TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Deconvolución ciega de imágenes usando Deep Image Prior

Autor

José Antonio Torres Coca

Directores

Rafael Molina Soriano

Francisco Miguel Castro Macías

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 24 de junio de 2024

Deconvolución ciega de imágenes usando Deep Image Prior

Autor
José Antonio Torres Coca

Directores
Rafael Molina Soriano
Francisco Miguel Castro Macías

Deconvolución ciega de imágenes usando Deep Image Prior

José Antonio Torres Coca

Resumen

En este trabajo se aborda la restauración de imágenes borrosas y ruidosas mediante técnicas avanzadas de aprendizaje profundo. Se propone un método que combina una red de desemborronamiento dinámico preentrenada y Deep Image Prior.

Inicialmente, revisaremos el problema de la deconvolución de imágenes explorando los enfoques clásicos y modernos empleados para abordarlo. También analizaremos los fundamentos del aprendizaje profundo y de Deep Image Prior, proporcionando una base sólida para entender el método propuesto.

En segundo lugar, presentamos el modelo utilizado: Dynamic Deblurring Network (DDNet). Explicamos cómo la pseudoinversa de Moore-Penrose puede proporcionar una estimación inicial de la imagen nítida a partir del operador de convolución de emborronamiento. Luego, el problema de deconvolución se aborda mediante la utilización de una red de filtros dinámicos para eliminar artefactos de deconvolución y preservar los detalles de la imagen.

A continuación, se introduce el concepto de Deep Image Prior, el cual consiste en el entrenamiento de una red neuronal específica para cada imagen que se desea restaurar. Este enfoque permite ajustar los pesos de la red neuronal de manera individualizada para cada imagen, lo que otorga una flexibilidad significativa. Esta flexibilidad es uno de los factores clave que ha llevado a DIP a obtener resultados sobresalientes en la restauración de imágenes. Este método se integra con DDNet para refinar la restauración inicial obtenida mediante DDNet.

Finalmente, evaluamos el método propuesto utilizando un conjunto de imágenes reales emborronadas. Los resultados obtenidos demuestran como la combinación de ambas técnicas puede mejorar significativamente la calidad de las imágenes restauradas.

Todo el código empleado para generar, analizar y visualizar los datos y resultados obtenidos es de libre acceso en el siguiente enlace de Github: `https://github.com/joseantonio2001/CHSG-DDNet-DIP-model`

Blind Image Deconvolution using Deep Image Prior

José Antonio Torres Coca

Keywords: blind image deconvolution, deep learning, neural network, deep image prior.

Abstract

This paper addresses the restoration of blurred and noisy images using advanced deep learning techniques. A method combining a pre-trained dynamic deblur network and Deep Image Prior is proposed.

Initially, we will review the problem of image deconvolution by exploring the classical and modern approaches employed to address it. We will also discuss the fundamentals of deep learning and Deep Image Prior, providing a solid foundation for understanding the proposed method.

Second, we present the model used: Dynamic Deblurring Network (DDNet). We explain how the Moore-Penrose pseudoinverse can provide an initial estimate of the sharp image from the blurring convolution operator. Then, the deconvolution problem is addressed by using a dynamic filter network to remove deconvolution artifacts and preserve image details.

Next, the concept of Deep Image Prior is introduced, which consists of training a specific neural network for each image to be restored. This approach allows the neural network weights to be adjusted individually for each image, which provides significant flexibility. This flexibility is one of the key factors that has led DIP to achieve outstanding results in image restoration. This method integrates with DDNet to refine the initial restoration obtained using DDNet.

Finally, we evaluate the proposed method using a set of real smeared images. The obtained results demonstrate how the combination of both techniques can significantly improve the quality of the restored images.

All the code used to generate, analyze and visualize the data and results obtained is freely available at the following Github link: `https://github.com/joseantonio2001/CHSG-DDNet-DIP-model`

Yo, **José Antonio Torres Coca**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 53914813F, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Antonio Torres Coca

Granada a 24 de junio de 2024.

D. **Rafael Molina Soriano**, Profesor Catedrático del Departamento Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Francisco Miguel Castro Macías**, Profesor contratado FPU en el Departamento Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Deconvolución ciega de imágenes usando Deep Image Prior***, ha sido realizado bajo su supervisión por **José Antonio Torres Coca**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 24 de junio de 2024.

Los directores:

Rafael Molina Soriano **Francisco Miguel Castro Macías**

Agradecimientos

No podría comenzar de otra manera que dedicando unas palabras de agradecimiento a mis padres, Eleuterio y Ana María, por el esfuerzo que han hecho durante todos estos años. Han luchado incansablemente por mi educación y hoy sé que están tan orgullosos de mí como yo lo estoy de ellos. Todo lo que consiga en la vida jamás habría sido posible sin vosotros.

Gracias a Rafael y Francisco, mis tutores, por la cercanía que han mostrado y por haberme guiado durante todo este trabajo. Su apoyo y paciencia han sido esenciales. No puedo pedir una mejor atención que la que vosotros me habéis dado. Gracias a Rafael y Francisco, mis tutores, por la cercanía que han mostrado y por haberme guiado durante todo este trabajo. Su apoyo y paciencia han sido esenciales. No podría haber pedido una mejor atención que la que vosotros me habéis dado.

Agradecer también a todos los familiares, amigos y compañeros de clase, por estar siempre a mi lado, por escucharme y animarme en los peores momentos. Vuestra presencia y apoyo constante han sido una fuente de fortaleza para mí, y estoy eternamente agradecido por teneros en mi vida.

Por último, quería hacer una mención especial a mi abuelo Pepe, mi ángel de la guarda, que desde ahí arriba estará viendo todo lo que he conseguido y debe de estar orgulloso. Aunque el tiempo que compartimos fue breve, su amor y su ejemplo han dejado una huella imborrable en mi vida. Abuelo, este logro es en tu honor.

Índice general

Capítulo 1

Deconvolución Ciega de Imágenes

La deconvolución ciega de imágenes es un área fundamental en el procesamiento y restauración de imágenes. Este proceso es esencial en diversas aplicaciones, como la mejora de imágenes médicas, microscopía y fotografía forense, donde la calidad y la precisión de las imágenes restauradas son críticas para su análisis e interpretación.

En este capítulo definiremos la deconvolución ciega de imágenes, los desafíos que presenta y exploramos tanto los enfoques clásicos como modernos para abordar estos desafíos.

1.1. Definición y Problemas Asociados

La deconvolución ciega de imágenes (BID, por sus siglas en inglés de Blind Deconvolution of Images) se refiere a los modelos empleados para la restauración de imágenes degradadas sin conocimiento previo del filtro de degradación utilizado. Este proceso se describe matemáticamente a través del modelo de convolución:

$$y = h * x + n \tag{1.1}$$

donde y es la imagen degradada, x es la imagen original que se desea recuperar, h es el filtro de degradación o núcleo de convolución (PSF, Point Spread Function), n representa el ruido aditivo, normalmente asumido como ruido gaussiano y $*$ denota la operación de convolución. Si h es conocido, el problema se denomina **Deconvolución no ciega de imágenes (NBID)**. De lo contrario, se conoce como **Deconvolución ciega de imágenes**.

El objetivo de la deconvolución ciega es estimar tanto la imagen original x como el filtro de degradación h partiendo de la imagen observada y. Debido

a su naturaleza inversa, la falta de información completa sobre h, la presencia de ruido en la imagen observada y la posibilidad de múltiples soluciones, este problema resulta complejo.

BID se presenta como un problema *ill-posed* (mal definido), ya que no siempre tiene una solución única o estable. Esto significa que pequeñas perturbaciones en los datos observados pueden causar grandes variaciones en las estimaciones de la imagen original x y el filtro de degradación h. Además el ruido n presente en la imagen observada puede amplificarse durante el proceso de deconvolución, resultando en artefactos y distorsiones en la imagen restaurada. La separación efectiva del ruido del contenido de la imagen es, por lo tanto, un reto crucial para lograr una restauración de alta calidad.

Otro aspecto crítico es la precisión en la estimación del filtro de degradación h, ya que un modelo incorrecto o impreciso del PSF puede llevar a una restauración ineficaz de la imagen original. Factores como el movimiento de la cámara, el desenfoque de la lente o las condiciones de iluminación pueden afectar la variabilidad del PSF. Además, los métodos de deconvolución ciega a menudo requieren técnicas iterativas y optimización no lineal, que son computacionalmente costosas. Balancear la calidad de la imagen restaurada y la eficiencia computacional es una consideración importante en la práctica.

La evaluación objetiva de los resultados de la deconvolución ciega también se presenta como un reto, ya que la imagen original x no está disponible para la comparación directa. Se utilizan métricas como el índice de similitud estructural (SSIM) y la relación señal-ruido de pico (PSNR) para evaluar la calidad de la imagen restaurada, aunque estas métricas también tienen limitaciones.

En conclusión, la deconvolución ciega de imágenes enfrenta desafíos significativos relacionados con la falta de información precisa del filtro de degradación, la presencia de ruido y la complejidad computacional. En la siguiente sección, exploraremos los enfoques clásicos y modernos que se han desarrollado para abordar estos desafíos, detallando tanto los métodos deterministas y estocásticos tradicionales como las innovaciones recientes basadas en aprendizaje profundo.

1.2. Enfoques clásicos y modernos

1.2.1. Enfoques clásicos

Los enfoques clásicos para BID pueden clasificarse en métodos deterministas y estocásticos. Estas técnicas analíticas suelen basarse en un proceso iterativo para estimar el núcleo de desenfoque y la imagen nítida, y se dividen normalmente en dos fases. La primera se centra en obtener una estimación

precisa del núcleo de desenfoque extrayendo las características más destacadas de la imagen; En la segunda fase se utiliza el núcleo estimado para obtener la imagen nítida. Los errores cometidos durante la fase inicial de estimación del desenfoque pueden causar problemas importantes, como timbres y distorsiones. Esto subraya la necesidad de la segunda fase que permite combatir las imprecisiones del núcleo mediante métodos NBID.

Métodos Deterministas

Los métodos deterministas implican la definición de un criterio de optimización, como minimizar la norma L2 (o Euclidiana) del error $||y - Hx||^2$, donde y es la imagen observada, H es la matriz asociada al núcleo de desenfoque y x la imagen nítida, con el fin de minimizar la discrepancia entre la imagen borrosa observada y y la imagen nítida convolucionada con la matriz de desenfoque estimado, Hx. Para garantizar la robustez de la solución se emplean técnicas de regularización, como:

- Regularización de Variación Total [1]: Este enfoque fomenta la suavidad local en la imagen penalizando la variación total de esta, eliminando detalles no deseados mientras se preservan otros importantes como los bordes.

Métodos Estocásticos

Los métodos estocásticos tratan las incógnitas como variables aleatorias y emplean marcos probabilísticos para inferir las soluciones más probables. Los enfoques comunes incluyen:

- Máxima Verosimilitud [2]: define una función de verosimilitud basada en la probabilidad de observar los datos de la imagen desenfocada dados los parámetros (imagen original y núcleo de desenfoque) y luego optimizan esa función para encontrar los valores de los parámetros que maximizan la verosimilitud.
- Máximo a Posteriori [3]: mejora el método ML incorporando información previa sobre la imagen y núcleo de desenfoque. Maximiza la distribución a posteriori, que combina la verosimilitud de los datos observados con distribuciones previas sobre los parámetros.
- Métodos Bayesianos Jerárquicos [3] [4]: extienden el enfoque MAP introduciendo múltiples niveles de parámetros e hiperparámetros, creando un marco probabilístico más flexible. Este enfoque modela no solo la imagen y el núcleo de desenfoque, sino también sus distribuciones subyacentes, y utiliza técnicas como *MCMC* (Markov Chain Monte

Carlo) o *inferencia variacional* para inferir toda la distribución posterior.

1.2.2. Enfoques Modernos

La llegada del Deep Learning ha revolucionado la restauración de imágenes al aprovechar el poder de las redes neuronales para aprender mapeos complejos de imágenes borrosas a imágenes nítidas. Los métodos modernos se pueden categorizar en enfoques puramente de aprendizaje profundo y enfoques híbridos que combinan el aprendizaje profundo con métodos analíticos.

Métodos basados en Deep Learning

Estos métodos se basan exclusivamente en el uso de redes neuronales convolucionales (CNN) para abordar distintos tipos de desenfoque:

- **CNNs Residuales Multi-escala** [5]: Estas redes procesan la imagen borrosa a múltiples escalas. Esto es importante porque las imágenes pueden tener características importantes a diferentes niveles de detalle (escalas). Por lo tanto, estas redes están diseñadas para capturar y procesar estas características a múltiples resoluciones o escalas dentro de la misma arquitectura.
- **Redes Generativas Adversariales** [6] [7]: Este tipo de arquitectura consisten en dos redes neuronales que compiten entre sí. La primera red, denominada generadora aprende a generar imágenes de alta calidad a partir de imágenes borrosas, mientras que la segunda red, llamada discriminadora evalúa la autenticidad de las imágenes generadas. Ambas redes compiten durante el proceso de entrenamiento para mejorar sus habilidades, resultando en la restauración de imágenes cada vez más realistas por parte del generador.
- **Deep Image Prior** [8]: Este enfoque aprovecha la estructura de una red neuronal en sí misma como una distribución a priori, desafiando la suposición común de que la restauración efectiva de imágenes depende de redes neuronales convolucionales profundas (CNNs) preentrenadas en grandes conjuntos de datos. En lugar de eso, este método propone explorar la capacidad inherente de la estructura de las CNNs para capturar estadísticas de imágenes y realizar restauraciones efectivas sin necesidad de entrenamiento previo en grandes conjuntos de datos.

Aunque estos métodos pueden resultar efectivos en gran número de casos,

la diversidad de tipos de desenfoque y niveles de ruido puede plantear un desafío significativo.

Enfoques Híbridos

La combinación de métodos analíticos y de aprendizaje profundo ha demostrado ser prometedora para abordar las limitaciones de ambos enfoques por separado, aprovechando así la flexibilidad de los enfoques analíticos y la efectividad de las DNN. Entre las estrategias más populares encontramos Técnicas de División de Variables y Modelos de Refinamiento de la Estimación Inicial.

Técnicas de División de Variables Los enfoques híbridos a menudo utilizan técnicas de división de variables como el Método de la Dirección Alterna de los Multiplicadores (ADMM) [9] y la División Cuadrática Media (HQS) [10] para descomponer el problema de deconvolución en subproblemas más manejables que se resuelven alternativamente utilizando métodos analíticos y de DL. El enfoque general implica:

- Subproblema A (Componente Analítico): Recuperación regularizada de la imagen latente, a menudo resuelta utilizando la transformada de Fourier.
- Subproblema B (Componente de DL): Reducción de ruido o la eliminación de artefactos utilizando una red neuronal. La red está diseñada para manejar los tipos específicos de ruido y artefactos introducidos por la deconvolución analítica inicial.

Al resolver iterativamente estos subproblemas, el enfoque híbrido puede refinar eficazmente tanto la imagen como la estimación del desenfoque, conduciendo a una calidad de restauración superior. Sin embargo, la naturaleza iterativa puede resultar en altos costos computacionales

Modelos de Refinamiento de la Estimación Inicial En estos enfoques, se obtiene una estimación inicial de la imagen latente utilizando un método analítico, que luego es refinada por un modelo de DL. El proceso típicamente implica:

- Estimación Inicial Analítica: Se realiza una o varias estimaciones iniciales de la imagen reconstruida utilizando métodos como el filtrado de Wiener, la regularización de Tikhonov [11] u otras técnicas clásicas. Este paso reduce la complejidad del desenfoque y el ruido que el modelo de DL subsecuente necesita procesar.

- Refinamiento: La estimación inicial es refinada por un modelo de DL diseñado para corregir artefactos y mejorar la calidad de la imagen, produciendo la estimación final.

Este enfoque de dos fases reduce la variabilidad de las degradaciones que el modelo de DL debe manejar, mejorando la efectividad y la robustez general del proceso de deconvolución.

Capítulo 2

Aprendizaje Profundo

El **Aprendizaje Profundo** (DL, por sus siglas en inglés Deep Learning) constituye un campo interdisciplinario que combina principios matemáticos, estadísticos y de ciencias de la computación, con el propósito de construir modelos con la capacidad de adquirir representaciones complejas a partir de conjuntos de datos, entendiendo y capturando estructuras intrincadas y características detalladas de estos. Se basa en el uso de Redes Neuronales Artificiales (Artificial, Neural Networks, ANNs) para aprender patrones complejos a partir de un conjunto de ejemplos de entrenamiento y realizar predicciones.

En este capítulo, profundizaremos en los conceptos y técnicas fundamentales de DL basándonos en los libros [12], [13] y los artículos científicos [14], [15]. Comenzaremos explorando las redes neuronales, detallando los distintos tipos de capas que se utilizan dentro de estas redes y analizaremos diferentes arquitecturas. A continuación, examinaremos su proceso de entrenamiento, centrándonos en los componentes esenciales, como la función de pérdida, el principio del gradiente descendente y el método de *backpropagation*. Además, presentaremos Pytorch, un popular framework para aprendizaje profundo que utilizaremos en este trabajo. Cubriremos sus bloques de construcción fundamentales como los tensores, módulos Pytorch y Funciones de pérdida y backward.

2.1. Redes Neuronales

Una **red neuronal** es un modelo de aprendizaje automático que toma decisiones mediante una estructura de capas de procesamiento que trabajan conjuntamente para identificar patrones, sopesar opciones y llegar a conclusiones.

2.1.1. Capas: Definición y ejemplos

Las capas en redes neuronales modernas se definen como operaciones de tensores complejas y compuestas. Estas capas a menudo incorporan parámetros entrenables y presentan a un nivel de granularidad conveniente para diseñar y describir modelos profundos de gran tamaño. Esto significa que, al organizar las redes en capas, es posible manejar y comprender modelos complejos de manera más sencilla y ordenada, facilitando tanto su construcción como su explicación.

Las **capas lineales** son un tipo fundamental de capa utilizada en las redes neuronales. Éstas realizan transformaciones lineales donde se toma un tensor de entrada y se convierte en un tensor de salida mediante una operación lineal que implica pesos y sesgos. Entre las capas lineales más populares destacamos *Capas Completamente Conectadas* o (Fully Connected) y *Capas Convolucionales*.

La capa lineal más básica son las capas completamente conectadas, parametrizada por una matriz de pesos y un vector de sesgo. Estas conectan cada neurona de una capa con todas las neuronas de la siguiente capa. La salida de una capa completamente conectada se calcula como la multiplicación matricial del tensor de entrada por el tensor de peso, seguida de la suma del tensor de sesgo.

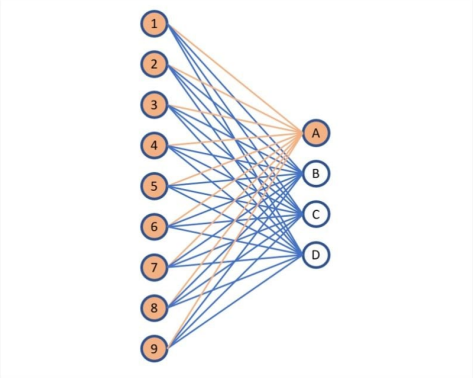


Figura 2.1: Representación de conexiones de neuronaes en Fully Connected Layer.

Las capas convolucionales requieren de tres componentes principales: datos de entrada, un filtro (kernel) y un mapa de características. Supongamos que la entrada es una imagen en color, que está formada por una matriz de píxeles en 3D (altura, ancho y profundidad). El kernel se moverá recorriendo las diferentes submatrices de la imagen.

El kernel es una matriz de pesos 2D. Este se aplica en la distintas áreas de la imagen calculando para cada área un producto escalar entre los píxeles

de entrada y el filtro. Este producto escalar se introduce a continuación en una matriz de salida. El filtro continua desplazándose y repitiendo el proceso hasta que haya recorrido toda la imagen. El resultado final de la serie de productos escalares de la entrada y el filtro se conoce como mapa de características.

Si I es la matriz de entrada y K es el filtro (o kernel), la operación de convolución se define como:

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) \cdot K(m, n)$$

donde $I(i+m, j+n)$ es un submatriz de la imagen de entrada, y $K(m, n)$ son los pesos del filtro. La salida de esta operación es un mapa de características.

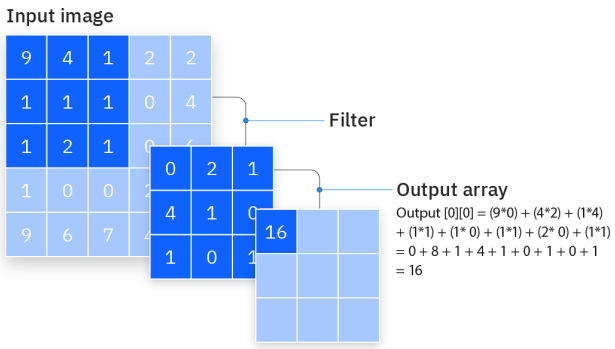


Figura 2.2: Operación de convolución.

Las **capas de agrupación (Pooling)** reducen la dimensionalidad espacial de los mapas de características, simplificando la representación y mejorando la eficiencia computacional. De manera similar a la capa convolucional, la operación de agrupación barre un filtro por toda la entrada, pero con la diferencia de que este filtro no tiene ningún parámetro. Hay dos tipos principales de agrupación:

- Max Pooling: Selecciona el píxel con el valor máximo para enviar a la matriz de salida. Con un tamaño de filtro $p \times p$, se define como:

$$y(i, j) = \max_{0 \leq m,n < p} x(i + m, j + n)$$

  donde x es la entrada, y es la salida, (i, j) es posición donde se almacenará el píxel de valor máximo en la matriz de salida y y (m, n) son los índices dentro del filtro de tamaño $p \times p$, y permite recorrer los valores dentro de una región cuadrada de $p \times p$ en la matriz de entrada x.

- Average Pooling: Calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida. Con un tamaño de filtro $p \times p$, y la entrada es una matriz x, la salida y en la posición $(i,\ j)$ se define como:

$$y(i,j) = \frac{1}{p \times p} \sum_{m=0}^{p-1} \sum_{n=0}^{p-1} x(i+m, j+n)$$

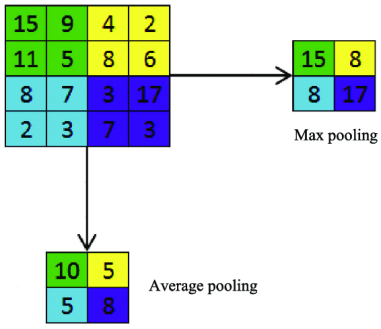


Figura 2.3: Operaciones de Max Pooling y Avg Pooling.

Las **funciones de activación** tienen el propósito de introducir no linealidad en la salida de una capa. Algunas de las más famosas son *ReLU* y *Softmax*.

- Función ReLU (Rectified Linear Unit): Es una función muy popular por su eficiencia computacional y su simplicidad. Dado un elemento x, la función se define como el máximo de ese elemento y 0:

$$ReLU(x) = \text{máx}(0, x)$$

  Informalmente, la funció ReLU retorna 0 para valores negativos y el mismo valor para valores positivos. Es muy utilizada en capas convolucionales y de agrupación (pooling).

- Función Softmax: La función softmax es comúnmente utilizada en redes neuronales para convertir un vector de números reales en una distribución de probabilidad, que asigna probabilidades a cada elemento del vector.

  Se define como:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

  donde $\mathbf{x}$ es un vector de entrada, x_i es el elemento i-ésimo de $\mathbf{x}$, y $\text{softmax}(\mathbf{x})_i$ es el i-ésimo elemento del vector resultante después de aplicar softmax.

La función softmax garantiza que todas las salidas sean positivas y su suma sea igual a 1, convirtiendo así los valores de entrada en una distribución de probabilidad. Esto es útil en la capa de salida de una red neuronal, donde se desea clasificar las entradas en diferentes categorías mutuamente excluyentes. La función softmax es especialmente útil en problemas de clasificación multiclase.

2.1.2. Arquitecturas

Perceptrón Multicapa

La arquitectura profunda más sencilla es el Perceptrón Multicapa (Multilayer Perceptron, MLP), que adopta la forma de una sucesión de capas totalmente conectadas separadas por funciones de activación. En estas cada nodo dentro de una capa está conectado a los nodos de la siguiente capa, y cada conexión tiene un peso y un umbral asociados. Si la salida de un nodo supera el valor del umbral especificado, ese nodo se activa y transmite datos a la siguiente capa de la red. De lo contrario, no se envía ningún dato. Estas operaciones de transmisión de datos se realizan mediante las funciones de activación.

Esta arquitectura se compone de tres tipos de capas principales:

- Capa de Entrada (Input Layer): Esta es la capa inicial de la red neuronal donde se introducen los datos de entrada. Cada nodo en esta capa representa una característica de los datos que se está procesando. Por ejemplo, en el caso del reconocimiento de imágenes, cada nodo podría representar un píxel en la imagen.

- Capas Ocultas (Hidden Layers): Estas capas se encuentran entre la capa de entrada y la capa de salida y son responsables de realizar cálculos complejos para extraer características importantes de los datos de entrada. Las redes neuronales pueden tener una o varias capas ocultas, dependiendo de la complejidad del problema. En estas capas, cada nodo está conectado a todos los nodos de la capa anterior y posterior, lo que permite la propagación de la información a lo largo de la red.

- Capa de Salida (Output Layer): Esta es la capa final de la red neuronal donde se produce el resultado de la predicción o clasificación. La cantidad de nodos en esta capa depende del tipo de problema que se esté abordando. Por ejemplo, en un problema de clasificación binaria, puede haber un solo nodo que represente la probabilidad de pertenencia a una clase, mientras que en un problema de clasificación multiclase,

puede haber varios nodos, cada uno representando la probabilidad de pertenencia a una clase diferente.

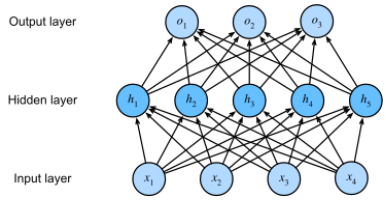


Figura 2.4: MLP con cuatro entradas, una capa oculta de cinco nodos y tres salidas.

En comparación con modelos lineales simples, los perceptrones multicapa permiten aprender representaciones más complejas de los datos. Mientras que los modelos lineales operan bajo la hipótesis de linealidad entre las características de entrada y las salidas, las capas ocultas permiten capturar relaciones no lineales entre estas.

Redes Neuronales Convolucionales (CNNs)

A diferencia de las redes neuronales tradicionales, las CNNs son arquitecturas de redes neuronales diseñadas específicamente para procesar datos con una topología en forma de cuadrícula, como las imágenes. Esta arquitectura única hace que las CNNs sean particularmente efectivas para tareas que implican jerarquías y patrones espaciales.

Las CNN constan principalmente de capas convolucionales (componente central de la CNN y es donde se produce la mayor parte de los cálculos), capas de agrupación o pooling (realizan una reducción de dimensionalidad, reduciendo la cantidad de parámetros en la entrada.) y capas totalmente conectadas (FC).

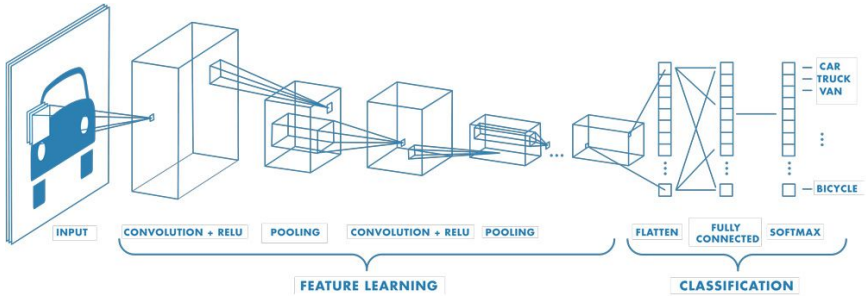


Figura 2.5: Representación de una CNN para una tarea de clasificación de imágenes.

2.2. Entrenamiento de redes neuronales

Para comprender una aproximación básica al entrenamiento de redes neuronales, consideraremos la analogía del alpinista presentada en [14]. Imagina que eres un alpinista en la cima de una montaña y ha caído la noche. Necesitas llegar a tu campamento base al pie de la montaña, pero en la oscuridad, con solo una pequeña linterna, solo puedes ver más unos pocos metros del suelo frente a ti. Entonces, ¿cómo bajas? Una estrategia es mirar en todas direcciones para ver en qué dirección el terreno desciende más y luego avanzar en esa dirección. Repite este proceso muchas veces y gradualmente irá cuesta abajo cada vez más. Esta estrategia eventualmente te llevará al pie de la montaña.

Este escenario puede parecer alejado de las redes neuronales, pero resulta ser una buena analogía de la forma en que se entrenan. De hecho, la técnica principal utilizada para el entrenamiento de redes neuronales, conocida como **descenso de gradiente**, se asemeja considerablemente a este método.

En el contexto de redes neuronales, cuando hablamos de entrenamiento nos referimos a determinar el conjunto óptimo de pesos para optimizar la métrica objetivo del modelo. Esto se consigue minimizando una pérdida que refleja el rendimiento del modelo. Dado que los modelos suelen ser extremadamente complejos y su rendimiento está directamente relacionado con minimizar la pérdida, esta minimización es un reto clave, que implica dificultades tanto computacionales como matemáticas.

2.2.1. Función de Pérdida

Para ajustar nuestro modelo a los datos, se necesita definir una medida de adecuación/ajuste, ésta es la función de pérdida, la cual cuantifica las discrepancias entre las salidas predichas por la red neuronal y los valores objetivo reales. Elegir una función de pérdida adecuada es primordial, ya que influye directamente en la convergencia del modelo, la velocidad del entrenamiento y, en última instancia, en la precisión y generalización de la red.

La elección de la función de pérdida está intrínsecamente ligada a la naturaleza de la tarea de aprendizaje automático: clasificación, regresión, restauración, etc. Cada tipo de tarea tiene requisitos y características específicas que requieren diferentes funciones de pérdida.

Para tareas de regresión lineal, donde el objetivo es predecir valores continuos, la función de pérdida más común es el **Error Cuadrático Medio** (MSE), el cual consiste en calcular el promedio de los cuadrados de las diferencias entre los valores reales, y, y los valores predichos por el modelo,

x:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2$$

La pérdida es un número no negativo en el que los valores más pequeños son obviamente mejores y las predicciones perfectas incurren en una pérdida de 0.

Para tareas de restauración de imágenes, donde es crucial manejar valores atípicos y pequeñas variaciones en los datos, la función **Charbonnier Loss** es especialmente adecuada, la cual se basa en una penalización cuadrática para las diferencias grandes, pero con una raíz cuadrada adicional para proporcionar una respuesta menos sensible a las discrepancias extremas:

$$L_{Charbonnier}(y, \hat{y}) = \sqrt{(y - x)^2 + \epsilon^2} \tag{2.1}$$

donde y representa el valor verdadero real, x representa la predicción del modelo y ϵ es un valor pequeño denominado factor de estabilidad, añadido para evitar la singularidad en el caso de que $y{=}x$ y suavizando la función.

2.2.2. Descenso de Gradiente

Intuitivamente, la forma en que funciona el descenso en gradiente es similar a la analogía del alpinista que presentamos al comienzo de esta sección. El proceso comienza inicializando los parámetros con valores aleatorios y tomándolos como punto de partida. Luego se calcula en qué dirección la función de pérdida $L(w)$ desciende más con respecto al cambio de parámetros (esto se estima mediante el cálculo del gradiente de la función de pérdida) y se avanza ligeramente en esa dirección. Es decir, vamos actualizando los parámetros de modo que la función de pérdida disminuya en mayor cantidad, repitiendo este proceso iterativamente hasta que los parámetros se estabilicen en valores que correspondan al mínimo de la función de pérdida.

El **gradiente** de una función es un vector que indica la dirección y la tasa de cambio más rápidos de la función. Para una función de pérdida $L(w)$, el gradiente con respecto a los parámetros w, denotado como $\nabla L(w)$, es un vector de derivadas parciales de L con respecto a cada uno de los parámetros. Si w es un vector de d dimensiones, entonces el gradiente se define como:

$$\nabla L(w) = \left[\frac{\partial L(w)}{\partial w_1}, \frac{\partial L(w)}{\partial w_2}, \ldots, \frac{\partial L(w)}{\partial w_d}\right]$$

La regla de actualización en la que se basa el descenso de gradiente para los parámetros w en la iteración n es:

$$w_{n+1} = w_n - \eta \nabla L(w_n)$$

donde η es la tasa de aprendizaje (o learning rate), un hiperparámetro positivo que determina el tamaño del paso de cada actualización, y $\nabla L(w)$ es el gradiente de la función de pérdida con respecto a los parámetros w.

La **tasa de aprendizaje** (η) es un hiperparámetro crucial en este algoritmo ya que determina la magnitud del paso que damos en cada iteración. La elección de η afecta significativamente a la convergencia del algoritmo, ya que si η es muy pequeña, la convergencia será lenta realizando actualizaciones muy pequeñas a los parámetros, y en casos extremos el algoritmo podría quedarse atrapado en un mínimo local y no encontrar el mínimo global en un tiempo razonable. Sin embargo, si η es muy grande, los pasos serán demasiado grandes lo que puede hacer que el algoritmo se desestabilice y no converja, o incluso que sobrepase el mínimo oscilando alrededor de este sin llegar a alcanzarlo.

Calcular este gradiente sobre todo el conjunto de datos puede ser computacionalmente costoso, especialmente para conjuntos de datos grandes. Por lo tanto, a menudo se utilizan aproximaciones.

Descenso de Gradiente Estocástico (SGD) por mini-lotes

Este algoritmo es una variación del descenso de gradiente completo, donde el conjunto completo de datos se subdivide en mini-lotes de datos de tamaño fijo. Este estima el gradiente y actualiza los parámetros utilizando solo un mini-lote de los datos en cada iteración mitigando así el problema del coste computacional. El procesamiento en mini-lotes se utiliza debido a que, en muchos casos, el conjunto completo de datos excede la capacidad de memoria de una GPU. Dividir los datos en mini-lotes permite aprovechar eficientemente la capacidad de procesamiento de las GPUs modernas, optimizando así el rendimiento computacional.

Se han propuesto muchas variaciones para abordar las limitaciones del modelo estándar. La más destacada es **Adam**, que mantiene en ejecución estimaciones de la media y la varianza de cada componente del gradiente, y las normaliza automáticamente, evitando problemas de escalado y diferentes velocidades de entrenamiento en distintas partes de un modelo.

2.2.3. Cálculo de gradientes: Propagación hacia atrás (Backpropagation)

La *propagación hacia atrás* se refiere al método de calcular el gradiente de los parámetros de la red neuronal. El método atraviesa la red en orden inverso, desde la capa de salida a la de entrada, según la regla de la cadena del cálculo. Durante este proceso, se calculan y almacenan las derivadas parciales necesarias para obtener el gradiente con respecto a algunos parámetros.

Para entender cómo funciona, consideremos dos funciones $Y = f(X)$ y $Z = g(Y)$, donde X, Y y Z son tensores de formas arbitrarias. Usando la regla de la cadena, podemos calcular la derivada de Z con respecto a X como:

$$\frac{\partial Z}{\partial X} = \frac{\partial Z}{\partial Y} \cdot \frac{\partial Y}{\partial X}$$

El objetivo de la propagación hacia atrás es calcular ahora cada gradiente $\frac{\partial L(w)}{\partial w_i}$. Para ello aplicamos la *regla de la cadena* (que nos permite descomponer una derivada como producto de sus partes funcionales individuales) de manera sistemática a través de las capas de la red neuronal. Si hacemos un seguimiento de las diferencias en un paso hacia delante a lo largo de cada conexión y las almacenamos, podemos calcular el gradiente tomando el término de pérdida encontrado al final del paso hacia delante y "propagando el error hacia atrás" a través de cada una de las capas.

Este proceso se facilita enormemente con el uso del algoritmo **Autograd** [16] [17], que automatiza el cálculo de gradientes en operaciones tensoriales complejas.

Cuando se define un modelo neuronal, cada operación que se realiza sobre los tensores se registra en un grafo computacional. Este grafo representa la estructura de cálculo desde la entrada hasta la salida del modelo. Cuando se solicita el cálculo del gradiente de alguna función de pérdida con respecto a los parámetros del modelo, autograd puede utilizar este grafo para calcular automáticamente las derivadas utilizando la regla de la cadena. Así, autograd simplifica significativamente la implementación del algoritmo de retropropagación, ya que maneja el seguimiento de las dependencias entre las operaciones y realiza eficientemente el cálculo de gradientes.

2.3. Pytorch

PyTorch [18] es una librería de código abierto orientada a aprendizaje automático desarrollada por Facebook's AI Research lab (FAIR), la cual facilita la creación y entrenamiento de modelos de redes neuronales. Esta se ha convertido en una de las herramientas más populares para el desarrollo de modelos de aprendizaje profundo debido a su facilidad de uso y flexibilidad.

2.3.1. Tensores

Los tensores conforman la estructura de datos fundamental en PyTorch. Estos son estructuras de datos multidimensionales similares a los arrays de NumPy, pero con la capacidad adicional de ser utilizados en una GPU para acelerar el cómputo. Estos tensores son fundamentales para el manejo de

datos y cálculos en redes neuronales.

Para crear un tensor en PyTorch, se puede utilizar:

```
import torch

# Crear un tensor sin inicializar de tamano 5x3
x = torch.empty(5, 3)

# Crear un tensor con valores aleatorios de tamano 5
    x3
x = torch.rand(5, 3)

# Crear un tensor con ceros y de tipo de dato long,
    de tamano 5x3
x = torch.zeros(5, 3, dtype=torch.long)

# Crear un tensor directamente desde datos, en este
    caso de tamano 1x2
x = torch.tensor([5.5, 3])
```

2.3.2. Módulos en Pytorch (torch.nn.module)

PyTorch facilita la construcción de redes neuronales mediante la clase `torch.nn.Module`. Esta clase proporciona una estructura base con lógica *forward pass*, que puede ser heredada para construir arquitecturas más complejas.

Otros módulos a tener en cuenta a la hora de construir redes son los referentes a las capas:

- Capas Lineales o Completamente conectadas: `nn.Linear`
- Capas Convolucionales: `nn.Conv2d`
- Capas de Agrupación: `nn.MaxPool2d`
- Funciones de Activación: `nn.ReLU`, `nn.Softmax`, etc.

Veamos un ejemplo simple de definición de una red neuronal:

```
import torch.nn as nn

class Net(nn.Module):

```

```
5     def __init__(self):
6         super(Net, self).__init__()
7         # Define una capa completamente conectada
              con 784 entradas y 512 salidas.
8         self.fc1 = nn.Linear(10, 512)
9         # Define una capa de salida con 512 entradas
              y 10 salidas.
10        self.fc2 = nn.Linear(512, 10)
11
12    def forward(self, x):
13        x = torch.relu(self.fc1(x))  # Aplicar ReLU
              a la salida de fc1
14        x = self.fc2(x)
15        return x
```

En el código anterior se define una red neuronal sencilla con dos capas completamente conectadas y una lógica forwad pass donde:

1. Los datos de entrada x se pasan a través de la primera capa lineal (torch.fc1) y la salida de esta es luego pasada a través de la función de activación ReLU (torch.relu).

2. La salida de la primera capa activada se pasa a la segunda capa (torch.fc2).

3. La salida de la segunda capa es retornada como la salida final de la red.

2.3.3. Funciones de Pérdida y Backward

Funciones de Pérdida

Pytorch proporciona varias funciones de pérdida predefinidas en el módulo torch.nn, como `torch.nn.MSELoss` para regresión y `torch.nn.CrossEntropyLoss` para clasificación. Sin embargo no contiene ninguna función definida para la función de pérdida CharbonnierLoss que nombramos anteriormente en este capítulo para procesamiento de imágenes. A continuación se propone una implementación sencilla de esta función:

```
1 class CharbonnierLoss():
2     def __init__(self, eps=1e-3):
3         self.eps= eps
4
5     def __call__(self, pred, target):
```

```
6         loss = torch.sqrt((pred - target)**2 + self.
              eps).mean()
7
8         return loss
```

donde la variable `target` corresponde a la imagen objetivo, `pred` es la predicción del modelo y `eps` es el factor de estabilidad.

Backward

El método backward se utiliza para calcular el gradiente de la función de pérdida con respecto a los parámetros del modelo. Una vez calculados los gradientes hay que actualizar estos parámetros mediante un optimizador, para ello Pytorch ofrece optimizadores como SGD y Adam.

A continuación se muestra como se aplica backward y un optimizador en el bucle de entrenamiento del modelo:

```
1 # Definir funcion de perdida y optimizador
2 criterion = CharbonnierLoss()
3 optimizer = optim.Adam(net.parameters(), lr=0.001)
4
5
6 # Ejemplo de entrenamiento
7 for epoch in range(num_epochs):
8     for data in enumerate(train_data_loader):
9         # Obtener los inputs; data es una lista de [
              inputs, targets]
10        inputs, targets = data
11
12        # Calcular perdida y actualizar parametros
13        optimizer.zero_grad()   # Reiniciar
              gradientes
14        outputs = net(inputs)
15        loss = criterion(outputs, targets)
16        loss.backward() # Calcular gradientes usando
              Backpropagation
17        optimizer.step()    # Actualizar parametros
              del modelo
```

En el código anterior, primero se instancia un criterio de pérdida y un optimizador, en este caso `CharbonnierLoss` y `Adam` respectivamente. A continuación en el bucle de entrenamiento del modelo, primero se reinician los

gradientes del optimizador con `optimizer.zero_grad()`. Se calcula la pérdida `loss` y seguidamente los gradientes con `loss.backward()`. Finalmente se actualizan los parámetros del modelo utilizando `optimizer.step()`.

Una vez estudiados los fundamentos de las redes neuronales profundas y comprendido sus componentes esenciales, en los próximos capítulos exploraremos diversos algoritmos y modelos que hacen uso de ellas.

Capítulo 3

Deep Image Prior

Tradicionalmente, tareas como la eliminación de ruido y emborronamiento, la superresolución y el relleno de huecos se han abordado mediante CNNs entrenadas en extensos conjuntos de datos para aprender patrones específicos de ellos. Sin embargo, Deep Image Prior (DIP) trata de demostrar que el diseño arquitectónico de las CNNs, puede servir como un poderoso predictor para estas tareas de restauración de imágenes.

En este capítulo nos adentraremos en el concepto de DIP, definiendo el enfoque y los objetivos que persigue. Para ello, exploraremos el método en detalle, destacando los pasos y técnicas implicados en la aplicación de DIP a la tarea de restauración de imágenes. Por último, analizaremos la importancia de la parada temprana en el contexto de DIP y cómo ayuda a evitar el sobreajuste, garantizando un rendimiento óptimo de la red.

3.1. Motivación y definición

El método Deep Image Prior (DIP) [19] desafía la suposición común de que la restauración efectiva de imágenes depende de redes neuronales convolucionales profundas (ConvNets) preentrenadas en grandes conjuntos de datos. En lugar de eso, este método propone explorar la capacidad inherente de la estructura de las ConvNets para capturar estadísticas de imágenes y realizar restauraciones efectivas sin necesidad de entrenamiento previo en grandes conjuntos de datos.

3.1.1. Método

En términos simples, una red generadora profunda es una función paramétrica $x = f_\theta(z)$ que mapea un vector de código z a una imagen x; Cuando hablamos de 'código', en este contexto nos referimos a un vector

que representa una cierta abstracción o característica de la imagen que se está generando o restaurando, es decir, una representación numérica de alguna propiedad de la imagen, que la red utiliza como base para generar o restaurar la imagen final. Estas redes generativas se emplean comúnmente para modelar una distribución compleja $p(x)$ sobre imágenes, como la transformación de una distribución simple $p(z)$ sobre los códigos.

En este contexto, interpretamos la red como una parametrización $x = f_\theta(z)$ de la imagen $x \in \mathbb{R}^{3\times H\times W}$, donde el código z es un tensor aleatorio fijo, y la red mapea los parámetros θ, que comprenden los pesos y sesgos de los filtros en la red, a la imagen x. La red tiene una estructura estándar y utiliza operaciones de filtrado como convolución lineal, upsampling y funciones de activación no lineales.

Sin entrenamiento en un conjunto de datos, no podemos esperar que una red f_θ conozca conceptos específicos como la apariencia de ciertas clases de objetos. Sin embargo, DIP demuestra que la red sin entrenar captura algunas de las estadísticas de bajo nivel de las imágenes naturales, en particular, la naturaleza local e invariante a la traslación de las convoluciones, lo que es suficiente para modelar distribuciones condicionales de imágenes $p(x|x_0)$ (donde x_0 es la imagen borrosa inical) que surgen en problemas de restauración de imágenes, como la eliminación de ruido, la superresolución y el inpainting (restaurar áreas dañadas o rellenar áreas faltantes).

En lugar de de trabajar con distribuciones, DIP formula estos problemas como problemas de minimización de energía del tipo:

$$x^* = \arg\min_x E(x; x_0) + R(x)$$

donde $E(x; x_0)$ es un término de datos dependiente de la tarea, x_0 es la imagen a restaurar, y $R(x)$ es un regularizador que captura la regularidad genérica de las imágenes naturales. DIP presciende de este $R(x)$ y en su lugar utiliza el modelo implícito capturado por la parametrización de la red neuronal, lo que resulta en la minimización de la energía de la forma:

$$\theta^* = \text{argmin}_\theta E(f_\theta(z); x_0),$$

$$x^* = f_{\theta^*}(z)$$

donde el minimizador θ^* óptimo se obtiene utilizando un optimizador como el descenso de gradiente, comenzando desde una inicialización aleatoria de los parámetros θ . Por lo tanto, la única información empírica disponible para el proceso de restauración es la imagen ruidosa x^*. Dado el minimizador resultante θ^* el resultado del proceso de restauración se obtiene como $x^* = f_{\theta^*}(z)$.

En la siguiente imagen se muestra de manera gráfica el proceso de restauración de imágenes utilizando Deep Image Prior:

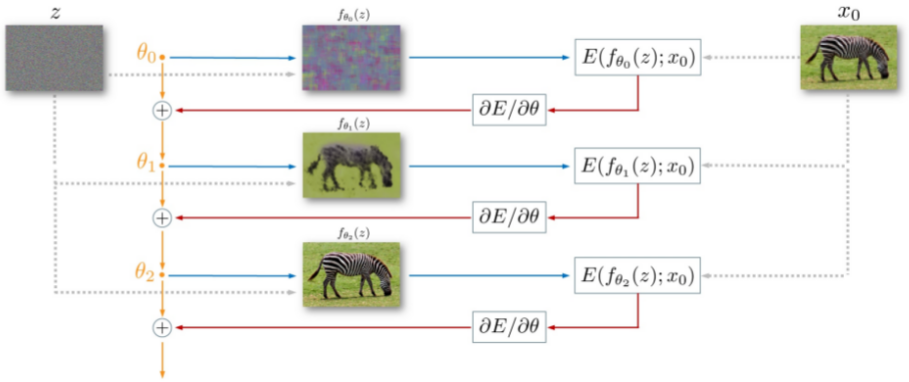


Figura 3.1: Restauración utilizando Deep Image Prior.

A continuación describimos brevemente el proceso:

1. **Inicialización**: Comenzamos con pesos aleatorios θ_0.
2. **Generación de imagen**: En cada iteración t, los pesos actuales θ_t se usan para mapear un tensor fijo z a una imagen:
$$x_t = f_{\theta_t}(z)$$
donde f es la red neuronal parametrizada por θ.
3. **Cálculo de la pérdida**: La imagen generada x_t se compara con la imagen original x_0 utilizando una función de pérdida $E(x_t, x_0)$.
4. **Cálculo del gradiente**: Se calcula el gradiente de la pérdida con respecto a los pesos:
$$\frac{\partial E}{\partial \theta}$$
5. **Actualización de los parámetros**: Utilizando el gradiente, los pesos θ se actualizan mediante un algoritmo de optimización, como por ejemplo el descenso de gradiente.
6. **Iteración**: Repetimos los pasos 2 a 5 hasta que la pérdida se minimice suficientemente o se alcance un número máximo de iteraciones. El objetivo es encontrar los parámetros optimizados:
$$\theta^* = \text{argmin}_\theta E(f_\theta(z); x_0)$$
7. **Resultado final**: La imagen restaurada se obtiene con los parámetros optimizados:
$$x^* = f_{\theta^*}(z)$$

3.2. Early stopping para Deep Image Prior

Los modelos DIP prácticos suelen estar significativamente sobredimensionados. Aunque este sobredimensionamiento permite que los modelos aprendan rápidamente el contenido visual deseado, también los hace susceptibles al sobreajuste (*over-fitting*). Durante el proceso de aprendizaje, estos modelos primero capturan las estructuras visuales esenciales, y seguidamente entran en una fase de sobreajuste en la que también aprenden del ruido presente en la imagen, degradando así la calidad de la imagen reconstruida. A este fenómeno se le denomina **Early learning then overfitting** (ELTO)

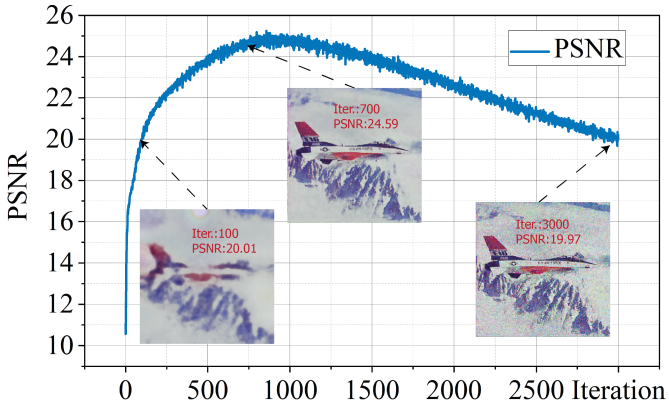


Figura 3.2: Representación del fenómeno ELTO en DIP

Este fenómeno requiere de una estrategia efectiva de interrupción temprana (Early Stopping, ES), que permita detener el proceso de entrenamiento en el punto óptimo antes de que ocurra el sobreajuste. Sin embargo, identificar el punto óptimo no es trivial y existen diferentes formas de abordarlo:

- **Inspección Visual**: Aunque este enfoque puede funcionar para tareas de pequeña escala se vuelve inviable en muchos escenarios de mayor escala como: recuperación de contenidos visuales difíciles de examinar con los ojos (por ejemplo, objetos visuales 3D o 4D), imágenes científicas de objetos desconocidos (por ejemplo, imágenes microscópicas de nuevas especies de virus), etc.

- **Métricas de Calidad de Imagen**: Las métricas tradicionales de calidad de imagen son a menudo inviables sin acceso a una imagen original de referencia. Aún considerando la pérdida como medida para determinar cuando parar el entrenamiento, no se ha encontrado correlación con el punto óptimo, ya que no indica de manera confiable cuándo el modelo comienza a sobreajustarse.

- **Ajuste Ad-Hoc del Número de Iteraciones**: Muchos trabajos previos recurren a ajustar manualmente el número de iteraciones. Este enfoque es altamente ineficiente ya que en DIP el número óptimo de iteraciones puede variar significativamente entre diferentes imágenes, requiriendo numerosos pasos de prueba y error que pueden conducir a resultados subóptimos.

- **Interrupción Temprana Basada en Validación**: Aunque es común en el aprendizaje supervisado, la interrupción temprana basada en validación es desafiante para los modelos DIP ya que se ocupan de instancias únicas sin datos de entrenamiento separados. Los intentos de adaptar este enfoque dividiendo la observación en conjuntos de 'entrenamiento' y 'validación' han resultado problemáticos.

Para abordar el problema de DIP de determinar el punto óptimo donde se ha de interrumpir el entrenamiento el artículo *Early Stopping for Deep Image Prior* [20] propone una estrategia innovadora de ES mediante la monitorización de la varianza durante el proceso entrenamiento del modelo.

A continuación describiremos el método de interrupción temprana basado en el cálculo de la varianza móvil (ES-WMV) presentado en [20], el cual utilizaremos posteriormente en la fase de experimentación.

3.2.1. Método

Dada una imagen de referencia desconocida x de tamaño N, la idea es reconstruir iterativamente x utilizando una red neuronal f_θ parametrizada por θ. En cada iteración t, x_t se obtiene como $f_\theta(z)$, donde z representa el ruido de entrada. El objetivo es determinar el punto en el que la red ha aprendido la información más útil antes de que el ruido z comience a dominar.

Esto atendiendo al MSE, corresponde gráficamente a un patrón en forma de 'U' durante el entrenamiento, caracterizado por una disminución rápida inicial seguida de un aumento debido a efectos de ruido, donde la reconstrucción óptima se encuentra en el valle de la 'U'. Dada que la imagen de referencia es desconocida, no podemos observar directamente el MSE. En su lugar se propone monitorizar la varianza en ejecución (VAR) de las salidas de la red sobre una ventana deslizante.

Definimos la secuencia de reconstrucción $\{x_t\}_{t\geq 1}$ donde $x_t \approx f_{\theta_t}(z)$. La varianza en ejecución VAR(t) se define como:

$$\text{VAR(t)} \approx \frac{1}{W}\sum_{w=0}^{W-1}\left\| x_{t+w} - \frac{1}{W}\sum_{i=0}^{W-1} x_{t+i}\right\|_F^2$$

donde W es el tamaño de la ventana. Inicialmente, a medida que el modelo aprenden de la imagen , el MSE y VAR disminuyen. La observación clave es que cerca de la reconstrucción óptima (valle de MSE), VAR(t) también muestra una curva en forma de U similar, alineándose así con el comportamiento de MSE.

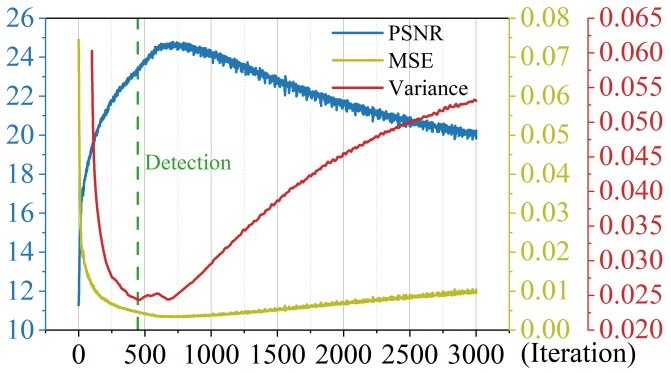


Figura 3.3: Representación donde podemos ver que el valle de la curva VAR, esta alineado con el valle de la curva MSE.

El algoritmo propuesto implica calcular la curva VAR y detectar su valle, deteniendo el entrenamiento cuando se detecte este. Para obtener la curva configuramos un parámetro *ventana* (W), que define cuántas iteraciones anteriores se consideran al calcular la varianza en ejecución. y calculamos la Varianza móvil en ventana (WMV). Se introduce también un parámetro *paciencia* (P), que permite tolerar el estancamiento de la varianza durante P pasos consecutivos.

El algoritmo ES-WMV comienza con $\text{VAR}_{\text{min}} = \infty$, que almacena la varianza mínima, una cola Q vacía para almacenar temporalmente los valores de x (reconstrucción de la imagen) que se procesan en cada iteración k, una semilla aleatoria z y θ_0 inicializado de manera aleatoria también. En cada iteración, se actualiza θ para obtener θ_{k+1} (donde k indica la iteración actual) y el nuevo valor x_{k+1}. Este valor se añade a la cola Q, y se elimina de la cola si la longitud de Q excede el tamaño de ventana W. Cuando Q alcanza el tamaño de W, se calcula la VAR (varianza) de los elementos en Q. Si la VAR calculada es menor que VAR_{min}, se actualiza VAR_{min} y se guarda el valor correspondiente de x_{k+1} como x. Si VAR_{min} permanece estancada durante P iteraciones consecutivas, es decir, la VAR no disminuye con respecto a VAR_{min} durante P iteraciones, el algoritmo se detiene y devuelve x. A continuación se presenta el algoritmo:

Algorithm 1: DIP con ES-WMV

Input : semilla aleatoria z, θ_0 inicializado aleatoriamente, tamaño de ventana W, paciencia P, cola vacía Q, contador de iteraciones $k = 0$, VARmin $= \infty$

Output: reconstrucción x^*

while *no se detenga* **do**
- actualizar θ para obtener θ_{k+1} y x_{k+1};
- agregar x_{k+1} a Q, desencolar si $|Q| > W$;
- **if** $|Q| = W$ **then**
  - calcular VAR de los elementos en Q;
  - **if** $VAR < VAR_{min}$ **then**
    - $\text{VAR}_{\text{min}} \leftarrow \text{VAR}$, $x^* \leftarrow x_{k+1}$;
  - **end**
  - **if** VAR_{min} *estanca durante* P *iteraciones* **then**
    - detener y devolver x^*;
  - **end**
- **end**
- $k = k + 1$;

end

Capítulo 4

Mejora de residuos basados en la pseudoinversa de Moore-Penrose

En este capítulo abordaremos el modelo presentado en [21] para la restauración de imágenes utilizando la pseudoinversa de Moore-Penrose para obtener una estimación inicial de la imagen restaurada a partir de una estimación del emborronamiento y, una red de filtros dinámicos para la eliminación de artefactos de la estimación inicial. Además se explorará como podemos mejorar este enfoque combinándolo con la técnica de DIP.

4.1. Descripción del método y entrenamiento

Retomemos la ecuación 1.1 para la deconvolución de imágenes presentada en la sección 1.1, manteniendo la misma notación, y siendo H la matriz asociada al núcleo de desenfoque.

En el contexto de NBID, la tarea de deconvolución se vuelve difícil debido a la naturaleza mal planteada del problema, esto se debe a la presencia de ruido y las propiedades espectrales de H que amplifican este ruido en la deconvolución. En el caso de BID la complejidad aumenta al tener que estimar H.

Consideremos un hipotético escenario donde el desenfoque H es conocido y no hay ruido n:

$$y = Hx$$

Idealmente para estimar la imagen nítida x, deberíamos calcular $x = H^{-1}y$, sin embargo, es posible que no exista la inversa de H. Para abordar este problema, en [21] se propone utilizar la **pseudoinversa de Moore-Penrose**

H^+.

La pseudoinversa de Moore-Penrose se define como una generalización de la inversa de una matriz para matrices no cuadradas o no invertibles. Dada una matriz H, la pseudoinversa H^+ satisface las siguientes propiedades:

- $HH^+H = H$
- $H^+HH^+ = H^+$
- $(HH^+)^T = HH^+$
- $(H^+H)^T = H^+H$

y puede demostrarse que [22]:

$$H^+ = \lim_{\delta \to 0^+} (H^T H + \delta I)^{-1} H^T \tag{4.1}$$

donde δ es un parámetro de regularización.

Entonces volviendo al escenario anterior donde no había presencia de ruido $y = Hx$, la imagen nítida x puede estimarse utilizando la pseudoinversa de Moore-Penrose:

$$x = H^+ y$$

Sin embargo, H^+y sólo puede recuperar las frecuencias de x en las que la transformada de Fourier de H es distinta de cero. Las frecuencias correspondientes al espacio nulo de H no son posibles de recuperar utilizando la pseudoinversa y por lo tanto se aprenden de la imagen original, para ello se define:

$$g_\theta(z) = (I - HH^+) f_\theta(z) + H^+ y, \tag{4.2}$$

donde $g_\theta(z)$ es la solución de deconvolución, y f_θ es una CNN con parámetros θ, donde z son las entradas que especificaremos más adelante.

Teniendo en cuenta la propiedad $HH^+H = H$, tenemos:

$$Hg_\theta(z) = H(I - HH^+) f_\theta(z) + HH^+ y = HH^+ y = y$$

Esto demuestra que $g_\theta(z)$ es una solución válida para la deconvolución, ya que reproduce la observación y.

Entonces la imagen restaurada $g_\theta(z)$ satisface:

$$x - g_\theta(z) = (x - H^+ y) - (I - HH^+) f_\theta(z),$$

resolviendo así el problema de deconvolución con desenfoque H conocido y sin ruido.

4.1.1. Modelo general

La presencia de ruido dificulta el uso del modelo anterior (4.2), ya que la aplicación directa de H^+ puede amplificar este. Para mitigar esto H^+ definida en (4.1) se utiliza al filtro de Wiener:

$$H_e^+ = (H^T H + eI)^{-1} H^T \tag{4.3}$$

donde $e > 0$ es un parámetro de regularización que controla el ruido y evita la pérdida de aquellas frecuencia para las cuales H es cero. Esto conduce a una estimación inicial de la imagen nítida:

$$x_w = H_e^+ \cdot y$$

Dada esta estimación x_0 obtenida mediante la aproximación del filtro de Wiener, el objetivo es mejorarla aprendiendo las frecuencias residuales perdidas durante la deconvolución inicial. Luego. (4.2) adoptaría la forma:

$$g_\theta(z) = (I - HH_e^+) f_\theta(z) + H_e^+ y = (I - HH_e^+) f_\theta(z) + x_w \tag{4.4}$$

donde f_θ es la CNN con parámetros θ y z es la entrada de la red, que se compone de la concatenación de x_w e y a lo largo del canal:

$$z = (x_w, y)$$

La entrada de la imagen borrosa y es importante debido a que algunos artefactos que aparecen en la deconvolución usando el del filtro de Wiener x_w, son difíciles de distinguir de estructuras reales, lo que complica su eliminación, por lo que incorporar y ayuda a la red a diferenciar entre artefactos y objetos reales de la escena, eliminando estructuras que no están presentes en y y manteniendo la coherencia con la observación.

4.1.2. Eliminación de artefactos de deconvolución con DFN

La imagen deconvolucionada obtenida en el apartado anterior por la aplicación de (4.4) aún puede presentar algunos artefactos provocados por las imprecisones en la estimación del desenfoque u otras incoherencias en el modelo. Para eliminarlos se propone una **Dynamic Filter Network** (DFN), la cual genera filtros espacialmente variantes que se aplican a la imagen deconvolucionada.

Para un píxel (i, j), sea $d_\psi^{u_{i,j}}$el filtro de soporte $(2L + 1) \times (2M + 1)$ generado por la red generadora de filtros con parámetros ψ a partir de la imagen de entrada u (notar que estamos utilizando notación bidimensional y que la red genera el filtro desde una región centrada alrededor de $u(i, j)$). El conjunto de todos los filtros generados a partir de la imagen u se denota

como d^u_ψ. Entonces, el píxel en la posición (i, j) de la imagen filtrada r obtenida por la capa de filtrado dinámico en la imagen de entrada q con el conjunto de filtros d^u_ψ se define como:

$$r(i,j) = \sum_{l=-L}^{L} \sum_{m=-M}^{M} d_\psi^{u_{i,j}}(l+L, m+M) \cdot q(i+l, j+m)$$

Por lo tanto, la aplicación del filtro dinámico a g_θ se espera que sea una imagen restaurada nítida sin artefactos $\hat{x}$ dada por:

$$\hat{x} = r_{\theta,\psi}(x_w, y) = D_\psi^{x_w,y} g_\theta(x_w, y) \tag{4.5}$$

donde $D_\psi^{x_w,y}$ representa la matriz de convolución dinámica asociada con el conjunto de núcleos $d_\psi^{x_w,y}$. La DFN adapta dinámicamente los filtros a las características locales de la imagen, eliminando eficazmente los artefactos y preservando los detalles de esta.

4.1.3. Arquitectura de la red

En la siguiente imagen se ilusta la arquitectura de la DDNet propuesta en [21]:

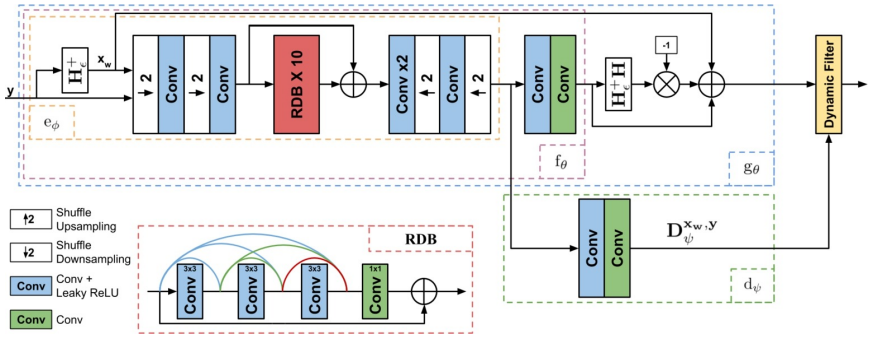


Figura 4.1: DDNet

El cuadro azul representa el modelo $g_\theta(x_w, y)$. La rama principal e_θ (la cual se comparte con la DFN para reducir el número de cálculos) sigue una estructura codificador-decodificador. En la fase de codificación, se extraen características de la imagen y se reduce su resolución espacial por un factor de 4 usando dos operaciones de espacio a profundidad ($S2 : [0,1]^{2H\times 2W\times C} \rightarrow [0,1]^{H\times W\times 4C}$). Luego, las características se transforman mediante 10 bloques densos residuales (RDB); estos son bloques residuales modificados con conexiones densas entre capas para reutilizar características y mejorar el rendimiento. Finalmente las características se escalan nuevamente al tamaño original de la imagen usando dos convoluciones sub-píxel de factor 2.

La reducción del tamaño espacial permite procesar la imagen más rápido y aumenta el campo receptivo sin incrementar la profundidad del modelo de CNN. Las características obtenidas por la rama principal se utilizan para calcular la estimación inicial de la imagen $g_\theta(x_w, y)$ y los filtros $d_\psi(x_w, y)$. El resultado de aplicar los filtros, calculados a través del módulo de filtro dinámico, produce la estimación final $\hat{x}$ en la ecuación (4.5) del modelo propuesto $r_{\theta,\psi}(x_w, y)$..

Todas las convoluciones que no son de salida fuera de un bloque RDB utilizan filtros $64 \times 3 \times 3$ seguidos por una activación Leaky ReLU con una pendiente negativa de 0,2. Las tres primeras convoluciones en cada RDB utilizan filtros de $32 \times 3 \times 3$, mientras que la última convolución utiliza filtros de $64 \times 1 \times 1$ para controlar el incremento en el número de parámetros debido al uso de conexiones densas. El tamaño de los filtros predichos por d_ψ es 5×5, con $M = L = 2$.

4.2. Mejora con Deep Image Prior

A raíz de este método surgió la duda de si podríamos aprovechar el potencial de DIP para mejorar los resultados obtenidos. La hipótesis se basa en que DIP orienta el entrenamiento del modelo a capturar las características y estructura de una imagen concreta. Lo que se propone entonces para mejorar los resultados de DDNet, es que una vez finalizado el entrenamiento de este para un conjunto de datos, y haber obtenido el conjunto de pesos de entrenamiento correspondiente, ahora para la deconvolución de una imagen concreta X, partir del conjunto de pesos inicial obtenido, y actualizar estos, reentrenando el modelo con la propia imagen X. De esta manera la deconvolución de una imagen consta ahora de dos fases: primero, se entrena el modelo con un conjunto extenso de datos para generar los pesos de entrenamiento iniciales; segundo, se actualizan estos pesos mediante el reentrenamiento de la red utilizando únicamente la imagen que se desea deconvolucionar. Finalmente, se emplean estos pesos actualizados para obtener la imagen deconvolucionada final.

El proceso de deconvolución aplicando DIP al modelo DDNet se ilustraría entonces de la siguiente forma:

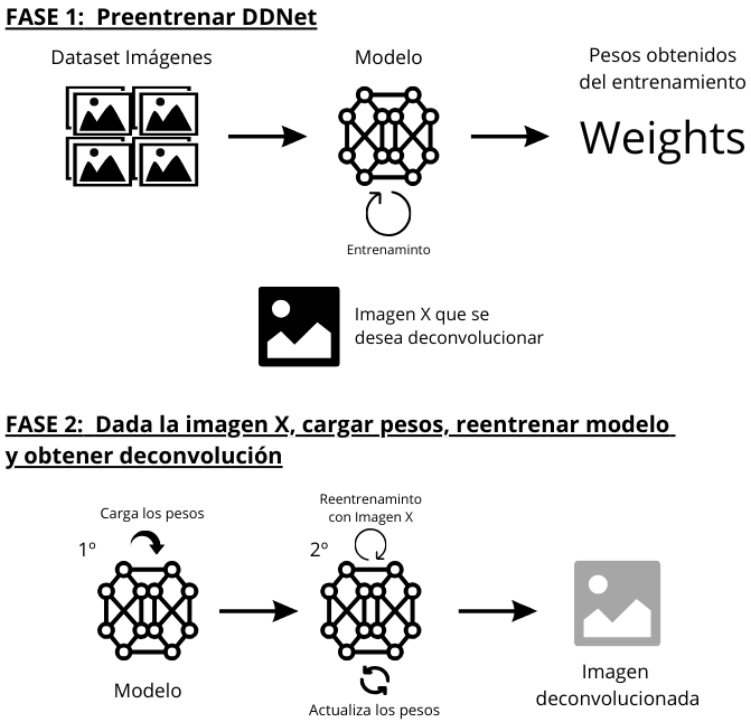


Figura 4.2: Proceso de deconvolución de una imagen con el modelo propuesto

Capítulo 5

Experimentación

En el capítulo anterior, exploramos el uso de la pseudoinversa de Moore-Penrose para la restauración de imágenes, presentando un modelo basado en la deconvolución, y mejorado con una red de filtros dinámicos (DDNet). Además, planteamos la posibilidad de combinar este enfoque con DIP, para obtener imágenes restauradas de mayor calidad.

El objetivo de esta capítulo es investigar como influye el uso de DIP en la restauración de imágenes, partiendo de una DDNet preentrenada. Para ello, utilizaremos los pesos generados por la DDNet tras su entrenamiento en un conjunto extenso de imágenes. DIP cargará estos pesos iniciales, y para cada una de las imágenes de experimentación (distintas a las del conjunto de imágenes de entrenamiento), el modelo se reentrenará con la imagen correspondiente y generará su versión restaurada.

Para el estudio experimental se utilizan dos variantes de DIP:

- DIP con un número preestablecido de épocas.
- DIP+ES, que implementa el método de interrupción temprana (ES) definido en la sección 3.2,

y se analizarán los resultados de ambas variantes en comparación con los de la DDNet.

Todo el código empleado puede encontrarse en `https://github.com/joseantonio2001/CHSG-DDNet-DIP-model`

5.1. Configuración

En esta sección, detallamos las configuraciones empleadas tanto para la DDNet, como para las variantes de DIP, utilizadas en el proceso de experi-

mentación.

5.1.1. Configuración de la DDNet

Para el entrenamiento de la DDNet se ha utilizado el dataset COCO 2017, el cual contiene 118,287 imágenes de entrenamiento, 5,000 imágenes de validación y 40,670 imágenes de prueba.

Para simular las imágenes degradadas se generaron 1,024 PSFs de tamaños entre 11x11 y 65x65 píxeles, con $T = 0.8$ y ansiedad $10^r/1000$, donde r es un número aleatorio de una distribución uniforme $[0, 1]$. De estos PSFs generados 736 se destinaron para entrenamiento, 32 para validación y 256 para pruebas.

Cada imagen de los conjuntos de entrenamiento y validación fue emborronada con 3 PSFs de su propio conjunto (imágenes de entrenamiento con PSFs de entrenamiento e imágenes de validación con PSFs de validación) y se les añadió ruido gaussiano con una desviación estándar $\sigma = 0.01$. Las imágenes con un tamaño inferior a 320 píxeles en la dimensión más corta se descartaron para evitar artefactos en los bordes. Finalmente, se recortó la parte central de 256x256 píxeles de cada imagen. Este procesamiento resultó en 347,436 imágenes de entrenamiento y 14,637 imágenes de validación.

Para las pruebas, se creó un conjunto de 512 imágenes, obtenidas degradando dos imágenes de prueba elegidas al azar para cada núcleo en el conjunto de prueba y agregando ruido gaussiano con $\sigma = 0.01$.

Dado que frecuentemente el kernel de desenfoque real no está disponible, se utilizó una variación del método de Deconvolución Ciega Bayesiana Rápida (Fast Bayesian Blind Deconvolution) con priors Huber Super Gaussian (HSG) para obtener estimaciones suficientemente precisas antes del entrenamiento. El desenfoque se estima utilizando un enfoque multiescala de grueso a fino para evitar mínimos locales. En cada escala, el método impone ´sparsity´ en los bordes de la imagen, utilizando un prior HSG (Sparsity es una característica de Deep Learning que permite realizar una detección automática de las características más relevantes del espacio de entrada). Para el desenfoque, no se supuso ningún conocimiento previo aparte de la no negatividad y que los coeficientes de desenfoque deben sumar uno.

Para la limpieza de los kernels se calcula los componentes conectados de 8 vecinos del kernel (tratado como una imagen). Los componentes conectados cuya suma es menor a un umbral de 0.1 se establecieron a cero, eliminando píxeles aislados o conjuntos de píxeles de pequeño valor, generalmente debido al ruido.

Durante el entrenamiento, se realizó un aumento de datos volteando y rotando aleatoriamente cada instancia vertical y horizontalmente en múltiplos

de 90°. El modelo se entrenó durante 35 épocas utilizando el optimizador Adam con un *weight decay* de 10^{-4}. Cada época de entrenamiento constaba de 5,428 lotes de 64 imágenes. La tasa de aprendizaje (*learning rate*) se fijó en 5×10^{-4} para las primeras 5 épocas, 10^{-4} para las siguientes 20 épocas y 10^{-5} para las últimas 10 épocas. El parámetro de regularización para el filtro de Wiener, ϵ, se estableció en 0.01.

Se utilizó la pérdida de Charbonnier 2.1 con el factor de estabilidad $\epsilon = 10^{-3}$.

5.1.2. Configuración de DIP

Para la evaluación de los modelos se utilizó un conjunto de 2500 imágenes, distinto al empleado para el entrenamiento de la DDNet.

Atendiendo al modelo DIP+ES, se estableció una ventana de tamaño 25 y una paciencia de valor 6. Los demás parámetros de entrenamiento para ambas variantes de DIP son idénticos a los utilizados en la DDNet, a excepción del número de épocas que se fijó en 110.

Para medir la calidad de los resultados obtenidos se han utilizado las siguientes métricas:

- Error Cuadrático Medio (MSE): Cuantifica la diferencia promedio entre los valores de los píxeles de una imagen original y una imagen degradada o reconstruida. Un MSE más bajo indica una mayor similitud entre las dos imágenes comparadas.
- Índice de Similitud Estructural (SSIM): Evalúa la similitud entre dos imágenes considerando cambios en la luminancia, el contraste y la estructura. El SSIM varía entre -1 y 1, donde 1 indica que las dos imágenes son idénticas.
- Relación Señal-Ruido Máxima (PSNR): Expresa la relación entre la máxima potencia posible de una señal y la potencia del ruido que afecta la fidelidad de su representación. El PSNR se mide en decibelios (dB). Valores más altos de PSNR indican mejor calidad de la imagen, es decir, menor nivel de ruido.

Hemos seleccionado estas métricas debido a sus características complementarias, las cuales permiten una evaluación integral de la calidad de la imagen restaurada.

5.2. Resultados

Los resultados obtenidos por las variantes del modelo DIP se compararon con los obtenidos para la DDNet. Para la comparación se analiza la media y la desviación típica de cada métrica empleada (MSE, SSIM y PSNR). Los resultados se presentan en la siguiente tabla, donde cada celda muestra los valores como *media* $\pm$ *desviación típica*):

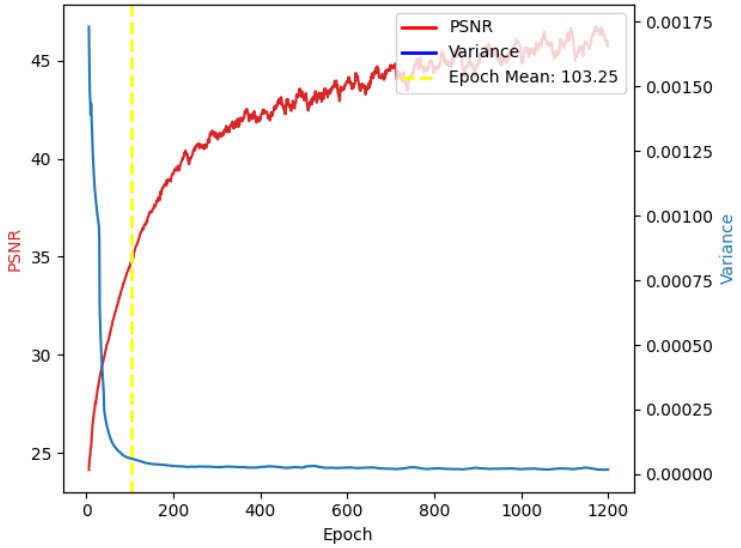
	MSE	**SSIM**	**PSNR**
DIP	49.28 ± 18.52	0.76 ± 0.12	31.58 ± 1.94
DIP+ES	49.57 ± 18.86	0.76 ± 0.12	31.56 ± 1.96
DDNet	52.49 ± 19.16	0.72 ± 0.14	31.28 ± 1.86

Analizando los resultados, podemos ver que la incorporación de DIP al modelo proporciona una clara mejora con respecto a la DDNet base. Específicamente, la variante DIP muestra una reducción significativa del MSE (de 52.49 a 49.28) y una mejora en el SSIM (de 0.72 a 0.76) y el PSNR (de 31.28 a 31.58). Estos resultados indican que el uso de DIP permite obtener imágenes restauradas con mayor precisión y calidad estructural. La reducción del MSE sugiere una menor discrepancia promedio entre la imagen restaurada y la imagen original, mientras que el incremento en SSIM y PSNR indica una mejora en la preservación de detalles y en la relación señal-ruido, respectivamente.

En cuanto a la desviación típica, DIP muestra una disminución en MSE (18.52 frente a 19.16 en DDNet) y en PSNR (1.94 frente a 1.86 en DDNet), lo que sugiere una mayor consistencia en los resultados. Sin embargo, la desviación típica de SSIM se mantiene similar (0.12 frente a 0.14 en DDNet), indicando que la mejora en la similitud estructural es consistente pero con una variabilidad comparable a la de DDNet.

La variante DIP+ES, muestra resultados muy similares a los obtenidos con DIP, presentando así cierta mejora con respecto a la DDNet. Específicamente, se observa un MSE de 49.57, un SSIM de 0.76 y un PSNR de 31.56, valores casi idénticos a los de DIP. En términos de desviación típica DIP+ES también presenta valores muy similares a los de DIP, donde MSE y PSNR es ligeramente mayor (18.86 frente a 18.52 para MSE y 1.96 frente a 1.94 para PSNR), pero SSIM se mantiene igual (0.12). Esto sugiere que ES no introduce variabilidad adicional significativa, manteniendo la consistencia de los resultados.

En el siguiente gráfico podemos ver la época promedio donde ES efectúa la interrupción del entrenamiento para un subconjunto formado por las 65 primeras imágenes del conjunto de experimentación:



Observamos que ES interrumpe el entrenamiento demasiado pronto, pues vemos que el PSNR continua aumentando en las épocas siguientes a la interrupción. Este no es el resultado esperado pues la curva correspondiente al PSNR debería comenzar a descender después de la interrupción como podemos observar en 3.3. El inesperado resultado puede deberse a los valores establecidos para los parámetros learning rate del modelo y paciencia de ES. Un learning rate bajo (como el establecido para el modelo) puede hacer que el modelo no converja los suficientemente rápido haciendo que ES pueda detener el entrenamiento antes de que el modelo haya alcanzado el punto óptimo. Para la paciencia pasa algo similar, ya que una paciencia baja puede hacer que ES detenga el entrenamiento antes de tiempo.

Sin embargo, a pesar de no haber conseguido el comportamiento esperado, el algoritmo ES permite parar el entrenamiento obteniendo resultados muy similares a los de DIP.

Queda claro que ambas variantes (DIP y DIP+ES) en combinación con el modelo DDNet superan en todas las métricas al modelo de referencia, ofreciendo así una restauración de imágenes óptima. Vemos también que las diferencias entre ambas variantes es mínima, lo que indica que aún no habiendo conseguido el comportamiento esperado para ES, esta estrategia resulta efectiva.

A continuación se presentan ejemplos visuales de la imagen original, degradada, y restaurada por los diferentes métodos para una imagen del conjunto de datos de experimentación:

IMAGEN 1

Imagen original

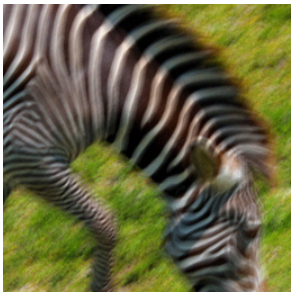


Imagen degradada

NO_DIP
MSE=92.35
SSIM=0.47
PSNR=28.47

DIP
MSE=74.79
SSIM=0.81
PSNR=29.39

DIP+ES
MSE=76.02
SSIM=0.81
PSNR=29.32

IMAGEN 2

Imagen original

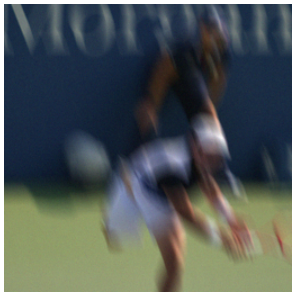


Imagen degradada

NO_DIP
MSE=29.51
SSIM=0.86
PSNR=33.43

DIP
MSE=26.81
SSIM=0.89
PSNR=33.85

DIP+ES
MSE=26.87
SSIM=0.89
PSNR=33.84

Capítulo 6

Conclusiones

En este estudio, hemos explorado dos métodos novedosos para la restauración de imágenes: Dynamic Deblurring Network (DDNet) [21] y Deep Image Prior (DIP) [19].

DDNet presenta un enfoque para la deconvolución de imágenes que combina la pseudoinversa de Moore-Penrose con una red neuronal profunda y una red de filtro dinámico (DFN). La pseudoinversa de Moore-Penrose se utiliza para obtener una estimación inicial de la imagen nítida, mientras que la DFN se emplea para eliminar artefactos de deconvolución y preservar detalles de la imagen.

DIP consiste en el entrenamiento de una red neuronal ajustando los pesos de esta de manera individualizada para cada imagen x que se desee restaurar, lo que otorga una flexibilidad significativa.

El método propuesto en este trabajo se basa en la combinación de ambas técnicas para la restauración de imágenes borrosas y ruidosas (DDNet para obtener una restauración inicial de la imagen y DIP para refinar esta restauración inicial). Mediante pruebas experimentales se ha demostrado que este método supera significativamente a la DDNet original en términos de calidad de la imagen restaurada, suponiendo así una técnica novedosa y eficaz para la restauración de imágenes.

Bibliografía

[1] Daniele Perrone and Paolo Favaro. A logarithmic image prior for blind deconvolution. *International Journal of Computer Vision*, 117(2):159–172, 2016.

[2] Maximum likelihood. `https://www.sciencedirect.com/topics/computer-science/maximum-likelihood`. (Recurso online).

[3] Pablo Ruiz, Xu Zhou, Javier Mateos, Rafael Molina, and Aggelos K. Katsaggelos. Variational bayesian blind image deconvolution: A review. *Digital Signal Processing*, 47:116–127, 2015. Special Issue in Honour of William J. (Bill) Fitzgerald.

[4] Bayesian approach. `https://www.sciencedirect.com/topics/computer-science/bayesian-approach`. (Recurso online).

[5] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3883–3891, 2017.

[6] Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Jiří Matas. Deblurgan: Blind motion deblurring using conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8183–8192, 2018.

[7] Orest Kupyn, Tetiana Martyniuk, Junru Wu, and Zhangyang Wang. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8877–8886, 2019.

[8] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

[9] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the

alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[10] Donald Geman and Chengda Yang. Nonlinear image recovery with half-quadratic regularization. *IEEE Transactions on Image Processing*, 4(7):932–946, 1995.

[11] Florent Sureau, Alexis Lechat, and J-L Starck. Deep learning for a space-variant deconvolution in galaxy surveys. *Astronomy & Astrophysics*, 641:A67, 2020.

[12] François Fleuret. The little book of deep learning. *A lovely concise introduction*, page 297, 2023.

[13] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. `https://D2L.ai`.

[14] How neural networks are trained. `https://ml4a.github.io/ml4a/how_neural_networks_are_trained/`. (Recurso online).

[15] Convolutional neural networks. `https://www.ibm.com/topics/convolutional-neural-networks`. (Recurso online).

[16] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey, 2018.

[17] Pytorch autograd. `https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html`. (Recurso online).

[18] Pytorch documentation. `https://pytorch.org/docs/stable/index.html`. (Recurso online).

[19] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. *International Journal of Computer Vision*, 128(7):1867–1888, March 2020.

[20] Hengkang Wang, Taihui Li, Zhong Zhuang, Tiancong Chen, Hengyue Liang, and Ju Sun. Early stopping for deep image prior, 2023.

[21] Santiago López-Tapia, Javier Mateos, Rafael Molina, and Aggelos K. Katsaggelos. Learning moore-penrose based residuals for robust non-blind image deconvolution. *Digital Signal Processing*, 142:104193, 2023.

[22] Chapter iii geometric and analytic properties of the moore-penrose pseudoinverse. In Arthur Albert, editor, *Regression and the Moore-Penrose Pseudoinverse*, volume 94 of *Mathematics in Science and Engineering*, pages 15–42. Elsevier, 1972.