se define como la liberación de electrones de un material causado por la influencia de fotones sobre el mismo [8].

La emisión de electrones es un evento aleatorio descrito por la *eficiencia cuántica* del material. Esta característica mide la sensibilidad lumínica, y se define como el valor esperado del cociente entre el número de fotones que impactan el material y el número de electrones que se liberan. Como detalle, un buen sensor CCD tiene una eficiencia cuántica de más del 90 %, mientras que la eficiencia del aparato visual humano es de apenas 15 %.

Otro factor influyente en la toma de imágenes digitales es el tamaño del sensel. Sensels con más

Figura 2.2: Efecto de *blooming*. Los dos focos de luz emiten una señal suficientemente intensa como para saturar los sensels.

superficie serán capaces de capturar más fotones, y por ende proveer una mejor medición. La profundidad del sensel es la que determina su capacidad. Es decir, los sensels solo pueden almacenar hasta cierta cantidad de carga, antes de ser saturados. La saturación de un sensel es conocida como *blooming*, y en los sensores utilizados actualmente es un fenómeno relativamente común (por ejemplo cuando se toma una imagen de una fuente lumínica fuerte como el Sol). La saturación de un sensel genera alteraciones en los sensels cercanos, dado que la carga en exceso es capturada por estos. La figura 2.2 muestra un caso de *blooming*.

La saturación de los sensores influye mucho en el tratamiento de imágenes de bajo nivel, dado que convierte un proceso potencialmente lineal en uno no lineal. La saturación también es conocida como *clipping*. Este concepto será explorado en mayor profundidad en la sección 2.5.

2.3.2. Dispositivos de Carga Acoplada

Como fue dicho anteriormente, el primer prototipo de cámara digital diseñado por Sasson se basó en un invento del momento: los sensores basados en dispositivos de carga acoplada (en adelante *CCD*). Este tipo de sensores fueron revolucionarios, siendo de principal utilidad en el área de la fotografía digital.

La tarea de un CCD es, en esencia, transmitir carga eléctrica en una dirección, potencialmente hacia un circuito que sea capaz de procesarla. Son construidos en base a una matriz de celdas que almacenan carga. Una vez terminada la carga del dispositivo, las diferentes cargas son movidas a través de la matriz hacia una salida única. La figura 2.3 muestra la estructura y funcionamiento de un CCD.

La innovación introducida por sensores basados en estos dispositivos fue la facilidad de transmisión de carga. Esto es especialmente útil en la fotografía digital, dado que los sensores lumínicos se disponen en una matriz.

Los sensores lumínicos basados en CCDs se construyen implementando una matriz de sensores lumínicos, cuya carga se almacena en una celda del CCD. Luego, toda la información es digitalizada fuera del CCD. Es decir, la única función del CCD es transmitir la carga almacenada por los sensores lumínicos al adaptador que convierte la carga en una señal digital.

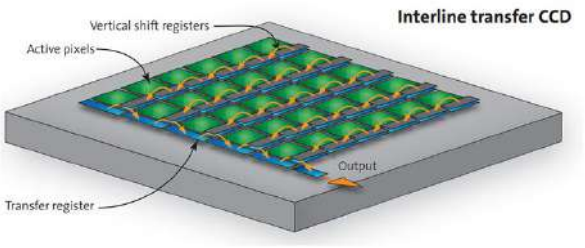


Figura 2.3: Dispositivo de carga acoplada. Pueden observarse los diferentes movimientos de carga hasta la salida. Imagen obtenida de https://www.stemmer-imaging.co.uk/en/knowledge-base/ccd

2.3.3. Sensores CMOS

CMOS, o Semiconductor Complementario de Óxido de Metal, es una tecnología de construcción de circuitos integrados utilizada ampliamente en el área del hardware informático.

Los sensores CMOS son la elección actual más común para implementar sensores de cámaras digitales. Al igual que ocurre con los sensores CCD, los sensores CMOS son llamados así porque se utiliza esta tecnología para construir el sensor.

Las ventajas principales aportadas por los sensores CMOS con respecto a los sensores CCD incluyen el bajo costo de producción, y la mayor resistencia a ciertos fenómenos eléctricos, como el *blooming*. Además, los sensores CMOS integran los sensores lumínicos con los adaptadores de conversión análoga-digital. Como desventaja, los sensores CMOS son más sensibles al efecto de barrido dado que la señal es procesada por filas.

2.3.4. Sensel

Se conoce como *sensel* a una celda de la matriz de sensores, y también a su output transformado a una señal digital. La diferencia fundamental entre el concepto de píxel y de sensel es que el sensel se corresponde con el valor digitalizado de la carga eléctrica obtenida por un sensor, mientras que un píxel es un elemento virtual que describe un área puntual de la imagen. Los píxeles pueden ser de uno o más canales (un canal en caso de una imagen monocromática, tres canales en una imagen a color, o hasta cuatro en una imagen a color con transparencias). Si bien la expresión *sensel* no es tan común en la literatura, en este informe se utilizará con el objetivo de evitar confusiones con la palabra *píxel*. En particular, se utilizará *sensel* para describir valores de la imagen previo a la realización de demosaicing, y *píxel* para elementos de la imagen luego de esa etapa del pipeline. El concepto de *demosaicing* es explicado más adelante.

2.4. Color en imágenes

2.4.1. Representación de imágenes

La mayoría de los dispositivos capaces de emitir luz para representar imágenes lo hacen a través de píxeles que emiten tres colores. En generar estos colores son rojo, verde y azul, aunque puede haber implementaciones con otro tipo de colores, e incluso más canales.

El estándar más utilizado actualmente es el llamado *sRGB* (*Standard RGB*), que determina que los píxeles de las imágenes deben estar codificados con tres canales: uno rojo, uno verde y uno azul, en general en ese orden. Si bien también existe el estándar *YCbCr*, que modela los píxeles en tres canales: uno asignado a la luminancia de la imagen, y los otros dos al croma azul y rojo respectivamente, es más común encontrar imágenes codificadas en *sRGB*. De cualquier manera existe una transformación biyectiva entre ambos modelos de color.

De forma análoga, los dispositivos que se encargan de capturar imágenes a color deben hacerlo determinando estos tres canales de información.

2.4.2. Captura de color

Los sensores vistos anteriormente son capaces de capturar la intensidad de señal lumínica, pero no son capaces (por sí solos) de distinguir los tres canales requeridos y obtener tres valores a partir de la señal inicial. Esto se basa en que los fotones son indistinguibles, sin importar el color de la señal que representan.

Una solución intuitiva al problema es la utilización de filtros para obtener los diferentes colores necesarios. En el caso de *sRGB* son necesarios filtros azules, rojos y verdes. Inmediatamente se presenta un problema: se necesitan tres sensores para poder capturar los tres colores por separado.

Si bien esta solución es ideal, presenta varios problemas de implementación. Entre ellos, el costo de triplicar la cantidad de sensores necesarios, y el problema de diseño del dispositivo: los tres sensores deben ser capaces de captar la misma escena al mismo tiempo, por lo que la luz debe ser desviada con estrategias especulares hacia los tres sensores en simultáneo. Estas cámaras son fabricadas actualmente, en general enfocadas a profesionales de la fotografía. Sin embargo, este tipo de implementaciones no es ideal para dispositivos pequeños como teléfonos celulares.

En base a esta necesidad es que surgen los arreglos de filtros y las técnicas de demosaicing.

2.4.3. Arreglos de filtros

Los arreglos de filtros son ordenamientos de filtros de color dispuestos encima de un sensor de una cámara digital. El objetivo de estos arreglos es filtrar la luz de forma de que a cada sensel le llegue un color particular. De esta forma es posible fabricar cámaras digitales con un único sensor.

El problema que presenta esta estrategia es que la información obtenida del sensor está incompleta. Dos tercios de los valores necesarios son desconocidos al momento de leer la salida del sensor. Para solucionar este problema se han propuesto a lo largo de los años diversas estrategias de interpolación para obtener los colores faltantes. El proceso de interpolar los valores faltantes se conoce como *demosaicing* o *demosaicking* (utilizados intercambiablemente en la literatura).

Uno de los arreglos de filtros más utilizados en los sensores actualmente es el arreglo de Bayer (también llamado patrón de Bayer o filtro de Bayer). Patentado por Bryce Bayer [9], el patrón de Bayer usualmente se modela como una matriz de 2×2. La figura 2.4 presenta una posible implementación del patrón de Bayer. En particular, es la implementación más usual actualmente. La focalización en filtros de color verde fue pensada de esa manera dado que el ojo humano es mucho más sensible a las frecuencias relacionadas con el verde que a las relacionadas con el azul o rojo.

La señal obtenida de un sensor que implementa algún arreglo de filtros es 'incompleta' en el sentido de que no es posible determinar el valor de todos los canales de todos los píxeles que tendrá la imagen final a partir de la información en sí. En particular, se conoce únicamente un canal de cada píxel.

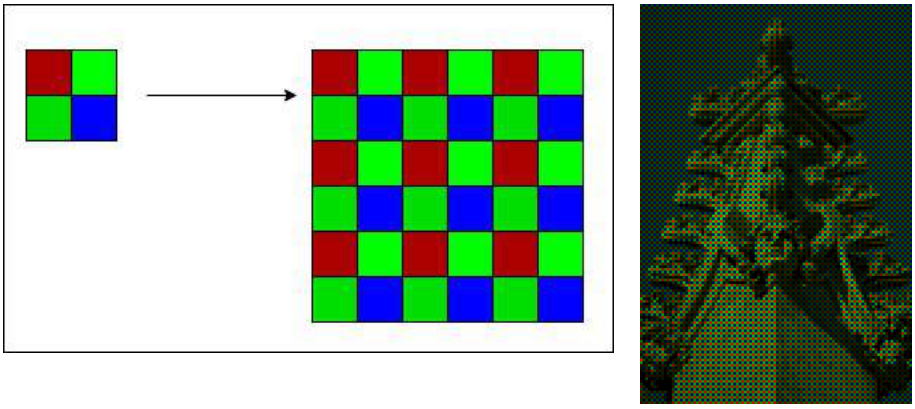


Figura 2.4: Ejemplo de la disposición del patrón de Bayer en un sensor. A la izquierda el arreglo de filtros y la replicación en un sensor. A la derecha una imagen obtenida sin realizar demosaicing, tomada de [4].

Una vez obtenida la señal digital del sensor, los colores faltantes deben ser interpolados de alguna manera. Aquí entra en juego el *demosaicing*.

2.4.4. Demosaicing

El proceso de demosaicing consiste en interpolar los valores de los canales desconocidos de los píxeles de una imagen. La figura 2.4 muestra una imagen tomada sin haber aplicado demosaicing. Si bien la imagen tiene una resolución relativamente baja, es muy notable la falta de información.

A lo largo de la historia de las cámaras digitales se han propuesto y desarrollado varias estrategias de demosaicing. Hoy en día cada fabricante implementa la solución que mejor funciona con el hardware del que disponen.

Si bien la tarea de demosaicing no se aplica únicamente a las imágenes tomadas con el patrón de Bayer, en este informe se utilizará el término para referirse específicamente a su implementación sobre una fotografía tomada utilizando este patrón, a menos que se especifique lo contrario.

Realizar demosaicing de la forma más eficiente y eficaz posible es vital en las cámaras digitales de la actualidad. Las cámaras modernas son capaces de mostrar en tiempo real cómo se va a ver la fotografía luego de tomarla, y esto implica, entre otras cosas, la necesidad de ejecutar un algoritmo de demosaicing en tiempo real.

Actualmente lo más común entre los fabricantes es utilizar una estrategia de interpolación. Una de las más simples que se pueden utilizar es la interpolación bilineal. Con el advenimiento del aprendizaje automático y el aumento de la capacidad de procesamiento en tiempo real de los dispositivos, también se ha incursionado en técnicas de demosaicing utilizando aprendizaje automático.

2.5. Ruido en Imágenes

Los sensores utilizados para captar la señal lumínica son susceptibles a ruido. Además de la tarea de demosaicing, es muy importante intentar eliminar el ruido de los sensores.

Ambos tipos de sensores descritos anteriormente sufren de dos tipos diferentes de ruido.

2.5.1. Ruido Fotónico

El ruido fotónico es causado por la variación aleatoria del arribo de los fotones al sensor. La lluvia de fotones se comporta siguiendo una distribución Poisson, por lo que es razonable asumir que el ruido fotónico siga también una distribución de este tipo.

Con el avance de la tecnología a través de los años, los dispositivos se han vuelto cada vez más pequeños, entre ellos los sensores lumínicos. Cada sensel dispone de una superficie muy acotada para capturar fotones. Dado que el ruido fotónico está directamente relacionado con la cantidad de fotones captados, que los sensores sean más pequeños los hace más susceptibles al ruido fotónico.

2.5.2. Ruido de Lectura

El ruido de lectura ocurre en la etapa de traducción de la señal análoga a digital. Tanto la iluminación como la temperatura del sensor generan movimientos de cargas eléctricas que alteran la lectura. Además, los demás circuitos cercanos acoplados al sensor pueden influir con su propia carga eléctrica. Las ineficiencias de este tipo en general siguen una distribución normal.

2.5.3. Modelo de ruido en imágenes

En este proyecto se hace utiliza el modelo de Foi et al. [10]. Los investigadores proponen un modelo basado en los dos tipos de ruido anteriormente presentados. El ruido puede ser modelado de la siguiente manera:

$$z(x) = y(x) + \sigma(y(x))\xi(x)$$

Donde z corresponde con la señal leída, y es la señal sin ningún tipo de ruido o interferencia, y $\sigma(y(x))\xi(x)$ es el componente de ruido asociado a la señal. El valor de x determina un píxel de y.

Los investigadores argumentan que el componente de ruido se compone de dos partes independientes entre sí: el componente de Poisson y gaussiano. De esta forma, el componente de ruido puede ser reescrito como:

$$\sigma(y(x))\xi(x) = \eta_p(y(x)) + \eta_g(x)$$

Donde η_p es el componente de ruido de Poisson, y η_g es el componente de ruido gaussiano. Importa notar que, inicialmente, los investigadores conjeturan que el componente de ruido de las imágenes se comporta de la manera antes definida. Luego, en sus experimentos, muestran empíricamente que este modelo es estadísticamente confiable.

Notar que el componente de Poisson depende del valor de la señal real, mientras que el componente gaussiano es independiente. Esto se debe a que el ruido fotónico está directamente influenciado por la intensidad de la señal. Las escenas con muy poca iluminación son mucho más propensas a ruido fotónico que las bien iluminadas, por ejemplo.

Poder detectar el ruido de una imagen y suprimirlo es muy importante. En la sección 4.5 se expandirá con respecto al trabajo de Foi et al. , donde presentan un algoritmo de estimación de la distribución de ruido de una imagen dada.

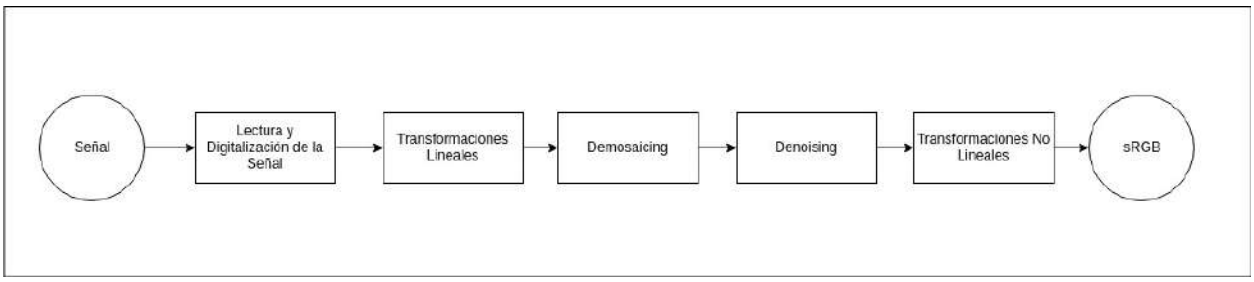


Figura 2.5: Diferentes etapas del pipeline digital, partiendo desde la lectura e interpretación de la señal, y llegando a la imagen final.

2.5.4. Denoising

Se conoce como *denoising* a la tarea de intentar estimar y remover el ruido de una imagen. Esta tarea, al igual que el demosaicing, representa una parte muy importante del procesamiento que debe realizar una cámara digital.

A lo largo de la historia, el denoising ha sido implementado utilizando heurísticas rápidas, aunque también se han publicado trabajos académicos en los que se incursiona en el área del aprendizaje automático.

2.6. Pipeline Digital

Las cámaras digitales deben realizar diversas transformaciones a la imagen obtenida de los sensores lumínicos. Estas transformaciones pueden ser tanto lineales como no lineales, y cada fabricante implementa su versión de cada una. La ejecución de estas transformaciones en serie, partiendo de la captura de la señal lumínica, hasta la obtención de la imagen virtual final es conocida como pipeline digital, o bien *ISP* (*Image Signal Processor* en inglés).

Si bien cada fabricante realiza su propia implementación (en muchos casos propietaria y secreta), se presentará a continuación una versión genérica del pipeline digital de una cámara [4][11], que incluye las tareas necesarias en un orden razonable. La figura 2.5 muestra las diferentes etapas del pipeline digital.

2.6.1. Transformaciones lineales

Luego de digitalizar la señal es necesario realizar algunas correcciones lineales. Esta necesidad surge de varios fenómenos que ocurren al tomar la fotografía.

En primer lugar, los sensores lumínicos son sensibles al calor. Temperaturas relativamente altas generan movimiento de cargas dentro del sensor, que pueden influenciar las mediciones de los valores de los sensels.

Para mitigar esta situación, los sensores usualmente disponen de un borde de sensels cubiertos. La transformación llamada *corrección de oscuros* se encarga de sustraer el valor de los sensels apagados de los sensels no apagados. Esta operación también es conocida como la remoción del sesgo del sensor.

Otra transformación usual es la corrección del nivel de blancos. Al tomar una imagen, los colores obtenidos pueden no respetar las condiciones de la visión humana. Por esto, muchas veces es necesario realizar una corrección de los valores de color de la imagen. Para lograr esto, las cámaras usualmente almacenan un coeficiente de corrección de color (uno diferente para cada canal de la imagen), y realizan una ponderación

de todos los valores de cada canal por su correspondiente coeficiente de corrección.

Un fenómeno común es el *viñeteado* (o *vignetting* en inglés). Este efecto ocurre en general cerca de los bordes de las imágenes y es causado por la obstrucción del lente de la cámara. Los rayos de luz que provienen de los bordes de la escena se topan con obstáculos en el lente y por ende se capta una señal menos intensa. El viñeteado se combate realizando una amplificación del valor obtenido por los sensels cercanos a los bordes.

Naturalmente, los fabricantes son libres de implementar estas tareas como lo vean necesario. Incluso las tareas antes mencionadas no son estrictamente necesarias en un pipeline digital. La corrección de oscuros, por ejemplo, es obviada por algunos fabricantes.

Sin embargo, los pipelines digitales usualmente reservan estas primeras etapas para transformaciones lineales. Esto tiene la ventaja de hacer más fácil el demosaicing y denoising, dado que las propiedades básicas de la señal se mantienen.

2.6.2. Transformaciones no lineales

Luego de realizar demosaicing y denoising, el resultado es una imagen de varios canales cuyos valores son totalmente conocidos. En esta etapa usualmente las cámaras intentan transformar las imágenes de manera de que se vean más naturales para un ser humano.

Esta tarea no es trivial, e involucra varios tipos de transformaciones no lineales sobre las imágenes.

En algunos casos, por ejemplo, es necesario realizar un cambio de base de color, conocido también como conversión del espacio de color. Las cámaras emiten imágenes en diversos modelos de color (*sRGB* es uno de los más usuales, aunque también existe *AdobeRGB*). Las transformaciones de conversión de espacio de colores son no lineales en algunos casos. Además, generan dependencias entre canales. Es decir, el valor de un píxel de un canal dado depende de los valores de píxeles previos de otros canales. Este tipo de dependencias rompen las características específicas de cada canal de la imagen original.

El ojo humano está adaptado a percibir intensidades lumínicas de forma no lineal. Los sensores lumínicos, por otro lado, en general responden linealmente a los cambios de intensidad lumínica. Por esta razón es necesario realizar un cambio en la distribución de intensidades de la imagen. Esta transformación no lineal es conocida usualmente como *corrección gamma*. Dado el valor de un canal de un píxel (denotado por x), su corrección gamma está dada por un parámetro arbitrario γ. La fórmula para obtener la corrección de un píxel es la siguiente:

$$\Gamma(x) = 255 \times \left(\frac{x}{255}\right)^{1/\gamma}$$

Usualmente el valor de γ se determina en base al espacio de color codificado por la cámara. Uno de estos espacios de colores es el previamente mencionado *sRGB*, para el cuál el valor de γ más común es $\frac{1}{2{,}2}$. Para el caso de las imágenes *raw*, el valor de γ es usualmente 1 para lograr una función gamma lineal.

2.7. Malvar

Uno de los algoritmos más simples para implementar demosaicing es la interpolación bilineal. Este método consiste en que, dada una imagen obtenida de un CCD con el patrón de Bayer, se calcula la información de los canales faltantes de cada pixel en base a sus vecinos más próximos.

La interpolación bilineal del canal verde en un pixel correspondiente al canal rojo o azul en el patrón de Bayer, se calcula utilizando la siguiente ecuación:

$$\hat{G}^{bl}(i,j) = \frac{1}{4}(G(i-1,j) + G(i+1,j) + G(i,j-1) + G(i,j+1)$$

Donde G se asocia al canal verde. Para interpolar los valores de los canales rojo y azul se utilizan los vecinos diagonales (excepto en el caso de un pixel verde del mosaico, para el cual se usan solo dos). A continuación un ejemplo de la interpolación del canal azul en un pixel rojo:

$$\hat{B}^{bl}(i,j) = \frac{1}{4}(B(i-1,j-1) + B(i+1,j-1) + B(i-1,j+1) + B(i+1,j+1)$$

Una de las principales ventajas de la interpolación bilineal es que es extremadamente fácil de implementar, y es muy eficiente su ejecución si se modela como convoluciones con un filtro diferente para cada caso.

El problema de esta estrategia de interpolación es que para determinar el valor de un canal utiliza únicamente información circundante asociada a ese canal. Esto genera artefactos no deseables, principalmente en áreas con componentes de alta frecuencia, como son bordes y fronteras de objetos. En estos casos se puede ver discordancias de color.

Por esta razón, Malvar, He y Cutler [12] proponen un algoritmo de interpolación (conocido como interpolación de Malvar) que mejora la interpolación bilineal agregando nuevos términos. Uno de los principales aportes es que la interpolación de un canal se basa en información circundante de todos los canales, evitando así las desventajas de la interpolación bilineal.

El algoritmo, al igual que con la interpolación bilineal, es fácilmente implementable con convoluciones con filtros diferentes para cada caso. La figura 2.6 muestra los ocho posibles filtros que utiliza el algoritmo, junto con la situación en la que se utiliza.

Para la realización de este proyecto se utiliza la interpolación de Malvar como una de las bases de comparación para el desarrollo de la red. En particular, se utiliza la implementación presentada por Pascal Getreuer [13].

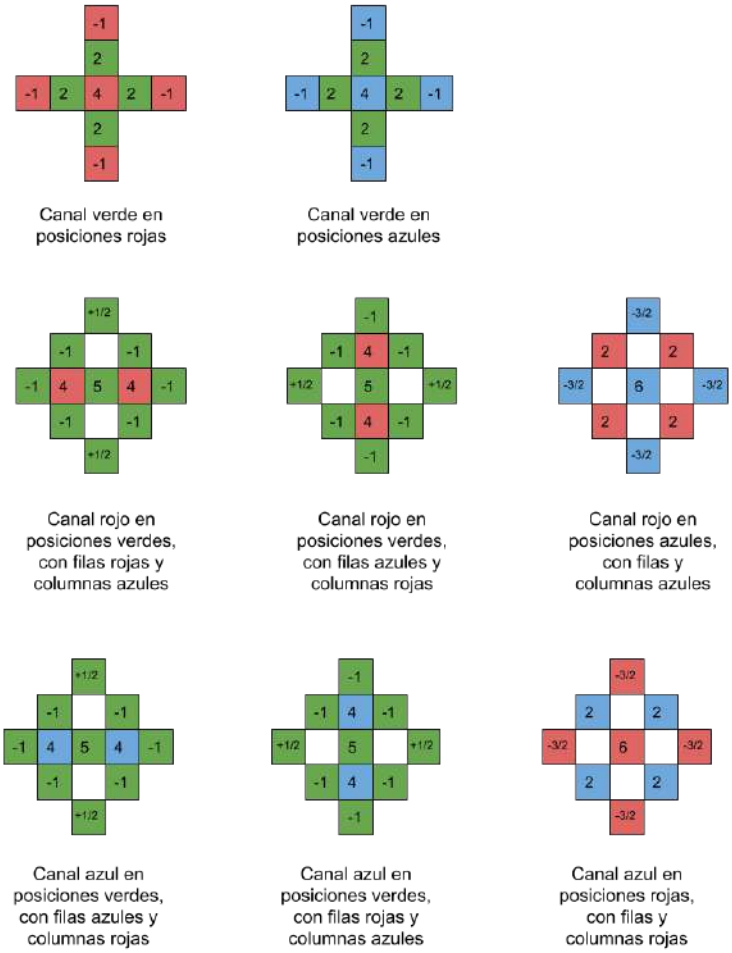


Figura 2.6: Los ocho tipos de filtros que utiliza el algoritmo de Malvar para interpolar. Imagen tomada y adaptada de [12].

Capítulo 3

Redes Neuronales

Las redes neuronales son modelos matemáticos utilizados para aprender una cierta tarea. Estas tareas son usualmente extremadamente complejas de modelar con exactitud, por lo que se recurre a modelos de aprendizaje automático (como son las redes neuronales) para intentar aprenderlas.

Usualmente una red neuronal se compone de unidades interconectadas que procesan e intercambian información. En esta sección se presenta una breve historia de las redes neuronales y los conceptos fundamentales asociados a ellas.

3.1. Perceptrón

En 1958 Rosenblatt propone al perceptrón como unidad de aprendizaje [14], cuya misión es representar matemáticamente a una neurona. Un perceptrón recibe una cantidad arbitraria de entradas, les aplica una transformación lineal y luego una función de activación. El resultado de esa función de activación es la salida emitida por el perceptrón. Estos coeficientes son generalmente conocidos como los pesos de la unidad. En el caso del perceptrón, la función de activación utilizada es la función umbral, expresada de la siguiente forma:

$$f(x) = \begin{cases} 0 & \sum_1^m x_i w_i + b < z \\ 1 & \text{en otro caso} \end{cases}$$

Donde m es la cantidad de entradas al perceptrón, x_i es el valor de la entrada i y b es un valor escalar variable conocido como *sesgo*. w_i corresponde al coeficiente i de la transformación lineal antes mencionada.

Una de las arquitecturas más básicas de red neuronal consiste en una única capa de perceptrones, mostrada en la figura 3.1. Las unidades de entrada y salida son simplemente una forma de representar visualmente la cantidad de entradas que espera la red, y la cantidad de salidas que emite. Este tipo de red es conocida como red pre alimentada. Es decir, las salidas de cada capa alimentan la siguiente capa, por lo que la información fluye en un único sentido.

Si bien las redes neuronales eran conceptualmente poderosas, varios factores limitaban el uso que se les podía dar. En particular, el poder computacional necesario para entrenarlas era alto para el hardware disponible en la época. Además, no existía un algoritmo de entrenamiento efectivo.

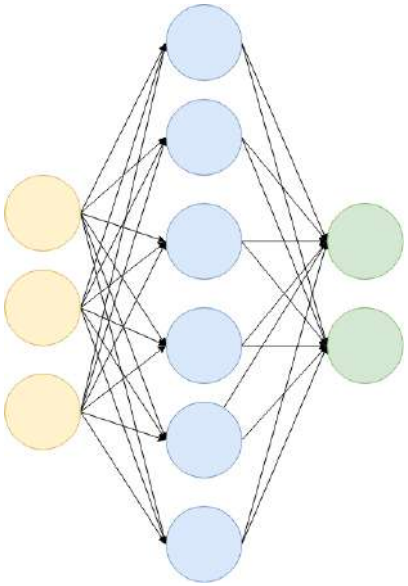


Figura 3.1: Una arquitectura básica de red neuronal. En amarillo las unidades de entrada, en azul los perceptrones (capa oculta) y en verde las unidades de salida.

3.2. Propagación hacia atrás

Esto último empezaría a cambiar en el año 1970 con la presentación por parte de Linnainmaa del método de diferenciación automática [15] (AD por sus siglas en ingles). A partir de este método, Werbos en el año 1974 menciona la posibilidad de utilizarlo en redes neuronales [16], y en 1982 publicaría su aplicación a funciones no lineales[17], lo que luego seria conocido como el método de propagación hacia atrás [18](*backpropagation* en inglés). Este algoritmo es vital para el entrenamiento de redes neuronales profundas (con más de una capa de neuronas interna), y consiste en propagar el gradiente de la función de costo hacia atrás, actualizando los pesos de cada neurona.

La propagación hacia atrás permite ayudar a corregir los pesos actuales de la red, minimizando el error que comete en un conjunto de ejemplos de entrenamiento. Para lograr esto es necesario conocer el gradiente de la función de activación de cada unidad de cada capa. El algoritmo calcula iterativamente para cada capa estos gradientes, tomando como entrada los gradientes calculados para la capa inmediatamente siguiente (la que es alimentada por la actual). ¿Cómo se calcula entonces el gradiente de la última capa? Primero se debe realizar una "pasada hacia adelante"(*forward pass*).

Un ejemplo de entrenamiento consiste de un par $(x, y), x \in \mathbb{R}^n, y \in \mathbb{R}^m$, donde x corresponde al valor de entrada y y corresponde al valor de salida esperado de la red. Una pasada hacia adelante consiste en computar el resultado de la red para x, llamado $\hat{y}$. Si $|\hat{y} - y| > \overrightarrow{0}$ significa que la red no modela exactamente el comportamiento dictado por el ejemplo de entrenamiento (x, y). Aquí es donde el algoritmo de propagación hacia atrás es útil, ya que se debe realizar alguna corrección en los pesos de la red para mejorar el desempeño de la red en ese ejemplo.

Primero que nada es necesario poder cuantificar qué tanto error está cometiendo la red. Esto se hace definiendo una función de pérdida, que es totalmente arbitraria y usualmente elegida en base a, entre otras cosas, la tarea que se desea aprender. Esta función de pérdida debe ser diferenciable para que el algoritmo de propagación hacia atrás pueda ser realizado. Volviendo a la consigna anterior, el gradiente inicial calculado por el algoritmo es el de la función de pérdida con respecto a $\hat{y}$.

Con este gradiente inicial es posible calcular los gradientes de cada unidad de cada capa. La regla de

la cadena para diferenciar permite "propagar.el cálculo del gradiente hacia capas anteriores. Los gradientes obtenidos pueden ser interpretados como una cuantificación del error cometido por cada capa. Con esta información es posible corregir los pesos de cada capa para acercarse más a la representación exacta de la función que se desea aprender. El entrenamiento de una red en el contexto del algoritmo de propagación hacia atrás es, justamente, el proceso de calcular los gradientes de cada capa utilizando como entrada un conjunto de pares de entrenamiento y actualizar debidamente los pesos de la red para disminuir el error cometido por la red.

Esta es una versión extremadamente simplificada del proceso de entrenamiento por propagación hacia atrás. Existen muchísimas formas de complementar el entrenamiento. Por ejemplo ajustando el impacto de los gradientes a la hora de actualizar los pesos (esto es usualmente conocido como la tasa de aprendizaje).

Como dato de interés, en el año 1989 Cybenko publica un trabajo en el que se demuestra formalmente que una única capa de perceptrones con arbitraria cantidad de neuronas y la función sigmoide como activación es capaz de representar cualquier función [19]. Si bien el trabajo no indica un algoritmo o método de aprendizaje, es un resultado prometedor para el aprendizaje por redes neuronales.

3.3. Funciones de activación

Las neuronas de una red usualmente utilizan una función de activación. Este tipo de funciones agrandan el espacio de soluciones al incluir soluciones no lineales. Recordemos que los nodos de la red realizan una combinación lineal (ponderada por los pesos aprendidos) de las entradas. Sin una función de activación una red neuronal sería completamente lineal, por lo que la capacidad de aprendizaje se vería seriamente comprometida [20].

En base a esto último, la elección en la función de activación no siempre es trivial, y existen varias opciones, algunas más populares que otras. A continuación se detallan las más relevantes.

3.3.1. Función umbral

Una de las funciones de activación más simples e intuitivas es la función umbral. Esta función fue la utilizada inicialmente con los perceptrones, y no tiene ningún componente no lineal. También llamada "salto binario", la función umbral emite un valor binario en base a la entrada, y su definición es la siguiente:

$$U(x) = \begin{cases} 0 & x < z \\ 1 & x \geq z \end{cases}$$

El valor de z es usualmente fijado antes del entrenamiento. El problema principal de esta función es la falta de expresividad: cada nodo puede emitir, como mucho, dos valores diferentes en todo momento. Además, la falta de un componente no lineal no permite representar funciones no lineales de forma simple cuando los nodos se agrupan en una red.

3.3.2. Función Sigmoide

Una de las funciones de activación más conocidas cuando se trata de redes neuronales, la función Sigmoide [20] (también se utiliza el término "Logística", dado que la función logística es una de las funciones sigmoides más utilizadas) introduce un grado de no linealidad a las neuronas de la red. Una función sigmoide es monótona creciente y que presenta asíntotas en ambos extremos en un valor finito.

Dos de las funciones sigmoides exploradas al utilizar redes neuronales son la función logística estándar y la tangente hiperbólica, definidas a continuación:

$$f(x) = \frac{1}{1 + e^{-z}}$$
$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Donde $f(x)$ representa la función logística estándar. Este tipo de funciones tienen algunas ventajas destacables:

- *No linearidad:* el componente no lineal introducido por ellas permite a las redes neuronales aprender funciones no lineales.
- *Normalización de la salida:* dado que estas funciones son asintóticas, sus salidas siempre se encuentran dentro de un rango fijo. Esto, por un lado, facilita el entrenamiento dado que se evita que las salidas de las redes crezcan (o decrezcan) infinitamente. Sin embargo también introduce una desventaja, explicada a continuación.

De cualquier manera, las funciones de tipo sigmoide presentan desventajas. Una de las más notables es el potencial de disipar los gradientes de la red. Esto ocurre usualmente en los extremos de la función. Dado que el gradiente de las sigmoides tiende a 0 en los extremos, la capacidad de mejora durante el entrenamiento de una red decrece si las neuronas emiten valores cercanos a los extremos. Una de las recomendaciones para evitar este caso es implementar una inicialización de pesos inteligente, cercana al eje de simetría de la función.

3.3.3. Unidad Lineal Rectificada (*ReLU*)

La eficiencia a la hora de entrenar una neurona influye mucho en el tiempo total de entrenamiento de la red. Las funciones sigmoides, si bien son capaces de dar gran expresividad a la red, son difíciles de calcular de forma rápida.

Los rectificadores son funciones casi lineales, rápidas de computar, y con un poder de expresividad muy similar al de las funciones sigmoides. En efecto, los rectificadores son las funciones de activación más populares en la actualidad, particularmente en el área de aprendizaje profundo. Una de ellas, la Unidad Lineal Rectificada [21] (*Rectified Linear Unit* o *ReLU* en inglés) ha probado ser una de las más eficientes.

Extremadamente simple de definir y computar, la función ReLU se expresa de la siguiente forma:

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & \text{en otro caso} \end{cases}$$

Donde z es un valor arbitrario. Variantes de *ReLU* permiten que la red modifique z en tiempo de entrenamiento

3.4. Inicialización de pesos

Un problema al que se deben enfrentar las redes neuronales en el procesamiento de la red es el de evitar que los resultados de las salidas de las capas de activación se vuelvan demasiado grandes o demasiado

pequeños, ya que si esto sucede impactaría directamente en el método de propagación hacia atrás y dificultaría la convergencia de la red.

Para solucionar este problema se han propuesto diferentes tipos de inicializaciones de los pesos de la red [22][23][24][25]. Entre las distintas inicializaciones, las mas conocidas son la de Xavier [23] y la de Kaiming[22].

La inicialización de Xavier consiste en inicializar los pesos de cada neurona utilizando una distribución de probabilidad con varianza $\frac{2}{n_{in}+n_{out}}$ siendo n_{in} la cantidad de entradas de la neurona en cuestión y n_{out} la cantidad de neuronas a las que alimenta con su salida. Esta estrategia de inicialización funciona mejor para las activaciones sigmoide, mientras que Kaiming utiliza una distribución con varianza $\frac{2}{n_{in}}$ y funciona mejor para los rectificadores. Tanto Xavier como Kaiming son mejoras a la inicialización propuesta por LeCun[20].

3.5. Aprendizaje profundo y redes neuronales convolucionales

El aprendizaje profundo es una sub-familia de algoritmos pertenecientes al aprendizaje automático, en la que se encuentran las redes neuronales profundas: redes con más de una capa oculta. Uno de los avances propuestos en aprendizaje profundo fueron las redes neuronales convolucionales [26].

La diferencia fundamental entre las redes neuronales completamente conectadas y las convolucionales es que las neuronas de las capas completamente conectadas reciben las salidas de todas las neuronas de la capa anterior, mientras que las neuronas de las capas convolucionales observan un subconjunto de las salidas de las neuronas de la capa anterior. Este nuevo tipo de redes neuronales surge como una manera de aplicar el aprendizaje profundo al tratamiento de imágenes y la visión por computadora.

Una capa convolucional básica se define por la cantidad de canales de entrada (N), la cantidad de canales de salida (M) y el tamaño del núcleo de convolución (D). La entrada y salida de estas capas puede ser pensada como una imagen de N y M canales respectivamente. Cada capa recibe N canales y aplica una convolución a cada canal con un núcleo cuadrado de tamaño D (uno diferente para cada canal). Luego, los N resultados obtenidos son combinados para obtener un único canal de salida. Este proceso se repite por cada canal de salida definido. Lo que la red neuronal debe aprender son los coeficientes de los núcleos de convolución de cada capa.

Las redes neuronales convolucionales son extremadamente potentes a la hora de procesar imágenes. Las capas convolucionales clásicas aceptan entradas de tamaño arbitrario y son mucho más eficientes al procesar entradas de gran tamaño comparadas con las redes completamente conectadas. Además, en las imágenes la información local tiende a ser mucho más importante que la información global. Es decir, el valor de un píxel probablemente está estrechamente relacionado con los valores de los píxeles cercanos, pero no con aquellos que están a cientos de píxeles de distancia. Las redes convolucionales explotan esta propiedad, dado que aplican convoluciones con núcleos generalmente pequeños. Por último, la cantidad de pesos que debe aprender es relativamente baja, y, si se trabaja con capas convolucionales tradicionales, no depende del tamaño de la entrada de la red.

3.6. Normalización en lote

El aprendizaje profundo ha probado ser una herramienta extremadamente poderosa. El poder computacional al que tiene acceso la humanidad ha aumentado constantemente, por lo que con el pasar del tiempo las redes se han hecho cada vez más profundas.

La profundidad genera un problema: cuando las entradas de una red cambian su distribución (a

causa del aprendizaje durante el entrenamiento), cada neurona debe adaptarse a estos cambios, lo que genera un desbalance que se propaga hacia adelante en la red. Redes de gran profundidad sufren este problema de forma muy notable.

Por esta razón surge en 2015 el concepto de normalización en lote [27]. El concepto es simple: normalizar la media y varianza de las salidas de la red en lote. Se calcula la media y varianza de un mini lote que entra a la neurona y se ajusta adecuadamente ambos valores para cada salida. La normalización por lote se realiza en cada neurona por separado.

Sea μ_B el promedio de las salidas de una neurona dentro del lote B, y sea σ_B^2 la varianza de las mismas dentro del mismo lote. Sea x una de las salidas de la neurona dada una entrada perteneciente al lote B. La nueva salida $\hat{x}$ luego de normalizar es:

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Donde ϵ permite asegurar la estabilidad numérica en caso de que la varianza sea muy cercana a cero, y es un valor arbitrariamente pequeño.

3.7. Convoluciones por profundidad separables

Con la popularidad del aprendizaje profundo y sus aplicaciones en visión por computadora, se ha vuelto cada vez más importante que los modelos entrenados puedan ser ejecutados en dispositivos de bajas prestaciones. En general los dispositivos móviles más usuales (celulares, tablets) no disponen de gran poder computacional, debido a que se priorizan otros aspectos como el costo de fabricación, la vida de la batería y el tamaño de los componentes.

Si bien actualmente los dispositivos móviles se están fabricando con componentes cada vez más poderosos, aún no son capaces de correr modelos pesados como son las redes convolucionales clásicas. Por esta razón algunos investigadores de Google presentan en 2017 un nuevo enfoque al problema: reducir el poder computacional necesario alterando el concepto clásico de convolución.

En el trabajo *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* [28], los investigadores proponen utilizar una nueva arquitectura de convoluciones, conocida como *convoluciones por profundidad separables*. Dada una convolución clásica con M canales de entrada, donde cada canal tiene dimensiones $D_f \times D_f$, N canales de salida y un kernel de tamaño $D_k \times D_k$, el costo computacional de ejecutar una convolución es:

$$D_k \times D_k \times M \times N \times D_f \times D_f$$

En este caso, a efectos de simplificar los cálculos, se asume que la convolución propuesta tiene stride de uno y padding suficiente para generar la salida esperada. Además, se asume que cada canal de salida es del mismo tamaño que los de entrada.

Una convolución por profundidad separable se define a partir de dos convoluciones clásicas. La idea de estas convoluciones es separar las etapas de filtrado y de combinación de una convolución normal en dos pasos separados.

El primer paso, llamado *convolución separable*, consiste en filtrar cada canal de entrada (asumimos M en este caso) con un filtro de tamaño $D_k \times D_k$. De esta convolución se obtienen M canales de tamaño $D_f \times D_f$. El segundo paso involucra una *convolución puntual*, que consiste en combinar linealmente cada

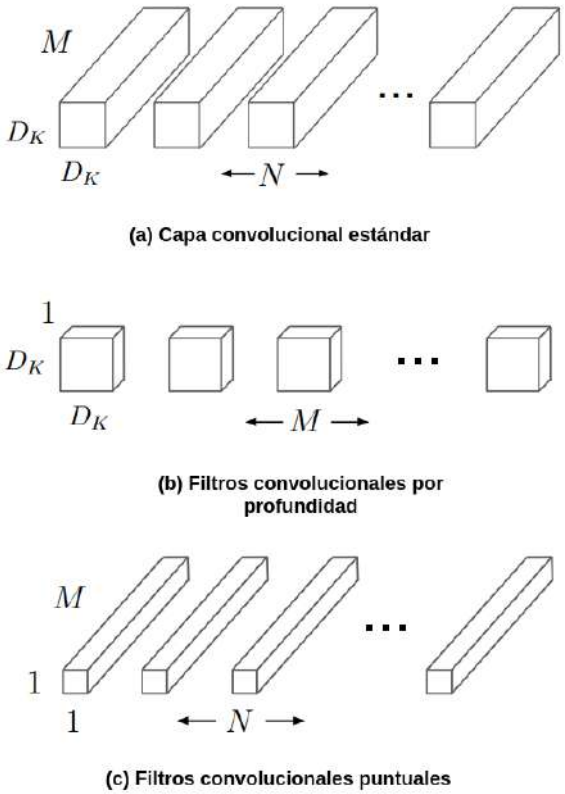


Figura 3.2: Estructura de las capas convolucionales tradicionales y los nuevos tipos de filtros propuestos por los investigadores. Imagen obtenida de [28].

entrada de los M canales obtenidos en la etapa anterior. Esta etapa consiste en una convolución con M canales de entrada, kernels de tamaño 1 y N salidas de tamaño $D_f \times D_f$. En la figura 3.2 se muestra una comparación entre la capa convolucional tradicional y la nueva arquitectura propuesta.

La diferencia principal con las capas convolucionales clásicas es que estas aprenden $M \times N$ filtros. Es decir, por cada canal de salida, se generan M filtros independientes del resto. Las convoluciones por profundidad separables descartan la noción de independencia de los filtros, y aprenden un único filtro por cada canal de entrada. En otras palabras, la cantidad de parámetros necesarios para realizar una convolución separable (la primer etapa del proceso) es independiente de la cantidad de canales de salida.

Esta arquitectura de capa desliga parcialmente la cantidad de canales de entrada y de salida a la hora de calcular la cantidad de parámetros de la capa, convirtiendo un solo término multiplicativo en una suma de dos términos multiplicativos. De esta forma, el poder computacional necesario para ejecutar una capa convolucional por profundidad separable se deduce como:

$$D_k \times D_k \times M \times D_f \times D_f + M \times N \times D_f \times D_f$$

Es interesante notar que tanto en el caso de capas convolucionales clásicas y las propuestas en el trabajo, el poder computacional requerido crece en la misma proporción al aumentar M o D_f. Sin embargo no es así al crecer N o D_k, donde la nueva arquitectura es más eficiente. Los investigadores muestran en su trabajo que estas propiedades generalizan para casos con entradas, salidas y kernels no cuadrados.

3.8. Funciones de pérdida

La función de pérdida es uno de los componentes más importantes a la hora de entrenar una red neuronal. Como está explicado en la sección 3.2, estas funciones permiten cuantificar la diferencia entre el valor que se espera de la red al procesar un ejemplo de entrenamiento y el valor real obtenido. Se tuvieron en cuenta varias alternativas de funciones de pérdida tomando en cuenta sus ventajas y desventajas.

3.8.1. L1

L1 es una de las funciones de pérdida mas simple y también una de las más utilizadas. Consiste en minimizar la sumatoria por píxel de la diferencia absoluta de la imagen de la red y la imagen original.

$$L1 = \sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}|$$

$y^{(i)}$ es el píxel i de la imagen original, $\hat{y}^{(i)}$ es el píxel i de la imágen obtenida como salida de la red. n corresponde a la cantidad total de píxeles de las imágenes.

Entre las ventajas de utilizar la norma L1 se encuentra su resistencia a los *outliers* en los datos, pero por otro lado sus derivadas no son continuas, por lo que dificulta el hallar una solución.

3.8.2. L2

L2 es otra de las funciones de pérdida mas utilizadas y también simple de calcular. Consiste en minimizar la sumatoria del cuadrado de la diferencia entre la imagen luego de ser procesada por la red y la imagen original.

$$L2 = \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

$y^{(i)}$ es el píxel i de la imagen original, $\hat{y}^{(i)}$ es el píxel i de la imágen obtenida como salida de la red. n corresponde a la cantidad total de píxeles de las imágenes.

A diferencia de lo que pasa con la norma L1, la norma L2 brinda una solución estable pero es sensible a los *outliers* en los datos.

3.8.3. SSIM

Si bien las normas L1 y L2 cuantifican la diferencia entre dos imágenes como la distancia entre dos vectores, estas métricas ignoran completamente la percepción humana. Por esta razón también se utilizan otro tipo de funciones de pérdida que miden la similaridad estructural entre las imágenes.

Esta métrica intenta analizar tres aspectos de las imágenes: luminosidad, contraste y estructura. Esto es debido a que la estructura de un objeto en una imagen es independiente de la iluminación [29].

Tomaremos x e y como señales de imágenes no negativas. El primer paso es comparar la luminosidad de las dos señales, para esto se asume que las señales son discretas, y se calcula como la intensidad media:

$$u_x = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Por lo tanto, se define la función de corporación de luminosidad l(x, y) como una función de u_x y u_y. Con esto calculado, removemos la intensidad media de la señal.

Para el contraste utilizamos la desviación estándar:

$$\sigma_x = (\frac{1}{N-1}\sum_{i=1}^{N}(x_i - u_x)^2)^1/2$$

La función c(x, y) es la función de comparación de contraste entre σ_x y σ_y. La señal es normalizada dividiéndola por su desviación estándar.

La comparación estructural s(x, y) es calculada utilizando las señales normalizadas, $(X - u_x)/\sigma_x$ y $(Y - u_y)/\sigma_y$.

Finalmente, los tres componentes son combinados para obtener una medida de similaridad:

$$S(x, y) = f(l(x, y), c(x, y), s(x, y))$$

Para completar la definición de esta medida de similaridad falta definir las funciones l(x, y), c(x, y), s(x, y) y la función de combinación f(·).

Para la comparación de luminosidad, definiremos:

$$l(x, y) = \frac{2u_x u_y + C_1}{u_x^2 + u_y^2 + C_1}$$

donde C_1 es incluida para evitar la inestabilidad cuando $u_x^2 + u_y^2$ es muy cercana a cero.

La función de comparación de contraste se define de forma similar:

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

donde C_2 cumple la misma función que C_1 en l(x, y).

La función de comparación de estructura se efectúa luego de quitar la luminosidad y normalizar, por lo que es definida de la siguiente forma:

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

Finalmente, uniendo las funciones de luminosidad, contraste y estructura, obtenemos la siguiente función:

$$SSIM(x, y) = [l(x, y)]^\alpha \cdot [c(x, y)]^\beta \cdot [s(x, y)]^\gamma$$

donde $\alpha > 0$, $\beta > 0$ y $\gamma > 0$ son parámetros utilizados para ajustar la importancia de los tres componentes.

3.8.4. MS-SSIM

SSIM permite analizar las diferencias estructurales entre dos imágenes. Si bien este enfoque se acerca más al concepto de "percepción humana", no toma en cuenta detalles como, por ejemplo, la distancia de un observador al plano de la imagen. MS-SSIM (*Multi-Scale Structural Similarity Index*) [30] intenta acortar esta brecha computando SSIM en varias resoluciones de las imágenes. Para esto, realiza iterativamente un cálculo de SSIM, aplicando un filtro pasa bajos y subsampleando la imágen en un factor de 2.

MS-SSIM es definida matemáticamente de la siguiente forma:

$$MSSSIM(x,y) = [l(x,y)]^{\alpha_M} \prod_{j=1}^{M} [c(x,y)]^{\beta_j} [s(x,y)]^{\gamma_j}$$

donde M es la cantidad de resoluciones.

3.9. Redes neuronales aplicadas a demosaicing

3.9.1. Syu, Cheng, et al. (2018)

Los autores Syu, Cheng, et al. presentan *Learning Deep Convolutional Networks for Demosaicing*, un trabajo de febrero de 2018 que ataca el problema de demosaicing con redes neuronales convolucionales inspiradas parcialmente en modelos que realizan súper resolución de imágenes.

Un detalle interesante es que la red no se encarga de procesar imágenes completas, sino parches pequeños (33×33), y emite como resultado un parche más pequeño coloreado (22×22) para evitar cometer errores en los píxeles cercanos al borde por falta de información. La red se basa en capas convolucionales exclusivamente y se divide en tres etapas: una etapa de extracción de propiedades (se obtienen propiedades de bajo nivel de la imagen), una etapa de mapeo no lineal entre propiedades de alto nivel y bajo nivel, y luego una etapa de reconstrucción, de la cual se obtiene el parche completamente coloreado.

Para entrenar la red, los investigadores generan un dataset de imágenes tomadas de Flickr, seleccionadas para generar una muestra estadística lo más realista posible, que contenga casos de imágenes con tendencia a la generación de artefactos (por ejemplo, imágenes con componentes de alta frecuencia). Luego aumentan los datos dando vuelta las imágenes horizontal y verticalmente, además de con rotaciones. El dataset se encuentra publicado [1].

3.10. Demosaicing y denoising en conjunto

3.10.1. Chen et al.

En el trabajo titulado *Learning to see in the dark* [31], Chen et al. se proponen atacar un problema del *pipeline* digital de las cámaras: el procesamiento de imágenes con baja iluminación. En este tipo de imágenes el ruido es usualmente un gran problema.

Una estrategia posible para combatir la baja iluminación de una escena es configurar un alto nivel de ISO en la cámara digital, lo que aumenta el brillo, pero también potencia el ruido de las imágenes.

[1] `http://www.cmlab.csie.ntu.edu.tw/project/Deep-Demosaic/`

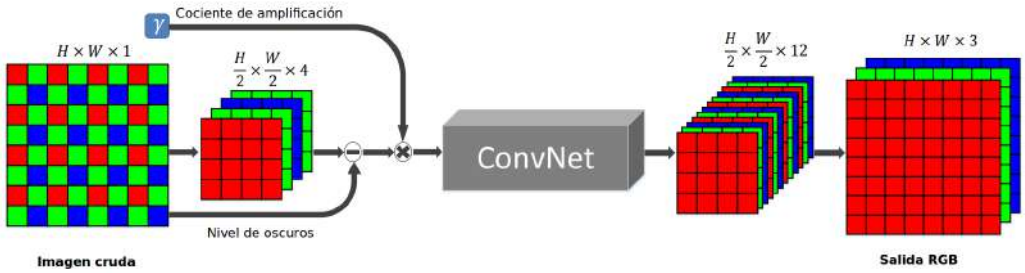


Figura 3.3: Arquitectura de la red convolucional profunda que aprende el *pipeline* de procesamiento digital para imágenes con muy baja iluminación, propuesto por Chen et al. Imagen tomada de *Learning to see in the dark* [31]

Como aporte, los investigadores proponen el diseño de una red convolucional profunda, a la que se le intenta enseñar el *pipeline* digital completo de la cámara. Esto incluye las etapas de corrección de color, *demosaicing*, *denoising* y otro tipo de etapas de mejoras de imagen. Para el entrenamiento utilizan un conjunto de imágenes recopiladas por ellos, enfocado a imágenes con muy poca iluminación. En la figura 3.3 puede verse un diagrama de la arquitectura propuesta por los investigadores.

Un detalle a destacar de la arquitectura es que la última etapa de la red emite 12 filtros donde cada uno corresponde a un canal con un cuarto de la superficie total de la imagen inicial, que luego son unidos para generar la imagen final. Esta es una técnica, utilizada también en otros trabajos, cuyo objetivo es aumentar el potencial de paralelización de la red neuronal aumentando la cantidad de filtros.

Para evaluar los resultados los investigadores utilizaron Amazon Mechanical Turk, una herramienta que permite que seres humanos evalúen las imágenes resultado. Esta es una forma relativamente simple de evaluar los resultados cualitativamente, donde obtuvieron buenos resultados, al igual que cuantitativamente.

Como conclusión, los investigadores plantean que, si bien la red tiene buen desempeño en los experimentos, existen varias oportunidades de mejora. Entre ellas se encuentra que el entrenamiento fue realizado tomando en cuenta el *pipeline* digital de una única cámara, por lo que habría que explorar las posibilidades de generalización.

3.10.2. Khashabi et al. (2014)

En su publicación *Joint Demosaicing and Denoising via Learned Non-parametric Random Fields* [4] de julio de 2014, Daniel Khashabi et al. proponen un algoritmo de aprendizaje automático capaz de realizar las tareas de demosaicing y denoising en simultáneo.

El modelo propuesto por los investigadores se basa en campos de árboles de regresión, y obtiene buenos resultados en base a métricas usualmente relacionadas con la percepción visual humana, como por ejemplo la relación señal/ruido de cresta (*peak signal-to-noise ratio* o *PSNR* en inglés).

Para entrenar su modelo, los investigadores no solo recopilaron imágenes para un dataset, sino que además proponen un algoritmo para generar un dataset para entrenar otros modelos. Este algoritmo es explicado con mayor profundidad en la sección 4.5, y es el utilizado en este proyecto para generar las imágenes utilizadas en la experimentación.

El dataset utilizado por los investigadores está publicado en Internet[2] y es el dataset utilizado en este proyecto para entrenar los diversos modelos.

[2] `https://www.microsoft.com/en-us/download/details.aspx?id=52535`

Si bien este trabajo no utiliza aprendizaje profundo para resolver el problema, trabajos posteriores de otros investigadores [5][6][32] se han basado en él para implementar este tipo de modelos para atacarlo.

3.10.3. Schwartz et al. (2018)

En enero de 2018 se publica *DeepISP: Learning End-to-End Image Processing Pipeline* [6]. En este trabajo los investigadores diseñan e implementan una red neuronal capaz de realizar demosaicing y denoising. Luego expanden el modelo para implementar todo el pipeline de procesamiento digital con un único modelo.

La arquitectura de la red propuesta está dividida en dos etapas: la de bajo nivel y la de alto nivel. La etapa de bajo nivel es la primera en ejecutarse y se encarga de realizar el procesamiento de bajo nivel del pipeline. Estas tareas incluyen demosaicing y denoising, así como la corrección gamma y de blancos. La arquitectura de esta etapa está compuesta exclusivamente por capas convolucionales tradicionales, con el detalle de que utilizan un enfoque residual; es decir, cada capa de la red genera una aproximación de la imagen resultado, que la siguiente capa debe refinar. Esta etapa emite una imagen de 64 canales, de los cuales se considera que tres corresponden a la estimación de la imagen resultado, y los restantes 61 corresponden a información que luego es alimentada a la siguiente etapa de la red.

La segunda etapa (llamada de alto nivel) es la encargada de realizar correcciones a la imagen para mejorar su percepción por humanos. Esta etapa recibe como entrada la información emitida por la etapa de bajo nivel (no recibe la estimación de la imagen, simplemente los restantes 61 canales) y se compone de capas convolucionales seguidas de capas de max pooling. Al final de esta etapa se tiene una capa de pooling global (cada canal inicial se termina reduciendo a un único valor), terminando en una capa completamente conectada que emite un vector de 30 atributos. Estos atributos determinan los parámetros de una transformación que luego se le aplicará a la imagen emitida por la primer etapa.

Para entrenar la red utilizan dos funciones de costo según si se entrena únicamente la etapa de bajo nivel o ambas. Para la etapa de bajo nivel utilizan l_2, mientras que al entrenar ambas etapas conjuntamente se utiliza una combinación de l_1 y MS-SSIM (Multi-Scale Structural Similarity Index), propuesta por [33] como una manera de mejorar la calidad perceptual.

Para evaluar los resultados los investigadores utilizan, además de los usuales métodos de comparación con otras estrategias, el *mean opinion score* (o MOS). Este método de evaluación involucra a humanos, que manualmente evalúan imágenes y determinan qué tan bien se ven.

El trabajo también presenta un dataset de imágenes para aprender el pipeline completo de una cámara digital y se encuentra publicado en Kaggle[3]. Se compone de imágenes de escenas tomadas con la cámara trasera un celular Samsung S7. Para cada escena se toman dos imágenes: una en formato raw y otra en modo automático (a la que se realiza todo el procesamiento del pipeline). Si bien es un dataset que es fácil de construir (no requiere implementar un algoritmo complicado), fuerza a la red neuronal a aprender el pipeline específico implementado por la cámara del celular, lo cual lo convierte en un modelo potencialmente sesgado.

3.10.4. Demosaicnet

Uno de los trabajos más recientes en demosaicing y denoising utilizando aprendizaje automático (en particular, deep learning) es el algoritmo presentado por Michael Gharbi et al. [5]. Este algoritmo consiste en la ejecución simultanea de demosaicing y denoising, y busca obtener buenos resultados en las imágenes que presentan artefactos lumínicos o el efecto Moiré.

[3] `https://www.kaggle.com/knn165897/s7-isp-dataset/data`

La arquitectura utilizada y que es la que utilizamos como base en el desarrollo de nuestra red, consta de los siguientes pasos:

- La entrada se compone de una imagen con un solo canal siguiendo el patrón de bayer, además de un σ estimado que representa el nivel de ruido de la imagen. Denotaremos como F^d a la feature map correspondiente a la d-esima capa.
- El primer paso de la red es separar la imagen de entrada para obtener varios canales con un cuarto de la resolución original de la imagen para poder reducir el coste computacional de los pasos siguientes. Para esto se aplica la siguiente feature map:

  $F_c^0 = M(2x + (c \mod 2), 2y + \lfloor \frac{c}{2} \rfloor)$

  F_c^0 se encarga de extraer parches de 2×2 de la imagen de entrada M y empaquetarlos como una feature map de 4 canales indexados por c. Por ultimo, se le agrega un quinto canal a esta feature map que corresponde al nivel de ruido σ de la imagen.
- Con la imagen reducida a varios canales se procesa la mayor parte de la red, que consiste en D capas convolucionales con filtros de tamaño K x K seguida cada una por una función de activación no lineal, en este caso se utiliza ReLu, dando como resultado W canales. La feature map correspondiente a cada una de estas capas se define de la siguiente manera:

  $F_c^d = f(b_c^d + \sum_{c'=1}^{W} w_{cc'}^d \times F_{c'}^{d-1})$ donde $c \in \{1...W\}$

  Siendo b_c^d un sesgo escalar para el canal c en la capa d, y $w_{cc'}^d$ un filtro de tamaño $K \times K$.
- La ultima feature map F^D utilizando baja resolución da como resultado 12 canales en vez de W. A estos canales se les aplica una superresolucion contraria a la que se implemento en la primer parte, para generar tres imágenes del tamaño de la imagen original. Además de estas tres imágenes, se generan otras tres a partir de mapear la imagen original según los colores del mosaico de bayer. Las feature maps correspondientes a esta capa son las siguientes:

  $F_c^{D+1}(x, y) = m_c(x, y)M(x, y)$ donde $c \in \{1..,3\}$

  $F_c^{D+1}(x, y) = F_c^D(\lfloor \frac{x}{2} \rfloor, \lfloor \frac{y}{2} \rfloor)$ donde $c \in \{4..,6\}$
- En la penúltima etapa se ejecuta una capa convolucional en alta resolución para producir F_c^{D+2} utilizando filtros de $K \times K$ y obteniendo como resultado W canales.
- Por ultimo se aplica una función afín al resultado anterior para generar la imagen resultado de la red, definida por la siguiente función:

  $O_c(x, y) = b_c^O + \sum_{c'} w_{cc'}^O F_{c'}^{D+2}(x, y)$

El dataset utilizado por Demosaicnet fue construido para resolver los problemas causados por artefactos lumínicos o el efecto Moiré. Para generar este dataset lo que hicieron fue utilizar una primera versión de la red (entrenada con el dataset de Imagenet [34]) y procesar miles de parches de imágenes descargadas de la web, luego con los parches que fallaban utilizaron distintas métricas para poder hallar imágenes con artefactos lumínicos y efecto Moiré.

Para el entrenamiento de la red se minimizo la norma L_2, mientras que para la optimización se utilizo ADAM. Esta red fue desarrollada utilizando Caffe y entrenada en una NVIDIA Titan X, lo que llevo entre unas dos y tres semanas.

Capítulo 4

Solución propuesta

El problema concreto que se plantea resolver es el siguiente: Dada una imagen de un sólo canal con ruido, capturada en un sensor lumínico utilizando el mosaico de Bayer, se desea diseñar e implementar un algoritmo capaz de procesar esa imagen y retornar una imagen RGB y sin ruido. Esta imagen resultado debería estar coloreada de la forma más fiel a la realidad; es decir, debería ser indistinguible de una imagen capturada con tres sensores, uno para cada color.

Como resultado final se propone un modelo de aprendizaje automático implementado a través de una red neuronal convolucional, que es un tipo particular de red neuronal. A lo largo de este capítulo se formaliza el problema a resolver y posteriormente se muestran los aspectos más relevantes de dicha solución. En general el diseño de una red neuronal se ve afectado por las decisiones tomadas en ciertos componentes fundamentales: la función de pérdida a utilizar, la arquitectura de la red (en cuanto a capas e interacciones entre las mismas), el optimizador utilizado, el conjunto de datos utilizado para entrenar y el algoritmo de entrenamiento. Malas decisiones tomadas al definir e implementar estos componentes podría ser la diferencia entre un modelo que aprende bien la tarea para la que se lo diseñó o uno que no.

Antes de plantear el modelo implementado como solución, en la sección 4.1 se define el problema en términos formales. En el resto de las secciones se detallan lo siguientes componentes:

- **4.2** *Función de pérdida:* función que se utiliza durante el entrenamiento para medir el desempeño de la red.
- **4.3** *Arquitectura:* estructura de la red en término de capas. Se detalla el tipo de capas que componen la red y la forma en la que están distribuidas. Además, se habla sobre el optimizador utilizado.
- **4.4** *Implementación:* en esta sección se encuentra todo lo relativo al hardware y plataformas utilizadas para desarrollar y entrenar la red.
- **4.5** *Dataset:* se explica el proceso utilizado para generar el dataset utilizado para entrenar la red.
- **4.6** *Entrenamiento:* se detalla el proceso utilizado para entrenar la red.

4.1. Formalización

Sea $P^{M \times N}$ el conjunto de entradas de una imagen de tamaño $M \times N$ tal que $P^{M \times N} = \{a_{i,j}\},\ i \in \{1, \ldots, M\},\ j \in \{1, \ldots, N\}, a_{i,j} \in Q \forall i, j$, donde Q es el conjunto de valores posibles para las entradas de las imágenes.

Sea $x \in S^{M \times N}$, donde x representa la imagen previa a la realización de demosaicing y denoising, y $S^{M \times N} = \{p_i\}, p_i \in P^{M \times N}$ el conjunto de imágenes de tamaño $M \times N$ de un solo canal. Sea $y \in S_3^{M \times N}$ donde $S_3^{M \times N} = \{s_1, s_2, s_3\}, s_i \in S^{M \times N} \forall i$ corresponde al conjunto de imágenes de tamaño $M \times N$ de tres canales. y representa la imagen ideal obtenida luego de aplicar demosaicing y denoising a x.

Se desea aprender la función ideal $D : S^{M \times N} \rightarrow S_3^{M \times N}$ de forma de que $D(x) = y$. Sea $\hat{D}$ una aproximación de D, se define una función de pérdida L tal que si $D(x) = \hat{D}(x) \rightarrow L(D(x), \hat{D}(x)) = 0$. El problema se puede expresar como un problema de optimización definido de la siguiente manera.

$$\begin{aligned} \min_{x} \quad & L(D(x), \hat{D}(x)) \\ \text{s.a} \quad & x \in S^{M \times N} \end{aligned}$$

4.2. Función de pérdida utilizada

Tomando en cuenta las funciones definidas anteriormente en la sección 3.8, en este proyecto hemos decidido unir las funciones de MS-SSIM y L1 siguiendo lo mostrado por Hang Zhao et al. [33].

Algunas de las desventajas con las que cuenta MS-SSIM es su sensibilidad a sesgos uniformes, lo que puede afectar el brillo o generar cambios de colores, aunque por otro lado preserva el contraste en regiones con alta frecuencia mejor que las otras funciones. Mientras que L1 preserva los colores y la luminosidad pero no produce el mismo contraste que MS-SSIM.

Para obtener las mejores características de estas dos funciones, Hang Zhao et al. [33] proponen la siguiente función:

$$L^{mix} = \alpha \cdot L^{MS-SSIM} + (1 - \alpha) \cdot G_{\sigma_G^M} \cdot L^{L1}$$

Siendo $L^{MS-SSIM}$ y L^{L1} los resultados de las funciones de perdida de MS-SSIM y L1 respectivamente, y $G_{\sigma_G^M}$ el coeficiente Gaussiano.

A pesar de las buenas propiedades de convergencia de L1 y L2, en este trabajo hemos decidido utilizar conjuntamente MS-SSIM y L1 basándonos en los buenos resultados obtenidos por Hang Zhao et al. [33] con respecto al resto de las funciones de pérdida.

En este proyecto tomamos $\alpha = \frac{1}{2}$ para dar un peso equitativo a MS-SSIM y L1.

4.3. Arquitectura de la red

Cómo una red neuronal está estructurada influye enormemente en la capacidad del modelo de representar la función a aprender [35]. Una red con mayor cantidad de pesos y mayor cantidad de capas puede aproximar funciones más complejas que una red pequeña. Sin embargo, las redes muy profundas y con muchos pesos presentan problemas. Uno de los más inminentes es el sobreajuste (*overfitting* en inglés) donde la red se ajusta a ciertas características muy específicas de los datos de entrenamiento, perdiendo generalidad para procesar datos fuera de ese conjunto.

Considerar estos y otros factores es imperativo para diseñar un buen modelo, capaz de aprender la función objetivo con la mayor exactitud y confiabilidad posible.

En el presente proyecto se desarrollaron dos redes neuronales para resolver el problema planteado. La primera de ellas, bautizada Símil Demosaicnet, consiste en una re-implementación de la red presentada

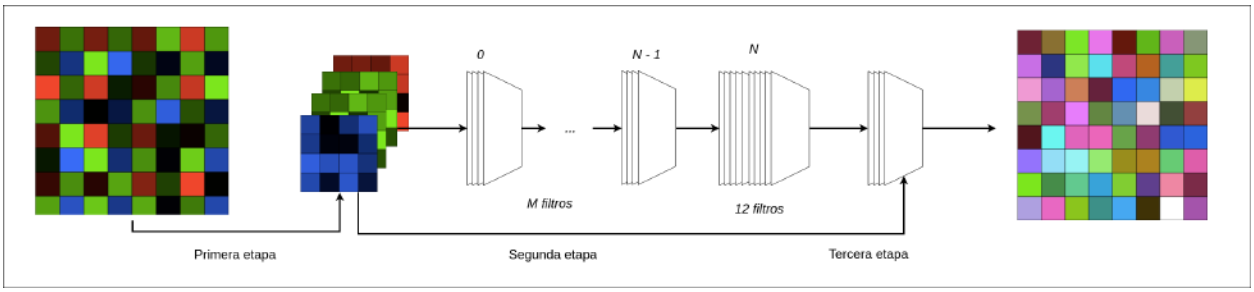


Figura 4.1: Arquitectura de Demosaicnet. En este proyecto el valor de N fue de 50, mientras que la cantidad de filtros utilizada por capa (M) fue de 20.

por Gharbi et al. [5], con la diferencia de que la profundidad de la red es menor para facilitar el entrenamiento con el *hardware* del que se dispuso en el proyecto. Por otro lado se implementó una segunda red neuronal convolucional que corresponde a la arquitectura de Símil Demosaicnet pero utilizando convoluciones por profundidad separables. A esta última se la refiere como Mobile Demosaicnet, haciendo alusión a *Mobilenet* [28]. En esta sección se detalla la arquitectura de ambas redes y se presentan las diferencias entre ambas.

4.3.1. Símil Demosaicnet

Como paso inicial, se implementó una red neuronal convolucional con una arquitectura similar a la de Demosaicnet [5]. Esta red neuronal se compone de tres etapas: una etapa inicial donde la imagen de entrada se separa en cuatro canales, uno por cada celda del mosaico de Bayer; luego el cuerpo central de la red compuesto por varias capas convolucionales convencionales; y por último una etapa que involucra la súper resolución de los canales previamente separados, para luego ser combinados por una capa convolucional más. En la figura 4.1 puede observarse un diagrama de alto nivel donde se muestran las diferentes etapas.

Primera etapa La entrada de la red es una imagen de un solo canal. Esta imagen debería corresponder a la obtenida del sensor lumínico de la cámara, capturada con el mosaico de Bayer. La imagen es separada en cuatro canales, correspondientes a las cuatro celdas del patrón de Bayer (un canal para el rojo, dos para el verde y uno para el azul).

Separar la imagen original en varios canales permite que a lo largo de la red se trabajen con matrices más pequeñas. Si bien la información que ingresa a la red es la misma en ambos casos, el separar la imagen en varios canales permite que los resultados intermedios sean más pequeños.

Segunda etapa La imagen de cuatro canales obtenida en la etapa anterior es procesada por cincuenta capas convolucionales, con núcleos de tamaño 5×5. Cada una de estas capas dispone de veinte filtros. Los investigadores proponen que esta arquitectura en particular tiene mejores resultados al aumentar la profundidad, con una cantidad relativamente baja de filtros por capa. La última capa convolucional dispone de doce filtros en vez de los veinte antedichos. La idea es que esta última capa se utilice posteriormente para generar la imagen resultado.

Tercera etapa Una vez terminada la etapa central de la red, la etapa final construye una imagen resultado. Primero, se genera una imagen de tres canales, a partir de la imagen de entrada de la red (es decir, la imagen de un solo canal). Al generar la imagen de tres canales, los valores de los píxeles que no se conocen se definen como cero. Luego se toma la salida de la última capa convolucional, de doce canales, y se construye una imagen de tres canales, donde los doce canales son separados en grupos de cuatro, a partir de los cuales

se construye un canal. Este diseño pseudo residual permite que la red aprenda exactamente cómo manejar los residuos, lo que la hace más flexible. Luego de esto se concatenan las dos imágenes de tres canales y se procesan en una capa convolucional idéntica a las de la segunda etapa (núcleo de 5×5, y veinte filtros). Por último se dispone de una capa convolucional que emite una imagen de tres canales.

4.3.2. Mobile Demosaicnet

Los trabajos más recientes que utilizan aprendizaje profundo para resolver el problema de demosaicing y denoising consisten principalmente en redes relativamente pesadas. Si bien los trabajos obtienen buenos resultados, las redes que presentan son virtualmente inaplicables en la realidad, donde la mayoría de los dispositivos que incorporan una cámara digital tienen muy pocos recursos de cómputo gráfico (si es que disponen de *hardware* de este tipo en primer lugar).

Las cámaras digitales muy usualmente disponen de un poder computacional mucho menor al de las máquinas que entrenan y ejecutan las redes neuronales. Probablemente, de todos los dispositivos con cámaras digitales, los más poderosos computacionalmente sean los celulares de última generación. Por esta razón, es importante no solamente diseñar y entrenar un modelo que aproxime la función objetivo con la mayor exactitud posible, sino también considerar cómo estos modelos serían aplicados en la realidad.

Por esta razón, en el presente proyecto se decidió explorar la propuesta de *Mobilenet* [28]. En su trabajo, los investigadores proponen aplicar el concepto de convolución separable por profundidad a la visión por computadora, en particular el problema de clasificación. La propuesta tiene éxito aumentando la velocidad de entrenamiento y procesamiento de las redes, dado que disminuye enormemente la cantidad de parámetros de la red, con un compromiso relativamente pequeño en precisión.

Cabe aclarar que el trabajo de *Mobilenet* ataca problemas de visión por computadora de alto nivel, como por ejemplo la clasificación de imágenes. En el presente trabajo, se ha diseñado una versión de la anterior arquitectura presentada, pero utilizando convoluciones separables por profundidad.

La única diferencia notable entre las dos arquitecturas es que cada convolución tradicional con parámetros entrenables ha sido reemplazada por una convolución separable por profundidad.

4.3.3. Diferencias entre las arquitecturas

Si bien ambas arquitecturas son bastante similares, las convoluciones separables por profundidad influyen enormemente en la cantidad de operaciones de la red. En el caso de las arquitecturas presentadas, el cambio a este tipo de convoluciones disminuye la cantidad de operaciones en aproximadamente un %88.

La arquitectura con convoluciones estándar requiere de aproximadamente de 500,000,000 de operaciones, mientras que al utilizar las nuevas convoluciones este número baja a 60,000,000.

4.4. Implementación

Para este proyecto se tuvieron en cuenta dos librerias de implementacion de modelos de aprendizaje profundo: *Pytorch*[1] y *Tensorflow*[2].

Pytorch es un *framework* de aprendizaje profundo de código abierto mantenido principalmente por

[1] https://pytorch.org
[2] https://www.tensorflow.org/

Facebook y basado en la herramienta *Torch*. Está escrito en Python y permite implementar redes neuronales de todo tipo fácilmente, y entrenarlas en diversos dispositivos de computación gráfica.

Otro de los *frameworks* más conocidos para desarrollar modelos de aprendizaje profundo es *Tensorflow*, creado por Google para implementar sus propios algoritmos de aprendizaje profundo.

Si bien Tensorflow es más popular en cuanto a la comunidad, Pytorch también tiene una comunidad muy activa de usuarios, y en comparación con el primero tiene una curva de aprendizaje menos pronunciada, lo que permite acelerar el proceso de prototipado de modelos, y lograr un sistema funcional en menor tiempo para realizar pruebas.

Durante el proyecto fue necesario realizar experimentos rápidos con la arquitectura de la red. La intuitividad del desarrollo con Pytorch resultó esencial para satisfacer esta necesidad. Sin embargo, la herramienta presenta algunas dificultades, detalladas más adelante. El código se ha escrito para la versión 3.6.4 de Python utilizando la versión 0.4.1 de Pytorch, y ha sido entregado junto con este informe.

Tanto el entrenamiento de la red como los benchmarks realizados para comparar fueron realizados sobre el mismo hardware. La máquina utilizada dispone de un procesador Intel i5-8400 de 2.80 GHz, con 16 Gigas de memoria y una tarjeta gráfica NVIDIA GeForce GTX 1070 Ti. Todo el software implementado y ejecutado para la realización del proyecto fue hecho en Ubuntu 16.04. La versión de Cuda utilizada fue la 9.2.

Problemas de performance

Al implementar las convoluciones por profundidad separables en Pytorch se encontraron problemas de performance. En particular, la forma de ejecutar este tipo de convoluciones es muy lento en comparación a la cantidad de parámetros y operaciones. En muchos casos resultó ser más lenta la ejecución de estas convoluciones que de las tradicionales. Estos datos se muestran en la sección 5.2.3.

La causa de este problema se le atribuye a la falta de soporte en Pytorch[3]. Si bien la mejora en tiempo al utilizar estas convoluciones no es posible de determinar (suponiendo que existe soporte), se decidió utilizar como métrica de análisis de performance la cantidad de pesos y operaciones necesarias.

Al encontrar este problema se decidió seguir utilizando Pytorch por varias razones. En primera instancia, ya se disponía de una cantidad importante de código que, si bien fue escrito de forma modular para minimizar el acoplamiento, estaba estructurado alrededor de la interfaz propuesta por Pytorch para trabajar. La inversión en re trabajo necesaria para migrar el código a Tensorflow resultó inviable para los tiempos manejados en el proyecto.

Por otra parte el problema no fue limitante en cuanto a la capacidad de implementar estas convoluciones, sólo afectó el tiempo de ejecución. Por esta razón se decidió realizar la comparación de los modelos propuestos tomando en cuenta la cantidad de parámetros necesarios para entrenar, lo que está asociado directamente con el tiempo de ejecución de la red.

4.5. Dataset

Una de las tareas principales a la hora de diseñar un modelo de aprendizaje automático, es la generación de un dataset adecuado para resolver el problema. Conseguir una cantidad aceptable de datos que describan de forma fiel la realidad del problema es tan importante como implementar un buen algoritmo que aprenda de ellos.

[3] https://github.com/pytorch/pytorch/issues/1708

Más específicamente, generar un conjunto de datos para aprender demosaicing y denoising no es una tarea simple. Una de las razones principales de esta dificultad es que no existe un método de demosaicing perfecto que se pueda tomar como referencia.

¿Cómo generar un dataset que proponga ejemplos de demosaicing y denoising cuando estas son las tareas que justamente se quieren aprender? Khashabi et al. [4] proponen una solución inteligente a este problema. Se presenta esta solución de forma detallada en esta sección del informe.

4.5.1. Requisitos del dataset generado

Se desea encontrar un proceso de generación de datos de entrenamiento de demosaicing y denoising para un cierto algoritmo. Estos datos de entrenamiento deben ser pares de imágenes. La primera de ellas con un sólo canal, que representa la imagen captada por los sensores de la cámara a la que se le está por aplicar demosaicing y denoising. La segunda imagen debe tener tres canales (suponiendo que se trabaja con RGB) y representa el resultado de aplicar demosaicing y denoising a la primera (y solamente este par de transformaciones).

Usualmente las cámaras digitales profesionales soportan la toma de fotos en formato crudo (raw). Esta práctica no es inusual en el trabajo de fotógrafos, dado que permiten la configuración manual de muchos aspectos que usualmente la cámara controla automáticamente (por ejemplo exposición y velocidad del obturador). Este tipo de imágenes reciben un procesamiento mínimo, que en general está limitado únicamente al proceso de digitalización de la señal lumínica, por lo que son emitidas con un sólo canal. Estas imágenes en formato crudo sirven, a priori, como base para generar la primer imagen de cada par del dataset. Teniendo esta imagen inicial, ¿cómo se puede generar la imagen con demosaicing y denoising perfecto?

La forma más utilizada de tomar imágenes digitales es utilizando una cámara digital. Esto limita seriamente la posibilidad de obtener datos perfectos para utilizar en el entrenamiento. Cualquier imagen que se capture con una cámara será procesada en el pipeline interno, que está diseñado por el fabricante, donde además de demosaicing y denoising se aplican otro tipo de transformaciones (lineales y no lineales). El diseño e implementación de este pipeline está a cargo de cada fabricante, y cada uno lo implementa de maneras potencialmente muy diferentes. Por esta razón, aunque se pudiera pedir a la cámara que emita las imágenes crudas y procesadas, el algoritmo que aprenda de ellas estará condenado a aprender el pipeline de la cámara (o cámaras) seleccionadas.

Otro problema es que las cámaras no solo realizan demosaicing y denoising, sino que también otras transformaciones lineales y no lineales (corrección de oscuridad, corrección de intensidad lumínica, corrección gamma), por lo que si se toma la salida procesada de la cámara como fuente de verdad, se estaría obligando al algoritmo a aprender estas transformaciones también.

El objetivo del presente trabajo es diseñar e implementar un modelo de aprendizaje profundo que consiga aprender las tareas de demosaicing y denoising de forma conjunta. Esto implica que el dataset debe representar únicamente el resultado de aplicar este par de transformaciones a una imagen. Es importante encontrar una manera de realizar demosaicing y denoising de forma agnóstica al procesamiento de cualquier cámara digital.

4.5.2. Propuesta de Khashabi et al. (2014)

En su trabajo, Khashabi et al. se enfrentan al mismo problema planteado anteriormente: generar un dataset de imágenes útiles para entrenar un algoritmo en las tareas de demosaicing y denoising. Los investigadores fundamentan la necesidad de generar un dataset basándose en las prácticas contemporáneas de otros investigadores, que utilizaban datasets de imágenes que incluían otro tipo de procesamiento (incluso transformaciones no lineales como es la corrección Gamma). Este tipo de datasets, argumentan Khashabi

et al., corresponden a un problema artificial, diferente al real. Un componente de software que se encarga explícitamente de realizar demosaicing debe aprender únicamente esa tarea.

Los investigadores hacen dos grandes aportes relevantes para el presente proyecto: presentan un dataset de imágenes para las tareas de demosaicing y denoising, y además publican el algoritmo utilizado para generarlo. La estructura del dataset se detalla en secciones posteriores. A continuación se expone el algoritmo utilizado para generar los pares de imágenes necesarios.

Modelo del Pipeline Digital

El pipeline de una cámara digital se compone de varios pasos que, como dicho anteriormente, dependen del fabricante. En particular, los investigadores identifican varias etapas de interés, implementadas en el pipeline en este orden:

1. **Arreglo de Filtros:** etapa donde la luz es filtrada para obtener algunos de los colores necesarios para generar cada pixel. Este arreglo de filtros es usualmente implementado en base al Mosaico de Bayer.
2. **Conversión Análoga-Digital:** etapa donde la señal lumínica captada por los sensores es traducida a una señal digital. La salida de esta etapa es lo que se considera una imagen en formato Raw (crudo).
3. **Transformaciones Lineales:** luego de obtener la señal digital se realizan algunas correcciones a la imagen.
4. **Demosaicing y Denoising:** etapa donde se toma la imagen en escala de grises y se convierte a una imagen con tres canales. Además, se estima y se elimina el ruido de la imagen.
5. **Corrección de Imagen:** en esta etapa se realizan las transformaciones no lineales que realizan correcciones. Por ejemplo, la corrección gamma se encarga de adaptar la variación de color de la imagen al ojo humano.

Los investigadores proponen, en base a este modelo del pipeline digital, el algoritmo que genera los pares de imágenes. El algoritmo fue diseñado de forma que la primera imagen del par sea la obtenida luego del paso 3 en el pipeline. La segunda imagen del par se corresponde con la salida del paso 4. Es importante notar que los investigadores generan el dataset a partir de imágenes tomadas utilizando el patrón de Bayer como arreglo de filtros.

4.5.3. Algoritmo de Generación del Dataset

Partiendo de una imagen en formato raw, los investigadores hacen uso de la herramienta *dcraw*[4], utilizada para interpretar imágenes en este formato y aplicarles un pipeline relativamente genérico. Dado que las etapas 1 y 2 del pipeline ya han sido efectuadas por la cámara, el algoritmo comienza aplicando la tercera etapa del pipeline utilizando dcraw. La salida de la tercer etapa es una imagen de un sólo canal.

La siguiente etapa consiste en generar la segunda imagen del par. Para esto se necesitará aplicar demosaicing de forma ideal, para lo que se aplica subsampleo sobre la imagen original. Al subsamplear cambia naturalmente la distribución del ruido sobre la imagen, por lo que también los investigadores enfrentan este problema.

[4] `https://www.cybercom.net/~dcoffin/dcraw/`

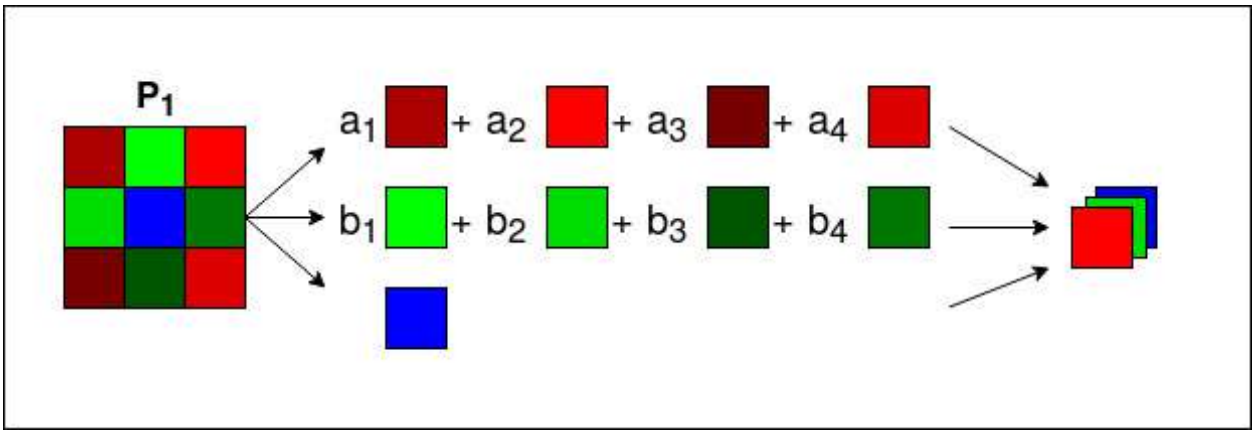


Figura 4.2: un parche de la imagen original es convertido a un único pixel en la imagen resultado. El parche inicial proviene de una imagen de un solo canal, pero se ha coloreado para mostrar el patrón utilizado.

Subsampleo

La estrategia de los investigadores para implementar demosaicing ïdeal"sin utilizar ningún algoritmo conocido es realizar subsampleo de la imagen de entrada.

La imagen es dividida en parches de $M \times M$ disjuntos y luego es subsampleada de forma de que cada parche se mapea a un pixel de la nueva imagen subsampleada. Para determinar el valor de cada canal de este nuevo pixel, se realiza un promedio ponderado de los sensels del parche que corresponden a ese canal. Este proceso se encuentra ilustrado en la figura 4.2.

En la figura, un parche de tamaño 3×3 es convertido a un pixel. Es importante notar que dado que es un promedio ponderado, $\sum_i a_i = 1 \forall i \in \{1, 2, 3, 4\}$ y $\sum_i b_i = 1 \forall i \in \{1, 2, 3, 4\}$. Otro detalle interesante es que no necesariamente todos los canales van a estar igualmente representados en el parche. Como puede verse, el canal azul es representado únicamente por un solo sensel del parche.

Una pregunta importante en este paso es qué tamaño darle a los parches. Los investigadores exploran varias opciones, y concluyen que los parches de tamaño impar son los ideales, y muestran que un simple promedio de los sensels de cada canal genera un pixel que no tiene desviación de color. Además, muestran que un subsampleo de este tipo con parche de tamaño impar suficientemente grande genera una imagen sin ruido (en las siguientes secciones se trata el ruido de las imágenes). En particular, los investigadores concluyen que un parche de tamaño 5×5 elimina virtualmente casi todo el ruido de la imagen.

El resultado de este proceso es, en conclusión, una imagen con tres canales que tiene un valor definido de cada canal para cada pixel, lo cual es el objetivo del demosaicing. La desventaja es que el tamaño de la imagen resultado es mucho menor al de la imagen original, por lo que son incompatibles como entrada y salida de un algoritmo de demosaicing clásico.

Inconsistencia de Tamaño

El subsampleo naturalmente reduce el tamaño de la imagen original. Esto es un problema, dado que las imagenes de los pares utilizados para entrenar un modelo para realizar demosaicing y denoising deben ser del mismo tamaño.

Para resolver la inconsistencia de tamaños, los investigadores descartan la imagen original utilizada, y simplemente eliminan los valores de los canales apropiados de la imagen subsampleada obtenida en la

etapa anterior. Esta estrategia aporta el beneficio adicional de que es posible crear un dataset adaptado para cámaras que tomen imágenes utilizando un arreglo de filtros distinto al de Bayer.

Sin embargo, se presenta otro problema: las imágenes subsampleadas mediante el procedimiento explicado reducen el ruido hasta un nivel prácticamente despreciable [4]. Por esta razón, los investigadores presentan una estrategia para estimar y agregar ruido a las imágenes.

4.5.4. Estimación y agregado de ruido

Luego del subsampleo, la imagen resultado contiene ruido despreciable (suponiendo que se utilizó un tamaño de bloque suficientemente grande). En casos donde el objetivo es entrenar un algoritmo que aprenda a realizar únicamente demosaicing, el agregado de ruido es innecesario.

Khashabi et al. diseñan el algoritmo de generación de imágenes para soportar agregado de ruido artificial. Para esto es necesario estimar el ruido de la imagen raw original. Los investigadores se basan en el modelo estudiado por Foi et al. [10].

En su trabajo, Foi et al. describen el modelo de ruido en las imágenes digitales como una suma de dos componentes: uno poissoniano y uno gaussiano. El modelo poissoniano describe el ruido causado por la lluvia de fotones sobre el sensor (los fotones arriban siguiendo una distribución poissoniana), mientras que el modelo gaussiano describe el ruido relativo a la carga eléctrica de los sensores.

Foi et al. no solo proponen un modelo de ruido para las imágenes digitales, sino que además proponen un algoritmo para estimar la distribución del ruido. El proceso se explica a continuación.

Se asume que la imagen resultante del subsampleo de la sección anterior está libre de ruido. Previo al subsampleo, se estima la distribución de error en la imagen original. Una vez obtenidos estos valores, se realiza el subsampleo y luego se agrega ruido artificial en la imagen subsampleada siguiendo la distribución estimada.

Los investigadores definen un modelo genérico del ruido de una imagen, dado por la siguiente igualdad:

$$z(x) = y(x) + \sigma(y(x))\xi(x)$$

Donde z es la imagen ruidosa, y es la imagen ideal sin ruido, x denota un píxel, ξ representa ruido aleatorio con media cero y desviación estándar 1. σ denota la función que, dado un valor de la señal retorna el valor de la desviación estándar del componente de ruido global.

El objetivo del algoritmo es llegar a estimar la función σ a partir de una única imagen inicial. El algoritmo inicialmente intenta estimar localmente pares de esperanza y desviación estándar a partir de secciones de la imagen inicial. Luego se intenta estimar un par global basándose en esos valores previos.

Análisis del dominio de wavelets

La idea de esta etapa es analizar el ruido transformando la imagen al dominio wavelet (función matemática que representa una señal) [36]. A la vez, la imagen resultante es subsampleada. La transformación realizada responde a la siguiente ecuación:

$$z^{\text{wdet}} = \downarrow_2 (z \circledast \psi)$$

Donde $\downarrow_2$ es el operador de subsampleo, que descarta píxeles correspondientes a filas o columnas pares. $\circledast$ es el operador de convolución, y ψ es la función discreta de wavelet, con norma ℓ^2 igual a 1 y media cero.

Se definen, además, los *coeficientes de aproximación normalizados* como:

$$z^{\text{wapp}} = \downarrow_2 (z \circledast \phi)$$

Donde ϕ se corresponde con la función de wavelet de escalado correspondiente, normalizada de forma de que $\sum \phi = 1$. En este caso, los investigadores seleccionan ψ y ϕ de forma de que se cumpla la siguiente relación:

$$\psi = \psi_1 \circledast \psi_1^T$$
$$\phi = \phi_1 \circledast \phi_1^T$$

Donde ψ_1 y ϕ_1 son funciones de wavelet y escalado de Daubechies [37] respectivamente.

Segmentación

En esta etapa la imagen resultante de la etapa anterior es separada en conjuntos de nivel, donde se puede asumir con razonable confianza que los píxeles de cada nivel se encuentran distribuidos uniformemente en un intervalo cercano a cierto valor.

Previo a la separación en conjuntos de nivel, los investigadores utilizan un filtro de suavizado espacial para disminuir el ruido y un detector de bordes para evitar toparse con bordes al momento de realizar el análisis.

Para realizar el suavizado, los investigadores utilizan un filtro pasa-bajos de tamaño 7×7, de forma de que el suavizado sea potente, y elimine la mayor parte del ruido. Para la detección de bordes se utiliza el enfoque de gradiente, donde el gradiente de la imagen en un punto dado se compara con una estimación de la desviación estándar local. Se el conjunto de suavizado como:

$$X^{\text{smo}} = \{x \in \downarrow_2 X : |\nabla(\Lambda(z^{\text{wapp}}))(x)| + |\Lambda(z^{\text{wapp}})(x)| < \tau \cdot s(x)\},$$
$$\Lambda(z^{\text{wapp}}) = \nabla^2 \text{medfilt}(z^{\text{wapp}}),$$

Donde $\downarrow_2 X$ es el subsampleo por eliminación de z^{wapp}, ∇ y ∇^2 corresponden a los operadores de primera y segunda derivada respectivamente, `medfilt` representa un filtro de mediana de tamaño 3×3, y τ es un umbral positivo constante.

Del conjunto de suavizado se extraen los conjuntos de nivel explicados anteriormente, caracterizados por su valor central y desviación permitida. Los conjuntos de nivel se denotan como $S_i, i \in \{1, \ldots, N\}$, donde N es determinado por el valor central y desviación permitida de todos los conjuntos. Los conjuntos de nivel son disjuntos entre sí.

Estimación local y estimación por máxima verosimilitud

Para cada S_i se define la variable desconocida y_i como:

$$y_i = \frac{1}{n_i} \sum_{j=1}^{n_i} E\{z^{\text{wapp}}(x_j)\}, \quad \{x_j\}_{j=1}^{n_i} = S_i$$

El valor de y_i representa el estimado del valor real de la señal para el conjunto S_i. Dado que la variable es desconocida, se aplica un estimador no sesgado para estimar su valor. Luego de haber estimado y_i, se estima el valor de σ_i con un estimador no sesgado. Los investigadores demuestran en su trabajo que ambos estimadores no son sesgados.

En base a las estimaciones realizadas ($\{\hat{y}_i, \hat{\sigma}_i\}_{i=1}^{N}$) los investigadores realizan una estimación global de y y σ.

Problemas de clipping

El algoritmo presentado no toma en cuenta un detalle no menor en el error de las imágenes: el fenómeno de clipping. El clipping ocurre cuando un sensor recibe una señal intensa, cuya intensidad es recortada para satisfacer el rango dinámico discreto de la señal digital que emite. Esto introduce un claro componente no lineal a la estimación del ruido.

Los investigadores consideran el caso de clipping en su modelo de ruido, y adaptan el algoritmo anterior para soportar esto. Este caso no se discutirá en este informe.

Modelo final de ruido

Foi et al. asumen que el ruido de una imágen se modela como la suma de dos componentes independientes (uno gaussiano y otro poissoniano). El modelo poissoniano depende del valor de la señal, mientras que el gaussiano es independiente:

$$\sigma(y(x))\xi(x) = \eta_p(y(x)) + \eta_g(x)$$

Estos dos componentes pueden ser caracterizados de la siguiente manera:

$$\chi(y(x) + \eta_p(y(x))) \sim P(\chi y(x)), \quad \eta_g(x) \sim N(0, b)$$

Donde $\chi, b \geq 0$ son parámetros escalares, y P y N corresponden a las distribuciones de Poisson y normal respectivamente. Sea $a = \chi^{-1}$, el resultado final del algoritmo de estimación de ruido son los dos parámetros a y b. Con estos dos parámetros, la distribución del ruido de una imagen es completamente modelable, y por ende, reproducible.

4.5.5. Dataset de Khashabi et al. (2014)

El dataset utilizado para entrenar es el entregado por Khashabi et al., en particular se utilizó el subconjunto de imágenes Panasonic con ruido, que contiene quinientos ejemplos para entrenar. Estos ejemplos fueron tomados con la cámara Panasonic Lumix DMC-LX3, y consisten en imágenes de tamaño 220×132. Una selección de imágenes de este dataset puede ser apreciada en la figura 4.3.

Figura 4.3: Algunas de las imágenes del dataset propuesto por *Khashabi et al.*

4.5.6. Detalles de implementación

Si bien los investigadores aportan un dataset con ruido y con ruido eliminado, se ha decidido implementar el algoritmo de generación de pares de imágenes para poder generar nuevos ejemplos de prueba con cámaras más modernas, y probar la solución propuesta, así como otras ya existentes.

Junto con este informe se hace entrega del código implementado en Python. Al igual que Khashabi et al., el código utiliza la herramienta *dcraw* para implementar las etapas iniciales del pipeline. El subsampleo ha sido implementado manualmente, mientras que el estimado de error de las imágenes originales se hace utilizando la herramienta provista por Foi et al. escrita en Matlab[5]. El código para generar ruido es público, pero es descargable únicamente en formato de código p de Matlab, por lo que no es posible acceder al código fuente. Esto dificultó el uso de la herramienta, ya que no es posible depurar el código.

El código generado para este proyecto se ha publicado individualmente en un repositorio público en Github[6]. En la figura 4.4 pueden verse dos ejemplos de imágenes generadas por el algoritmo, una con ruido agregado artificialmente y otra sin ruido.

Problemas en el agregado de ruido

El código proporcionado por Foi et al., y utilizado en este proyecto, presenta algunas dificultades con imágenes de cámaras modernas.

El principal que se presentó fue que la herramienta de Foi et al. no siempre estima el error correctamente. Para ciertas imágenes, la estimación retorna valores no razonables de a y b. En algunos casos son valores muy altos, lo que resulta en imágenes demasiado ruidosas, en las que incluso el estado del arte tiene desempeño muy pobre. En otros casos los valores estimados son negativos, lo que los hace inválidos. En la figura 4.5 puede apreciarse la diferencia entre dos parches para los cuales se estimó y posteriormente se aplicó artificialmente el ruido. Ambos parches son contiguos en la imagen original. Sin embargo, la diferencia entre los valores estimados de a y b es muy alta. Para el primer parche con un nivel de ruido razonable, los valores resultaron ser $a = 0{,}0004, b = 0{,}00013$, mientras que para el segundo parche $a = 0{,}0628, b = -0{,}00102$. No solo a es un valor muy alto, sino que b tiene un valor inválido ya que es negativo. Cabe notar que la estimación del ruido no se hace por parches al generar pares de imágenes, sino que se estima el ruido total. Para el ejemplo mostrado se estimó el error en parches contiguos de una imagen para mostrar las diferencias abruptas en los resultados de la herramienta.

Se realizaron pruebas utilizando únicamente la herramienta y el código de prueba de Foi et al,

[5]Se utilizó la versión 2.32, descargable en `http://www.cs.tut.fi/~foi/sensornoise.html`
[6]`https://github.com/mlvieras/demosaicing-dataset-generator`

Figura 4.4: Un ejemplo de la diferencia entre una imagen de entrada con ruido (a la izquierda) y una sin ruido (a la derecha). Ambas imágenes fueron generadas por el algoritmo de generación de pares de imágenes escrito para este proyecto.

Figura 4.5: Ejemplo de un caso de estimación errónea del ruido de las imágenes. A la izquierda, un parche con un nivel de ruido razonable ($a = 0{,}0004, b = 0{,}00013$). A la derecha, un parche para el cual se estimó una distribución de ruido muy alta ($a = 0{,}0628, b = 0$).

obteniendo los mismos resultados en todos los casos. Esto da a entender que el problema podría estar en la implementación de la herramienta, pero dado que el código fuente no está disponible, fue imposible encontrar el error. Luego de haber invertido un tiempo razonable en buscar una solución, se considera que el problema está fuera del alcance de este proyecto.

Sin embargo, dado que la herramienta para estimar el ruido funciona bien en la mayoría de los casos, se decidió continuar su uso, restringiendo los parámetros a y b obtenidos a un rango razonable ($a = 0,0005$ y $b = 0,0004$).

4.6. Entrenamiento

En esta sección se detalla el proceso realizado para entrenar las dos redes neuronales definidas por las dos arquitecturas expuestas anteriormente.

4.6.1. Preparación del dataset

El proceso de entrenamiento comienza con la preparación del dataset. Como paso inicial, se toman las 500 imágenes del dataset y se dividen en los conjuntos de entrenamiento, validación y prueba. En particular, un 20 % de las imágenes corresponden al conjunto de prueba y un 30 % de las imágenes al conjunto de validación.

Dado que el dataset utilizado contiene pocas imágenes, se decidió aumentar artificialmente la cantidad de ejemplos con recortes aleatorios. En particular, se tomaron 15 recortes cuadrados de tamaño 128 para cada imagen. Esto aumenta considerablemente el tamaño total del dataset a 7500 imágenes. Es importante notar que el aumento fue realizado posterior a la división de las imágenes en los tres conjuntos utilizados. Esto impide que información de la misma imagen base esté presente en dos o más conjuntos, ya que los recortes muy probablemente comparten información entre sí. Como resultado, los conjuntos de entrenamiento, validación y prueba contienen 3750, 2250 y 1500 imágenes respectivamente.

Finalmente, cada conjunto es reordenado de forma aleatoria para evitar que los lotes de entrenamiento contengan imágenes muy similares (muchos recortes de la misma imagen), y así probablemente lograr que el entrenamiento converja más rápidamente.

4.6.2. Proceso integral de entrenamiento

Para inicializar los pesos de la red se utilizó la inicialización de Kaiming [22]. Esta estrategia ha mostrado ser más eficiente que la inicialización de Xavier [23], y además da mejores resultados empíricos, disminuyendo el tiempo de convergencia y la pérdida según lo mostrado por Kaiming et al. [22].

En ambas arquitecturas, cada capa convolucional utiliza normalización en lote. Esta técnica permite mitigar el riesgo de explosión o muerte de gradientes, que se ve fomentado por la profundidad de las redes. Como función de activación, todas las capas convolucionales utilizan *ReLU* (*Rectified Linear Unit*). Esta función de activación tiene la ventaja de ser extremadamente rápida de calcular en comparación con otras, como la función Sigmoide, y además provee un grado de no linealidad suficiente para poder dar la versatilidad suficiente a la red para explorar el espacio de soluciones.

El entrenamiento en sí fue realizado utilizando mini lotes de tamaño 30. Esta técnica intenta facilitar y acelerar la exploración correcta del espacio de soluciones, y en muchos casos acelera la convergencia. Una época de entrenamiento consiste en el procesamiento de todos los mini-lotes provistos por el dataset.

Como es tradicional al entrenar redes convolucionales, la técnica para realizar el entrenamiento es el descenso por gradiente. La idea central es utilizar la información de los ejemplos de entrenamiento para actualizar la función aprendida por la red, descendiendo por el gradiente determinado por la función de pérdida. El algoritmo utilizado para actualizar los pesos de la red es *backpropagation*, o propagación hacia atrás [18][16]. Esto fue explicado en la sección 3.2.

La cantidad de épocas de entrenamiento se especifica previo a comenzarlo. Una vez que la red llega a la cantidad especificada, 2000 épocas en este trabajo, el entrenamiento termina y el modelo es almacenado. Si bien este criterio de parada es suficiente para terminar el entrenamiento, no se toma en cuenta el efecto del sobre ajuste, para evitar esto se decidió analizar el desempeño de la red en el conjunto de validación cada diez épocas. Si el valor de la función de pérdida en el conjunto de validación no disminuye en 500 épocas, se considera que la red está sobre ajustando y se revierte al modelo que dio la menor pérdida en validación. Esto es conocido como parada temprana o *early stopping* en inglés. Este criterio de parada fue de suma utilidad dado que redujo ampliamente el tiempo de entrenamiento necesario.

En todos los entrenamientos realizados la tasa de aprendizaje utilizada fue $0,001$. Además, para el entrenamiento se utilizó el optimizador *ADAM*(*ADAptive Moments*) [38]. ADAM es un método de optimización del entrenamiento que utiliza el concepto de inercia para ajustar la tasa de aprendizaje para cada parámetro a aprender. Para esto calcula en cada época de entrenamiento dos valores de inercia. Sea x el vector de pesos de la red (que contiene todos los pesos que la red aprende), sea dx la derivada de x (calculada utilizando el algoritmo de propagación hacia atrás). El primer valor de inercia depende del histórico de gradientes y se calcula manteniendo una suma ponderada en cada etapa:

$$m_1^{(t+1)} = \beta_1 m_1^{(t)} + (1-\beta_1)\nabla L^{(t)}$$
$$m_2^{(t+1)} = \beta_2 m_2^{(t)} + (1-\beta_2)(\nabla L^{(t)})^2$$

Donde $m_1^{(i)}$ y $m_2^{(i)}$ son los vectores de inercia calculados en la época i de entrenamiento. β_1 y β_2 son los coeficientes de inercia arbitrarios, que usualmente son $0,9$ y $0,999$ respectivamente. $\nabla L^{(t)}$ es el vector de gradientes calculados para cada peso en la época i. La actualización de los pesos de la red en la época $t+1$ puede escribirse como:

$$p^{(t+1)} = p^{(t)} - \eta \frac{m_1^{(t)}}{\sqrt{m_2^{(t)}} + \delta}$$

Donde δ es un número pequeño para asegurar la consistencia numérica (típicamente 1×10^{-7}), η es la tasa de aprendizaje (arbitraria), $p^{(i)}$ es el vector de pesos actualizados en la época i. Uno de los problemas con este enfoque son los valores iniciales de m_1 y m_2. Si se inicializan en un valor çentral"(como cero) generan un sesgo hacia ese valor. De forma de evitar esto se introducen una leve modificación en los valores de inercia utilizados para actualizar los pesos:

$$\hat{m_1}^{(t+1)} = \frac{m_1^{(t+1)}}{1-\beta_1^{t+1}}$$
$$\hat{m_2}^{(t+1)} = \frac{m_2^{(t+1)}}{1-\beta_2^{t+1}}$$

Luego simplemente se intercambian estos valores en la ecuación original de actualización:

$$p^{(t+1)} = p^{(t)} - \eta \frac{\hat{m_1}^{(t)}}{\sqrt{\hat{m_2}^{(t)}} + \delta}$$

Capítulo 5

Experimentación

En esta etapa se realizaron varias tareas, que en particular apuntan a investigar la ventaja de utilizar convoluciones separables por profundidad para el problema de demosaicing y denoising.

La experimentación fue realizada sobre el mismo hardware dispuesto para la implementación de los modelos propuestos. La descripción del mismo puede encontrarse en la sección 4.4.

5.1. Dataset de pruebas

Para realizar las pruebas propuestas se creó un dataset con imágenes tomadas especialmente para este fin. La cámara utilizada fue una Canon EOS T5i, y se tomaron un total de 48 imágenes de tamaño 5208×3476. Es oportuno destacar que ninguna de estas imágenes fue utilizada para entrenar ninguna de las redes utilizadas en este trabajo. Se intentó tomar imágenes representativas de escenas realistas, con componentes que podrían presentar una dificultad a la hora de aplicar demosaicing y denoising. Entre los fenómenos capturados se encuentran: escenas con poca iluminación, componentes de alta frecuencia, desenfoque, texturas con patrones complejos. El dataset fue generado utilizando el método de *Khashabi et al.*, explicado con detalle en la sección 4.5. Algunos ejemplos de imágenes pueden apreciarse en la figura 5.1.

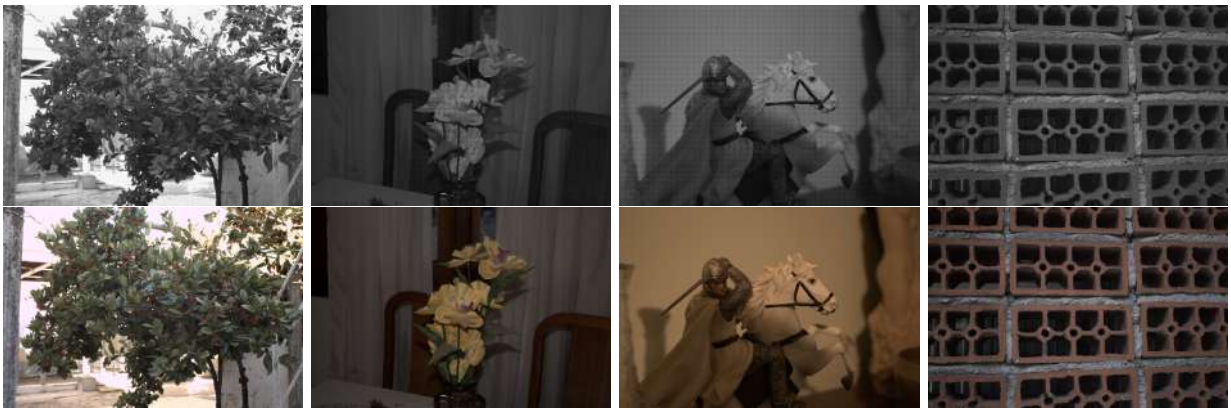
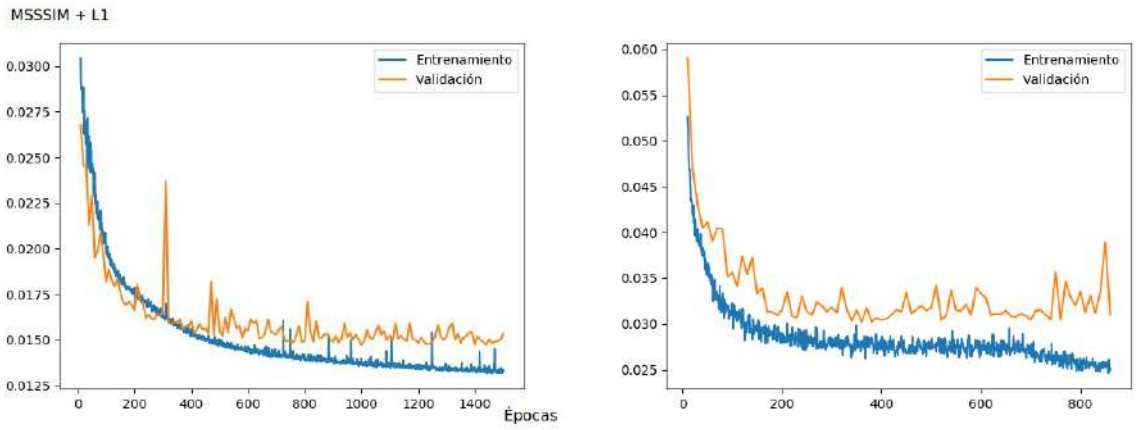


Figura 5.1: Selección de cuatro imágenes del dataset utilizado para la etapa de experimentación. Arriba los ejemplos de entrada, y abajo el ejemplo de salida correspondientes.



(a) (b)

Figura 5.2: Evolución de los resultados de la función de pérdida en ambas arquitecturas durante el entrenamiento. Para cada una de ellas se midió el desempeño en el dataset de entrenamiento y validación, promediando los valores para cada entrada del dataset. (a) Símil Demosaicnet, (b) Mobile Demosaicnet.

Al igual que el conjunto de fotografías utilizado para entrenar las redes, el utilizado para las pruebas se compone de pares de imágenes correspondientes a la entrada y salida de la etapa de demosaicing y denoising del pipeline digital. Dado que se realiza un proceso de subsampleo, cada elemento del dataset tiene un tamaño de 1040×694 (en contraste con los elementos del conjunto de *Khashabi et al*, que son de tamaño 220×132).

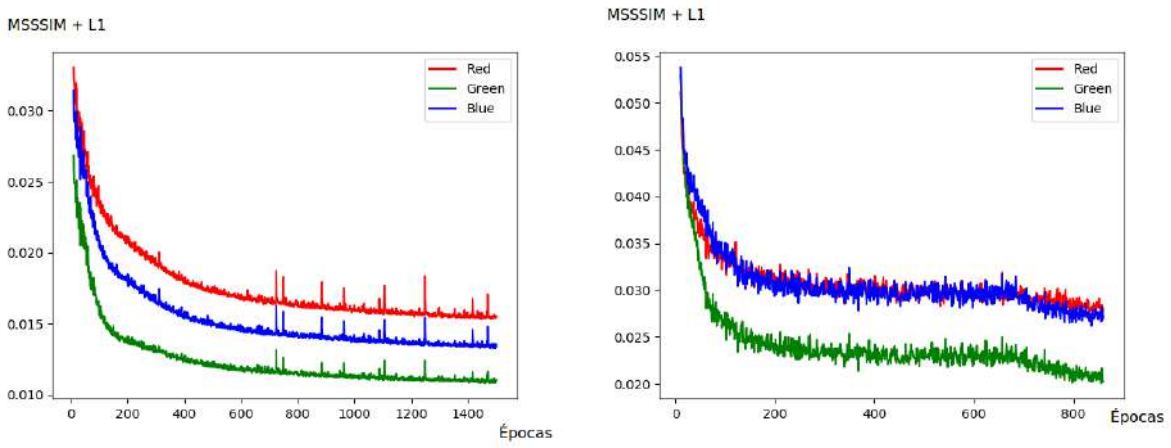
Con el objetivo de analizar el desempeño de las redes neuronales tanto en la tarea de demosaicing como en la tarea conjunta de demosaicing y denoising, se generaron dos conjuntos de imágenes, uno con ruido inicial y otro sin él.

5.2. Resultados

En esta sección se detallan los resultados obtenidos entrenando las redes utilizando una arquitectura de cincuenta capas. Se detallan los aspectos más relevantes del entrenamiento y luego se realiza un análisis de los resultados de forma cualitativa y cuantitativa para imágenes con ruido y sin ruido, así como un estudio del tiempo de ejecución de las redes.

El entrenamiento de ambas redes se inició especificando un máximo de 2000 épocas, límite que ninguna de las redes alcanzó (ambas terminaron de entrenar por *early stopping* antes). Al llegar al final del entrenamiento, ambas redes revirtieron sus pesos a la versión donde el error sobre el conjunto de validación fue menor. Para Símil Demosaicnet, la duración total del entrenamiento fue de 10 horas con 14 minutos, y el mejor valor de error sobre el conjunto de validación se dio en la época 1000. En el caso de Mobile Demosaicnet, el entrenamiento total fue de 4 horas y 35 minutos, revirtiendo a la época 360, donde se dio el mejor desempeño en validación. En la figura 5.2 se muestran los resultados de la función de pérdida para ambas arquitecturas tanto en el dataset de entrenamiento como en el de validación. Los resultados son el promedio del resultado de la función de pérdida en cada dataset.

Observando los resultados puede apreciarse que, si bien presenta algunas fluctuaciones en el desempeño del conjunto de validación, el entrenamiento de Símil Demosaicnet parece converger de una forma más rápida. La evaluación en el conjunto de validación de Mobile Demosaicnet es muy errática, al igual que el desempeño en el conjunto de entrenamiento.



(a) (b)

Figura 5.3: Error promedio por canal en cada época de entrenamiento. (a) el error en el entrenamiento de Símil Demosaicnet, (b) error de entrenamiento en Mobile Demosaicnet.

Arquitectura	P. Entrenamiento	P. Validación
Símil Demosaicnet	0.013558	0.014736
Mobile Demosaicnet	0.026348	0.030225

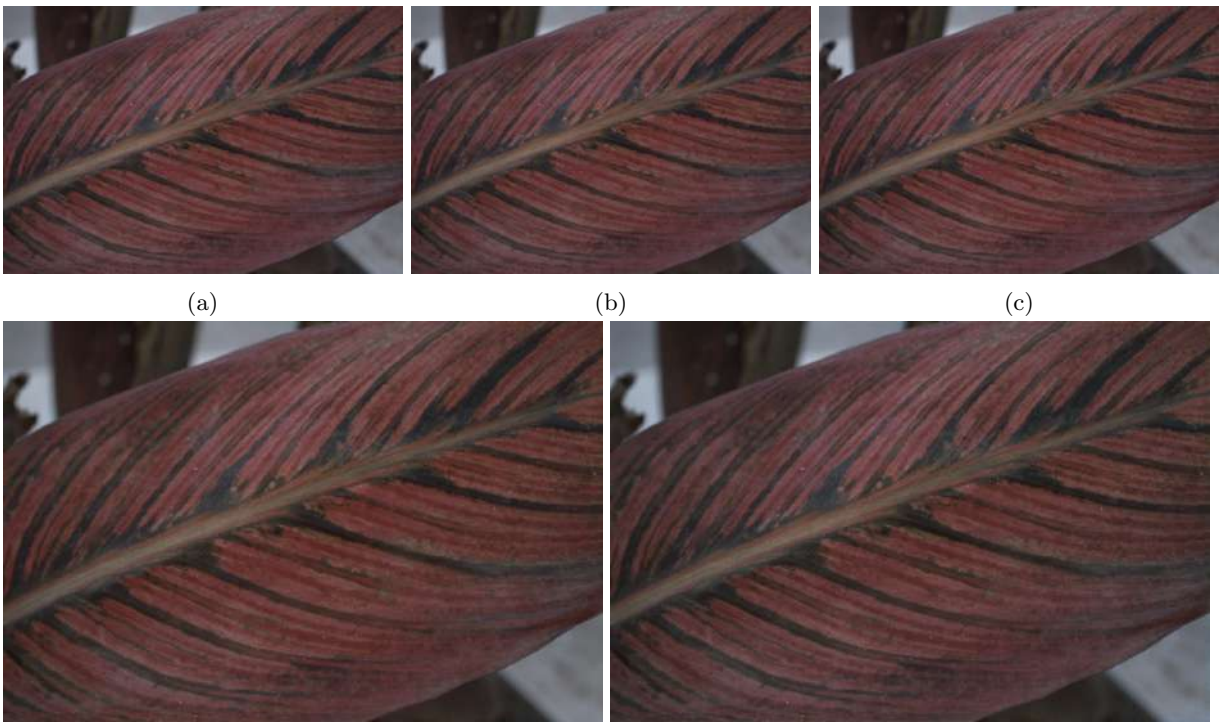
Cuadro 5.1: Resultados de la función de pérdida en el entrenamiento de los dos modelos. Los valores presentados corresponden con la menor pérdida promedio.

Esta diferencia en los resultados puede ser explicada por la posible baja en la capacidad de aprendizaje de las convoluciones separables por profundidad, en comparación con las convoluciones tradicionales. Al disponer de una menor cantidad de pesos para aprender, el espacio de soluciones se reduce en cuanto a dimensiones, facilitando la exploración, pero potencialmente reduciendo la probabilidad de encontrar una solución cercana a la óptima.

Además de la pérdida global se midió el error cometido por la red en cada canal en promedio. Como dato interesante, se dio que el error del canal rojo siempre es el peor, seguido del azul y luego del verde. Que el verde sea el canal en el que se comete menor error tiene sentido, dado que en el patrón de Bayer el verde es el color mejor representado. De cualquier manera resulta interesante que, aunque el canal rojo y el azul están representados equitativamente en el patrón, el canal rojo sea consistentemente el que comete mayor error que el azul en el entrenamiento de Símil Demosaicnet. Sin embargo, en el entrenamiento de Mobile Demosaicnet el error del canal azul y del rojo se mantiene bastante alineado. La figura 5.3 muestra un gráfico con el error promedio por época de cada canal.

En el cuadro 5.1 se muestran los resultados de la función de pérdida tanto en el conjunto de entrenamiento como en el de validación para los dos modelos entrenados. Los resultados mostrados corresponden al mejor promedio de la pérdida del dataset de entrenamiento y validación obtenido durante el entrenamiento.

Una vez entrenados los modelos, se realizaron benchmarks para poder comparar los resultados. Se utilizó el mismo conjunto de imágenes *raw* y se generaron dos conjuntos: uno con ruido y otro sin ruido. A continuación se muestran resultados para ambos conjuntos.



(a) (b) (c)

(d) (e)

Figura 5.4: Una de las imágenes del dataset procesada por los cuatro métodos estudiados en este trabajo. (a) *ground truth*, (b) Malvar, (c) Demosaicnet, (d) Símil Demosaicnet y (e) Mobile Demosaicnet.

5.2.1. Pruebas con imágenes sin ruido

Inicialmente se realizó un benchmark con imágenes sin ruido. En la figura 5.4 puede apreciarse los resultados para cuatro de las imágenes del dataset de benchmark. De arriba hacia abajo, el orden es el siguiente: *ground truth*, resultado de Demosaicnet, resultado de Malvar y por último los resultados para la Símil Demosaicnet y Mobile Demosaicnet.

A simple vista todas las imágenes parecen muy similares. Sin embargo existen algunas diferencias. Para explorar estas diferencias se realizó un análisis cuantitativo y cualitativo de las imágenes.

Análisis Cuantitativo

En el cuadro 5.2 se puede apreciar el promedio de los resultados de cada algoritmo evaluado con varias funciones de pérdida. Inmediatamente puede verse que Demosaicnet supera a todos los algoritmos en todos los casos. Además, puede verse que los resultados obtenidos para nuestras redes son los peores en todos los casos también.

Esto da la pauta de que tanto cualitativa como cuantitativamente el entrenamiento realizado no es suficiente para llegar a un nivel similar al de las estrategias evaluadas. En el cuadro 5.3 se presenta la relación entre el promedio la pérdida de los modelos presentados y el promedio de la pérdida de Demosaicnet en el conjunto de benchmark, además de la pérdida relativa entre los dos modelos.

Un resultado interesante es que en la función de pérdida donde se produce el menor error relativo es en *SSIM*, que no fue la utilizada para entrenar por ninguno de los modelos evaluados. Esto no quiere decir

	Malvar	Demosaicnet	Símil Demosaicnet	Mobile Demosaicnet
L1	7.5×10^{-3}	7.2×10^{-3}	1×10^{-2}	$1,3 \times 10^{-2}$
MSE	1.9×10^{-4}	1.5×10^{-4}	2.3×10^{-4}	$3,2 \times 10^{-4}$
SSIM	7.9×10^{-2}	7×10^{-2}	7.8×10^{-2}	$8,7 \times 10^{-2}$
MSSSIM	6.3×10^{-3}	5.1×10^{-3}	6.41×10^{-3}	$7,5 \times 10^{-3}$
L1 MSSSIM	1.4×10^{-2}	1.2×10^{-2}	1.7×10^{-2}	2×10^{-2}

Cuadro 5.2: Resultados de la función de pérdida en el entrenamiento de los modelos. Los valores presentados corresponden con la menor pérdida promedio.

	Símil Demosaicnet	Mobile Demosaicnet	Comparación interna
L1	42 %	76 %	30 %
MSE	55 %	125 %	40 %
SSIM	12 %	25 %	12 %
MSSSIM	27 %	47 %	17 %
L1 MSSSIM	37 %	64 %	18 %

Cuadro 5.3: Comparación del promedio del error experimentado por los modelos presentados con respecto al cometido por Demosaicnet en el conjunto de benchmark. En la tercer columna se muestra la pérdida relativa de Mobile Demosaicnet con respecto a Símil Demosaicnet.

necesariamente que las imágenes generadas tengan bajo valor de error con *SSIM*, solo significa que el error cometido en ellas es más similar con respecto a otras funciones de error.

Tomando en cuenta la función de pérdida utilizada para entrenar, la red con convoluciones por profundidad separable se desempeña un 18 % peor en promedio. Si bien es un porcentaje alto, no parece manifestarse de forma notoria en el análisis cualitativo. De cualquier manera, es una pérdida de eficacia bastante mayor a la percibida en la tarea de reconocimiento de imágenes implementada por Mobilenet [28]. Naturalmente, la diferencia entre ambos problemas (reconocimiento de imágenes y demosaicing y denoising es bastante grande, lo que podría indicar que las convoluciones separables no son tan buenas en tareas como *demosaicing*. Sin embargo, para llegar a tal conclusión se deberían entrenar múltiples modelos, para lo cual se necesita poder computacional mayor al disponible para este proyecto.

Análisis Cualitativo

El objetivo del análisis cualitativo es determinar si las imágenes resultado del benchmark son razonables desde el punto de vista de la percepción humana. En este marco, y para la tarea de demosaicing y denoising, se considera que una imagen es aceptable siempre y cuando los colores obtenidos se correspondan con la realidad de forma prácticamente imperceptible (lo que en la escena es azul debería ser de una tonalidad azul similar en la imagen resultante). Además no deberían existir artefactos notorios, como por ejemplo *aliasing*, ni aparición de elementos que no se encuentran en la escena (por ejemplo parches negros en una zona con colores claros). En este caso las imágenes de *ground truth* se tomarán como representantes de la escena original, por lo que todas las imágenes resultado se compararán con las mismas.

En la mayor parte de las imágenes procesadas se encontró que todas las estrategias probadas retornaban un resultado razonable a grandes rasgos. A continuación se presentan algunos artefactos y errores encontrados en algunos ejemplos.

En la figura 5.5 puede observarse un ejemplo de un artefacto encontrado. En la imagen central (resultado del modelo similar a Demosaicnet) puede verse un cuadriculado en la cara interna del balde. La aparición de este artefacto puede estar relacionado con el hecho de que el canal rojo es el que comete el mayor error durante el entrenamiento, y el balde tiene un componente fuerte en este canal.

(a) (b) (c)

Figura 5.5: Ejemplo de un artefacto de cuadriculado generado por la arquitectura similar a Demosaicnet. (a) *ground truth*, (b) resultado de Símil Demosacinet, (c) resultado de Mobile Demosaicnet. Es interesante notar que para la arquitectura con convoluciones separables el artefacto no ocurre.

Figura 5.6: Ejemplo del manejo de la oscuridad por parte de los modelos. De izquierda a derecha: *ground truth*, resultado de Símil Demosaicnet, resultado de Mobile Demosaicnet. Si bien en la tercer imagen no se producen artefactos, cabe notar que es mucho más oscura que el *ground truth*. El brillo y el contraste de todas las imágenes ha sido ampliado artificialmente (en la misma medida para las tres imágenes) para apreciar mejor las diferencias.

Otro de los desafíos encontrados fue el desempeño de la red en zonas con muy poca iluminación. Si bien ambas redes entrenadas procesan las zonas con iluminación de una forma razonable, se encontró que el procesamiento de zonas oscuras de las imágenes resulta en artefactos que, si bien no son extremadamente evidentes, deben ser notados.

En la figura 5.6 puede verse un ejemplo de estos artefactos. En la figura central, cuya imagen fue procesada por Símil Demosaicnet, puede apreciarse que el parche presenta artefactos diagonales y un mal manejo de las sombras. El resultado de Mobile Demosaicnet es mejor en el sentido de que este artefacto no ocurre, pero sí se da que la imagen es mucho más oscura que la esperada.

Si bien Mobile Demosaicnet se desempeñó mejor en los artefactos mencionados anteriormente que Símil Demosaicnet, sí presentó artefactos muy notorios en otros casos. En la figura 5.7 se muestran dos artefactos generados por la red. Lo interesante de estos artefactos es que no se producen en zonas complejas de la imagen, sino todo lo contrario. Además, parecen ocurrir de independientemente del valor de la señal en el parche.

Figura 5.7: Artefactos muy notorios generados por Mobile Demosaicnet. A la izquierda la imagen entera, a la derecha un acercamiento al artefacto.

Probablemente la causa de estos problemas es que la red no logró aprender bien la tarea en el tiempo de entrenamiento que se le dio. Una de las posibles soluciones a explorar sería aumentar aún más la profundidad de la red, con el objetivo de aumentar la capacidad de aprendizaje.

De las 48 imágenes del dataset de benchmark solo esas dos imágenes presentan artefactos de este tipo. En otras palabras, la red comete este tipo de errores en un 4 % del dataset, lo cual es un porcentaje bastante alto.

Los anteriormente detallados fueron los artefactos y errores más destacables de los resultados obtenidos en el benchmark. El principal problema encontrado es el manejo de la oscuridad en las imágenes, ya que escenas con sombras y poca iluminación son muy comunes. En la figura 5.8 puede apreciarse el manejo de oscuridad por parte de las redes entrenadas y por Demosaicnet. Si bien Demosaicnet presenta una imagen menos ruidosa, no logra resultados apropiados en cuanto a color, ya que se desvía hacia tonalidades más verdosas, con cierto cuadriculado.

En cuanto a la comparación entre las arquitecturas propuestas, no se encontraron diferencias muy grandes entre los resultados. El detalle principal es que el color de las imágenes resultantes de utilizar el modelo con convoluciones separables es un poco menos intenso que el esperado. En la figura 5.9 puede observarse la diferencia en dos parches generados por ambas arquitecturas.

(a) (b)

(c) (d)

Figura 5.8: Manejo de la oscuridad por parte de las redes entrenadas y Demosaicnet. (a) *ground truth*, (b) resultado de Demosaicnet, (c) resultado de Símil Demosaicnet, (d) resultado de Mobile Demosaicnet.

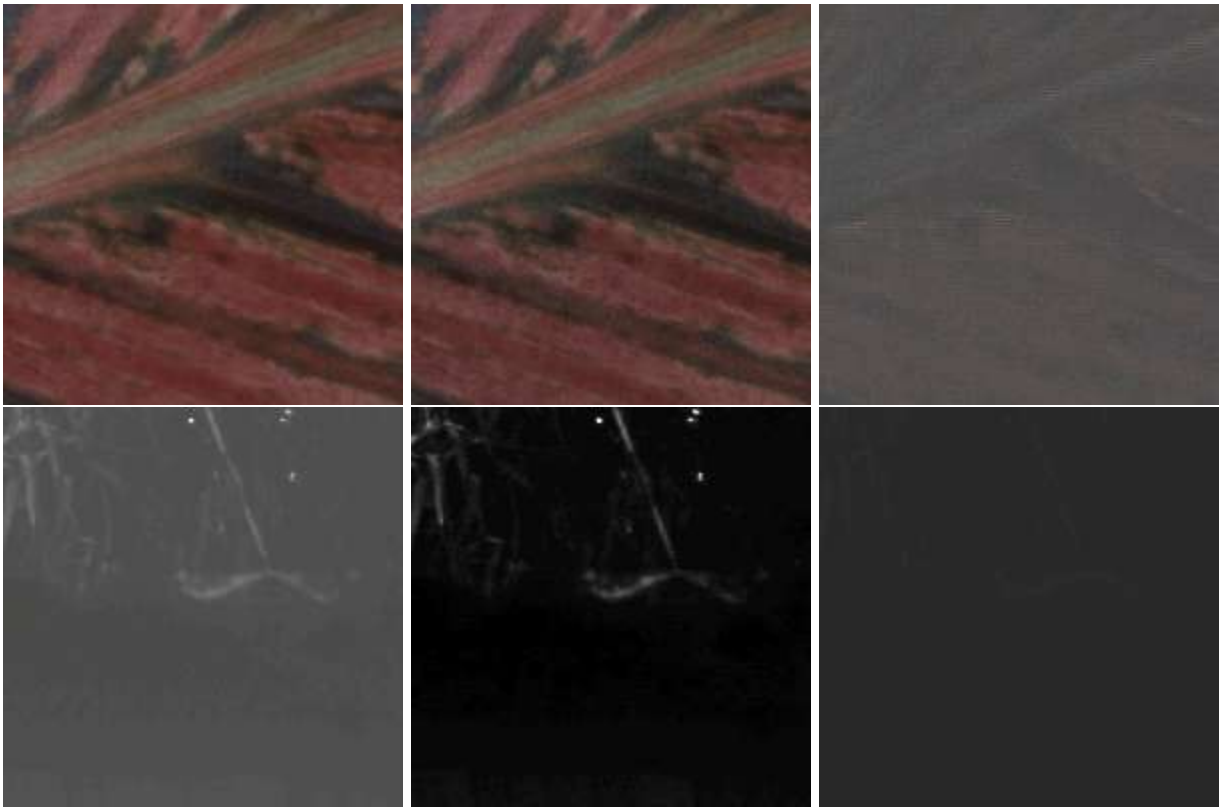


Figura 5.9: Diferencia entre las arquitecturas propuestas. De izquierda a derecha: imagen generada por Símil Demosaicnet, imagen generada por Mobile Demosaicnet, imagen diferencia entre las dos. El cambio de color es ligeramente perceptible en las imágenes bien iluminadas (fila superior). En las imágenes más oscuras la diferencia es más apreciable (fila inferior). El brillo y contraste de las imágenes más oscuras ha sido aumentado artificialmente para apreciar mejor las diferencias (en la misma medida).

En general las redes se comportan de forma muy similar en parches bien iluminados. Este resultado es muy positivo considerando la abismal diferencia en cantidad de pesos que se aprenden en ambos modelos. Sin embargo, en las zonas más oscuras es donde se notan las mayores diferencias entre ambas arquitecturas, lo cual es esperable dado que en escenas oscuras el sensor recibe menos luz, lo que se traduce en menor información descriptiva de lo que se está capturando.

5.2.2. Pruebas con imágenes con ruido

Luego de realizar pruebas con el dataset de benchmark sin ruido, se realizaron pruebas sobre el mismo dataset, pero agregando ruido a las imágenes, tal como se explica en la sección 5.1.

Al igual que en la sección anterior, se realizó un análisis cualitativo y cuantitativo de los resultados.

Análisis Cuantitativo

Al igual que en la sección anterior, en el cuadro 5.4 se presentan los resultados promedio de cada modelo evaluado. Por otro lado en el cuadro 5.5 se muestra la pérdida relativa de los modelos entrenados contra Demosaicnet. En la tercer columna se muestra la pérdida relativa entre Símil Demosaicnet contra Mobile Demosaicnet.

	Malvar	Demosaicnet	Símil Demosaicnet	Mobile Demosaicnet
L1	1.859×10^{-2}	1.678×10^{-2}	1.619×10^{-2}	$1,854 \times 10^{-2}$
MSE	6.135×10^{-4}	5.047×10^{-4}	4.825×10^{-4}	$6,273 \times 10^{-4}$
SSIM	2.397×10^{-1}	2.124×10^{-1}	1.813×10^{-1}	$2,087 \times 10^{-1}$
MSSSIM	2.802×10^{-2}	1.888×10^{-2}	1.828×10^{-2}	$2,154 \times 10^{-2}$
L1 MSSSIM	4.661×10^{-2}	3.566×10^{-2}	3.447×10^{-2}	$4,008 \times 10^{-2}$

Cuadro 5.4: Resultados de la función de pérdida en el entrenamiento de los modelos. Los valores presentados corresponden con la menor pérdida promedio. Es destacable que Símil Demosaicnet supera a Demosaicnet en cuatro de las cinco funciones de error evaluadas.

	Símil Demosaicnet	Mobile Demosaicnet	Comparación interna
L1	-3,5 %	10 %	15 %
MSE	-4,4 %	24 %	30 %
SSIM	-15 %	-2 %	15 %
MSSSIM	-3 %	14 %	18 %
L1 MSSSIM	-3,4 %	12 %	16 %

Cuadro 5.5: Comparación del promedio del error experimentado por los modelos presentados con respecto al cometido por Demosaicnet en el conjunto de benchmark. En la tercer columna se muestra la pérdida relativa de Mobile Demosaicnet con respecto a Símil Demosaicnet.

Nuestro modelo produce resultados marginalmente mejores que Demosaicnet en tres de las cinco funciones de pérdida probadas. Lo más notorio es la mejora de un 13 % con respecto a Demosaicnet cuando la pérdida es *SSIM*. Esto es consistente con las imágenes sin ruido, donde *SSIM* fue la que tuvo mejores resultados. Mobile Demosaicnet tuvo un mejor desempeño también, alejándose un 16 % en *L1 MSSSIM* del estado del arte, en comparación con un 64 % en las imágenes sin ruido.

Es interesante también comparar nuestros modelos entre sí. En este caso Mobile Demosaicnet tuvo mejor desempeño relativo con respecto a las imágenes sin ruido. El peor desempeño fue obtenido en *MSE*, pero se mantuvo a menos de un 20 % de diferencia en las demás métricas. Esto puede indicar que al agregar ruido la diferencia de precisión de los modelos se mantiene.

Análisis Cualitativo

Uno de los resultados más interesantes de esta etapa es que todas las estrategias evaluadas (incluyendo Demosaicnet y Malvar) no se desempeñan muy bien en cuanto a la eliminación de ruido. En todos los casos se nota la presencia de ruido en la imagen final. Si bien de Malvar es esperable que no funcione bien en presencia de ruido, se esperaba que Demosaicnet hiciera un mejor trabajo.

En la figura 5.10 se muestra un ejemplo de los resultados para todas las estrategias evaluadas. Rápidamente puede apreciarse que todas conservan cierto ruido en el resultado.

La diferencia de estos resultados con el *ground truth* es notable en la mayoría de los casos, y no se requiere un análisis muy profundo para darse cuenta que las imágenes son ruidosas. Si bien es esperable que Malvar se desempeñe mal en cuanto a ruido, no fue esperado el bajo rendimiento de Demosaicnet.

Una de las posibles explicaciones del bajo nivel de éxito de todas las estrategias en eliminar el ruido es que el dataset fue generado con demasiado ruido, al menos comparado con el dataset contra el que se entrenó. Si bien los parámetros que determinan el ruido son dinámicos (dependen de cada imagen), se utilizaron cotas superiores para estos parámetros relativamente bajas (explicados en la sección 4.5.6).

Si bien Malvar se presenta como una opción bastante viable para la tarea de *demosaicing* por sí

(a) (b) (c)

(d) (e)

Figura 5.10: Manifestación del ruido en imágenes resultado de todas las estrategias. (a) *ground truth*, (b) resultado de Demosaicnet, (c) resultado de Malvar, (d) resultado de Símil Demosaicnet, (e) resultado de Mobile Demosaicnet. Notar como en todos los casos el ruido es notable (en comparación con la imagen objetivo), particularmente en las zonas más oscuras de la imagen.

Algoritmo	Promedio CPU(s)	Promedio GPU(s)
Malvar	720,80	N/A
Demosaicnet	18971,04	N/A
Símil Demosaicnet	2977,89	459,95
Mobile Demosaicnet	2977,46	535,85

Cuadro 5.6: Tiempo de ejecución promedio de los algoritmos evaluados en el dataset de benchmark.

sola, se desempeña particularmente mal en las imágenes con ruido, muy especialmente en imágenes con poca iluminación (y por ende más ruido). Este resultado es esperable, dado que Malvar utiliza la información local asumiendo que el valor del canal que se está estimando no es independiente del valor de sus vecinos. Esta suposición, al agregar ruido, se convierte en no válida, dado que el ruido en cada píxel es independiente del resto.

En cuanto a los modelos entrenados, cabe destacar que Mobile Demosaicnet sigue presentando artefactos en algunas imágenes resultado. Estos son idénticos a los generados en las imágenes sin ruido, y aparecen en una proporción muy similar.

5.2.3. Tiempo de ejecución

Para cada una de las arquitecturas se midió el tiempo de ejecución en el dataset de benchmark y se recabaron estadísticas, tanto para ejecución en GPU como en CPU. Fue necesario medir el tiempo en ambas modalidades dado que el código de Demosaicnet no soporta la versión de CUDA utilizada, y además Malvar siempre corre en CPU. En el cuadro 5.6 puede apreciarse el promedio de tiempo de ejecución de los algoritmos evaluados sobre el dataset de benchmark.

Según los resultados del cuadro, Malvar es la técnica más rápida en CPU, lo cual tiene sentido dado que es la menos compleja por lejos. Demosaicnet es, por otro lado, el algoritmo que tarda más en procesar las imágenes, mientras que las redes presentadas en este informe tienen un tiempo promedio de aproximadamente 3 segundos.

En GPU ambas arquitecturas presentadas en este informe tardan medio segundo en procesar una imagen en promedio. Se estima que el tiempo necesario para ejecutar la red con convoluciones separables por profundidad sería mucho menor si existiera soporte adecuado.

Comparativamente, los modelos presentados en este informe son capaces de procesar imágenes a una velocidad aproximadamente seis veces mayor a Demosaicnet.

Capítulo 6

Conclusiones

El objetivo principal del presente proyecto fue investigar la posibilidad de encontrar un modelo de aprendizaje profundo que pudiera aprender las tareas de *demosaicing* y *denoising*, y ser ejecutado de forma rápida, comprometiendo lo menos posible la precisión.

Cualitativamente las imágenes obtenidas presentan buenos resultados exceptuando como se menciono anteriormente algunos artefactos, sobretodo en la red de Mobile Demosaicnet.

Cuantitativamente se puede observar que Símil Demosaicnet presenta mejores resultados que Mobile Demosaicnet tanto en las imágenes sin ruido alcanzando un máximo de %40 de mejora en MSE y un mínimo de %12 en SSIM, como en las imágenes con ruido alcanzando un máximo de %30 en MSE y un mínimo de %15 en L1 y SSIM. Mientras que por otro lado, a pesar de que en la imágenes sin ruido Demosaicnet presenta mejores resultados que Símil Demosaicnet con un máximo de mejora de %55 en MSE y un mínimo de %12 en SSIM, en las imágenes con ruido es Simil Demosaicnet la que presenta mejores resultados alcanzando una diferencia de mejora de %15 en SSIM y un mínimo de %3 en MSSIM.

Por ultimo, destacar la publicación en un repositorio publico del pipeline para generar pares de imágenes para utilizar en el entrenamiento, como fue explicado en la sección 4.5.6.

6.1. Trabajo futuro

Sin perjuicio de lo antedicho, el proyecto presenta muchas oportunidades de mejora. La principal es la reducción de los artefactos generados por el modelo con convoluciones separables.

Una de las principales limitaciones de este proyecto fue el acceso a hardware que permitiera aumentar el poder computacional utilizado para entrenar. Dentro del trabajo futuro propuesto se encuentra la posibilidad de realizar entrenamientos de redes más complejas, lo que llevaría más tiempo en el hardware utilizado actualmente.

Acceso a mayor recurso de cómputo permitiría, también, el análisis del efecto de cambios en ciertos hiper parámetros de los modelos, como son la función de pérdida utilizada, el optimizador, la función de activación de las capas convolucionales y la arquitectura.

Otra posibilidad de mejora se encuentra en el código utilizado para generar datasets de benchmark y de entrenamiento. Uno de los objetivos principales sería la eliminación de la dependencia más compleja: el código de *Foi et al* [10], implementando manualmente el algoritmo de estimación del ruido de imágenes,

y por ende eliminando por completo la dependencia con Matlab y con el código ofuscado publicado por los investigadores.

Alineado al algoritmo de generación de datasets, otra oportunidad de mejora encontrada es la generación de un dataset mucho más grande de training, basado en imágenes de cámaras modernas, como la utilizada en este proyecto. Se cree sería de utilidad generar un conjunto de imágenes para el entrenamiento de *demosaicing* y *denoising* con una cantidad de imágenes mucho mayor que las tomadas en este proyecto. Existen varias formas de atacar este problema. Una de ellas es tomar manualmente imágenes con diversas cámaras digitales, intentando asegurar buena diversidad de imágenes. Otra opción es encontrar un dataset de imágenes en formato *raw* de cámaras modernas ya existente y realizar las adaptaciones necesarias para generar un dataset para *demosaicing* y *denoising* a partir del mismo.

Desde el punto de vista de eficiencia y eficacia de los modelos, es de interés probar otras alternativas para disminuir el tiempo de ejecución de los algoritmos, comprometiendo lo menos posible la precisión. Entre ellas se encuentra la arquitectura utilizada por la versión *2.0* de Mobilenet [39], publicada en 2019 por investigadores de Google.

Bibliografía

[1] H. C. Lucas and J. M. Goh, "Disruptive technology: How kodak missed the digital photography revolution," *The Journal of Strategic Information Systems*, vol. 18, no. 1, pp. 46 – 55, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0963868709000043

[2] W. Kao, S. Wang, L. Chen, and S. Lin, "Design considerations of color image processing pipeline for digital cameras," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 4, pp. 1144–1152, Nov 2006.

[3] R. Ramanath, W. E. Snyder, Y. Yoo, and M. S. Drew, "Color image processing pipeline," *IEEE Signal Processing Magazine*, vol. 22, no. 1, pp. 34–43, Jan 2005.

[4] D. Khashabi, S. Nowozin, J. Jancsary, and A. W. Fitzgibbon, "Joint demosaicing and denoising via learned nonparametric random fields," *IEEE Transactions on Image Processing*, vol. 23, no. 12, pp. 4968–4981, Dec 2014.

[5] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, "Deep joint demosaicking and denoising," *ACM Transactions on Graphics*, vol. 35, pp. 1–12, 11 2016.

[6] E. Schwartz, R. Giryes, and A. M. Bronstein, "Deepisp: Toward learning an end-to-end image processing pipeline," *IEEE Transactions on Image Processing*, vol. 28, no. 2, pp. 912–923, Feb 2019.

[7] S. J. Sasson, "Electronic still camera," U.S. Patent 798,956, 12 26, 1978.

[8] M. Levoy, "Cs178: Digital photography: Photons and sensors." [Online]. Available: http://graphics.stanford.edu/courses/cs178-11/

[9] B. E. Bayer, "Color imaging array," U.S. Patent 555,477, 07 20, 1976.

[10] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian, "Practical poissonian-gaussian noise modeling and fitting for single-image raw-data," *IEEE Transactions on Image Processing*, vol. 17, no. 10, pp. 1737–1754, Oct 2008.

[11] M. Levoy, "Cmu 15-869: Graphics and imaging architectures: A camera's image processing pipeline." [Online]. Available: http://www.cs.cmu.edu/afs/cs/academic/class/15869-f11/www/

[12] H. S. Malvar, L. He, and R. Cutler, "High-quality linear interpolation for demosaicing of bayer-patterned color images," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, May 2004, pp. iii–485.

[13] P. Getreuer, "Malvar-He-Cutler Linear Image Demosaicking," *Image Processing On Line*, vol. 1, pp. 83–89, 2011.

[14] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain [j]," *Psychol. Review*, vol. 65, pp. 386 – 408, 12 1958.

[15] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, Jun 1976. [Online]. Available: https://doi.org/10.1007/BF01931367

[16] P. Werbos and P. J. (Paul John, "Beyond regression : new tools for prediction and analysis in the behavioral sciences /," 01 1974.

[17] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System Modeling and Optimization*, R. F. Drenick and F. Kozin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 762–770.

[18] Y. Lecun, "A theoretical framework for back-propagation," in *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. Morgan Kaufmann, 1988, pp. 21–28.

[19] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *MCSS*, vol. 2, pp. 303–314, 1989.

[20] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998, pp. 9–50. [Online]. Available: http://dl.acm.org/citation.cfm?id=645754.668382

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 807–814. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[23] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[24] J. Qiao, S. Li, and W. Li, "Mutual information based weight initialization method for sigmoidal feedforward neural networks," *Neurocomputing*, vol. 207, pp. 676 – 683, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121630491X

[25] J. Y. Yam and T. W. Chow, "A weight initialization method for improving training speed in feedforward neural network," *Neurocomputing*, vol. 30, no. 1, pp. 219 – 232, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231299001277

[26] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 253–256.

[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[29] Z. Wang, A. Bovik, H. Rahim Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, pp. 600 – 612, 05 2004.

[30] Z. Wang, E. Simoncelli, and A. Bovik, "Multiscale structural similarity for image quality assessment," vol. 2, 12 2003, pp. 1398 – 1402 Vol.2.

[31] C. Chen, Q. Chen, J. Xu, and V. Koltun, "Learning to see in the dark," *CoRR*, vol. abs/1805.01934, 2018. [Online]. Available: http://arxiv.org/abs/1805.01934

[32] N. Syu, Y. Chen, and Y. Chuang, "Learning deep convolutional networks for demosaicing," *CoRR*, vol. abs/1802.03769, 2018. [Online]. Available: http://arxiv.org/abs/1802.03769

[33] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 47–57, March 2017.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[35] "Cs231n: Convolutional neural networks for visual recognition." [Online]. Available: http://cs231n.stanford.edu/

[36] A. Graps, "An introduction to wavelets," *IEEE Computational Science and Engineering*, vol. 2, no. 2, pp. 50–61, Summer 1995.

[37] I. Daubechies, *Ten Lectures on Wavelets*, ser. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1992. [Online]. Available: https://books.google.com.uy/books?id=cwdjT3CWY1kC

[38] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[39] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381