Facultad de Ciencias

REVELANDO LO INVISIBLE

Revealing Invisible

Trabajo de Fin de Máster
para acceder al

MÁSTER EN CIENCIA DE DATOS

Master in Data Science

Autor: Saúl Cano Ortiz

Director\es: Francisco Matorras Weinig

Junio 2021

Agradecimientos

Para llevar a cabo el presente proyecto, propuesto por HP SCDS, perteneciente al Observatorio Tecnológico HP, he recibido la ayuda de distintas personas a las cuales me gustaría agradecer su trabajo.

Por parte de la empresa HP Solutions & Development Services (HP SCDS), me gustaría expresar mi total gratitud, en primer lugar, a **Pablo Rubio Fernández**, por su apoyo técnico y ayuda prestada; a **Alejandro Viloria Lanero**, por sus indicaciones y dirección y a **Juan Carlos Cordero González**, por la coordinación y gestión de reuniones.

Por parte de la Universidad de Cantabria (UC), me gustaría dar mi más sincera enhorabuena a mi tutor, **Francisco Matorras Weinig**, por su dedicación, esfuerzo, tiempo dedicado y apoyo, especialmente en los momentos en los que las cosas no salían.

Además, me gustaría agradecer al grupo de Computación Avanzada y al grupo *e-Science* del **Instituto de Física de Cantabria** (IFCA-CSIC-UC), por la oportunidad de emplear una de sus GPUs, que me han posibilitado disminuir los tiempos de cómputo en las fases de entrenamiento.

RESUMEN

El proyecto **Revealing Invisible**, perteneciente al Observatorio Tecnológico HP 2020/21, tiene como objetivo magnificar los cambios aparentemente imperceptibles en secuencias de vídeo. Se abordó el problema utilizando redes neuronales convolucionales (CNNs), utilizando librerías de alto nivel como **PyTorch** y **OpenCV**.

La propuesta se basa en que la CNN, se entrena con **dos imágenes de entrada** (original y original ligeramente magnificada), para dar como salida **una imagen de salida** magnificada (*groundtruth*). Para ello, no se parte de un conjunto de imágenes de entrenamiento y su respectivo conjunto de validación (imágenes magnificadas), sino que éstas se generan en caliente, de manera dinámica (**producción sintética del *dataset***). En definitiva, el **procesado de imágenes** comprende los siguientes pasos: realizar una copia de la imagen original, extraer una ventana aleatoria de la misma, aumentar su tamaño en un factor de escala y pegar dicha ventana magnificada en la copia, de manera centrada respecto a la posición de extracción. Dicho pegado se realizará mediante el método de ***alpha blending***, tomando como coeficiente el valor de la gaussiana. Por ende, las únicas diferencias entre la imagen ligeramente aumentada (entrada) y la de *groundtruth*, serían el factor de escala y el valor de la gaussiana en el borde.

Por tanto, la **red neuronal** propuesta debe **aprender** a **magnificar** dichos **cambios imperceptibles** a partir del método de procesado explicado previamente. Es decir, se entrenará y validará con imágenes, y cuando se aplique a vídeos, las secuencias de vídeo se dividiran en imágenes (*frames*), se pasaran por la red y se aunaran, para formar el vídeo magnificado.

En relación a la **optimización** de la red, se probaran distintas **arquitecturas**: capas convolucionales (ConvNets), con bloques residuales (ResNets) y bloque denso (DenseNets). A su vez, para cada arquitectura se ensayarán distintos **algoritmos de optimización** del descenso por gradiente: SGD, RMSprop y Adam. La evaluación de resultados, se realizará en base a la métrica de validación *Mean Square Error* (MSE), además de un estudio de las curvas de entrenamiento y validación. Tras escoger la estructura de la red, se confeccionará el estudio de distintos **hiperparámetros**: *learning rate*, número de capas y *weight decay*.

La decisión de la arquitectura, algoritmo de optimización e hiperparámetros óptimos, se llevará a cabo en base al estudio teórico realizado inicialmente, teniendo en cuenta criterios como *overfitting*, *vanishing gradient*, degradación de la función de coste o métricas de validación. Además, en el modelo final propuesto también se aplicarán métricas de calidad de imagen (*image quality assessment*) como PSNR, SSIM o MS-SSIM. Finalmente, el **modelo definitivo** se entrenará para un mayor número de épocas y se utilizará en la **magnificación de secuencias de vídeos**.

Palabras clave: Aprendizaje automático, Visión artificial, Procesado de imágenes.

Abstract

The **Revealing Invisible** project, part of the HP Technology Observatory 2020/21, aims to magnify seemingly imperceptible changes in video sequences. The problem was approached using convolutional neural networks (CNNs), utilising high-level libraries such as **PyTorch** and **OpenCV**.

The proposal is based on the fact that the CNN is trained with two input images (original and slightly magnified original), to produce as output a magnified image (groundtruth). For this, we do not start from a set of training images and their respective validation set (magnified images), but these are generated dynamically (**synthetic production of the dataset**). In short, **image processing** comprises the following steps: make a copy of the original image, extract a random window from it, increase its size by a scale factor and paste the magnified window into the copy in a way that is centered with respect to the extraction position. This pasting will be done using the method of **alpha blending**, taking as a coefficient the value of the Gaussian. Therefore, the only differences between the slightly enlarged image (input) and the groundtruth image would be the scale factor and the value of the Gaussian at the edge.

Therefore, the proposed **neural network** must **learn** to **magnify** such **imperceptible changes** from the processing method explained previously. That is, it will be trained and validated with images, while when applied to videos, the video sequences will be divided into images (frames), passed through the network and joined together, to form the magnified video.

In relation to the network **optimization**, different **architectures** will be tested: convolutional layers (ConvNets), with residual blocks (ResNets) and dense block (DenseNets). In turn, for each architecture, different **gradient descent optimization algorithms** will be tested: SGD, RMSprop and Adam. The evaluation of results will be carried out based on the validation metric Mean Square Error (MSE), in addition to a study of the training and validation curves. After choosing the network structure, the study of different **hyperparameters** will be carried out: learning rate, number of layers and weight decay.

The decision of the optimal architecture, optimization algorithm and hyperparameters will be made based on the initial theoretical study, taking into account criteria such as overfitting, vanishing gradient, cost function degradation or validation metrics. In addition, image quality metrics (image quality assessment) such as PSNR, SSIM or MS-SSIM will also be applied in the proposed final model. Finally, the **final model** will be trained for a larger number of epochs and will be used in the **magnification of video sequences**.

Keywords: Machine Learning, Computer Vision, Image Processing.

Índice general

Índice de figuras **7**

Índice de cuadros **9**

1. Objetivos **1**

2. Introducción **3**
2.1. ¿Por qué redes neuronales? . . . 3
2.2. Redes neuronales convolucionales . . . 7
2.3. Magnificación de vídeos . . . 9
2.3.1. ¿Cómo se magnifican los vídeos? . . . 10

3. Estudio preliminar para la caracterización de la red **12**
3.1. Arquitectura & Optimización de hiperparámetros . . . 12
3.2. Optimizers . . . 17
3.3. Métricas de validación. *Image quality assessment* . . . 20

4. Primeros pasos para la creación del modelo **23**

5. Establecimiento del modelo **29**
5.1. Recursos computacionales . . . 32
5.2. Arquitectura & *Optimizers* . . . 33
5.2.1. Capas convolucionales . . . 33
5.2.2. Capas convolucionales (bloques residuales) . . . 34
5.2.3. Capas convolucionales (bloque denso) . . . 35
5.2.4. Discusión de resultados. Establecimiento del modelo . . . 37
5.3. Optimización hiperparámetros . . . 38
5.3.1. Learning Rate . . . 38
5.3.2. Regularización L_2 . . . 39

5.3.3. Número de capas 40
5.3.4. Discusión de resultados. Perfilamiento del modelo definitivo 41
5.4. Propuesta definitiva del modelo. Resumen del análisis 41

6. Revelando lo visible de lo invisible **43**
6.1. Entrenamiento del modelo 43
6.2. Magnificación de secuencias de vídeo 45

7. Conclusión **49**

Bibliografía **50**

Apéndice A. Distribución de tareas **55**

Índice de figuras

2.1. Representación didáctica para la comprensión de la terminología de las redes neuronales. 4
2.2. Representación didáctica de Fig. 2.1 con notación matemática. 4
2.3. Representación gráfica de la operación convolucional [49]. 7
2.4. Resultados de la magnificación de vídeo de un recién nacido, donde el latido real es de 152 bpm (derecha) y el modelo produce un pico realmente cercano [3]. 10

3.1. Esquematización de redes convolucionales con bloque residual [15]. 13
3.2. Esquema didáctico que refleja la diferencia matemática entre las arquitecturas *ResNet* y *DenseNet* [61]. 16

4.1. Pasos 1 y 2. Extracción de ventana aleatoria y magnificación de la misma. . 25
4.2. Ventana ampliada con máscara gausiana. 25
4.3. Resultado final. *Groundtruth* de la imagen de estudio. 26
4.4. Representación esquemática de la ecuación de composición o α-blending: $C = (1-\alpha)B + \alpha F$ [71]. 27
4.5. Representación esquemática de la metodología para la generación del conjunto de imágenes amplificadas. 28

5.1. Representación gráfica de la arquitectura I. Capas convolucionales [45]. . . 31
5.2. Representación gráfica de la arquitectura II. Redes convolucionales con bloques residuales [45]. 31
5.3. Representación gráfica de la arquitectura III. Redes convolucionales con bloque denso [45]. 32
5.4. Representación de los problemas de memoria en local (superior), solventados con la GPU brindada por el IFCA (inferior). 32

5.5. MSE en función del número de épocas. La figura de la izquierda se corresponde a Adam, la central a SGD y la de la derecha a RMSprop para arquitectura tradicional de capas convolucionales. 33
5.6. MSE en función del número de épocas. La figura de la izquierda se corresponde a Adam, la central a SGD y la de la derecha, a RMSprop para arquitectura de bloques residuales. 34
5.7. MSE en función del número de épocas. La figura de la izquierda se corresponde a Adam, la central a SGD y la de la derecha, a RMSprop para arquitectura de bloque denso. 36
5.8. MSE en función del número de épocas para arquitectura de bloque denso, optimizador Adam ($\beta_1 = 0{,}9$, $\beta_2 = 0{,}99$) para el *learning rate* óptimo, 0.001. 39

6.1. MSE en función del número de épocas (300) para el modelo propuesto con arquitectura de bloque denso (9 capas convolucionales), optimizador Adam, $\gamma = 0{,}001$, sin regularización. 43
6.2. Representación de una de las imágenes en test para comprobar las métricas de calidad de imagen, donde se remarca la ventana ampliada, en la imagen original (izqda.) y magnificada (dcha.). 45
6.3. Comparativa grabación (real vs. red) vibración de la muñeca. 46
6.4. Comparativa grabación (real vs. red) de vela encendida. 46
6.5. Comparativa grabación (real vs. red) de taza humeante. 47
6.6. Comparativa grabación (real vs. red) de planta bajo la lluvia. 47
6.7. Comparativa grabación (real vs. red) de cuerda de mascarilla vibrando. . . . 48

Índice de cuadros

3.1. Operación matemática implementada en las respectivas arquitecturas propuestas de estudio 16

5.1. Valores óptimos de la función de coste en las fases de entrenamiento y validación para la arquitectura de capas convolucionales. 33
5.2. Valores óptimos de la función de coste en las fases de entrenamiento y validación para la arquitectura de capas convolucionales con bloques residuales. 35
5.3. Valores óptimos de la función de coste en las fases de entrenamiento y validación para la arquitectura de capas convolucionales formando un bloque denso. 36
5.4. Resumen de los resultados obtenidos de MSE óptimo en *training* y *validation* para los distintos optimizadores y las distintas arquitecturas propuestas. . . 37
5.5. Función de coste MSE (*training* & *validation*) para arquitectura de bloque denso y *optimizer*, Adam en función del *Learning Rate*. 38
5.6. Función de coste MSE (*training* & *validation*) para arquitectura de bloque denso y *optimizer*: Adam, $\gamma = 0.001$, en función del *weight decay* para regularización L_2. 40
5.7. Función de coste MSE (*training* & *validation*) para arquitectura de bloque denso y *optimizer*: Adam, $\gamma = 0.001$, en función del número de capas. . . . 40

6.1. Función de coste MSE (*training* & *validation*) para la propuesta de modelo. 43
6.2. Tabla de métricas de calidad de imágen para un total de 30 imágenes. . . . 44

Capítulo 1

Objetivos

La magnificación de secuencias de vídeo aparentemente sin movimiento permite apreciar pequeñas alteraciones invisibles para el ojo del ser humano. El estado del arte de las técnicas de magnificación del movimiento se fundamenta en el uso de filtros diseñados a mano. Sin embargo, dichas metodologías no siempre dan resultados óptimos, pudiendo causar ruido o emborronamiento en las imágenes aumentadas.

El interés actual en las técnicas de magnificación en secuencias de vídeo surge a raíz de su gran aplicabilidad, que va desde el análisis estructural de edificios o la vibrometría hasta su empleo en el análisis médico (repiratorio, cardiovascular) o la deteccción de microexpresiones faciales.

El principal objetivo del proyecto es proponer una estrategia innovadora donde no se construirán filtros manualmente, sino que se aprenderán los filtros directamente de las propias imágenes mediante redes neuronales convolucionales.

Dado que es poco habitual encontrar secuencias de vídeo estáticas con su correspondiente vídeo magnificado, se describirá en el capítulo 3, la propuesta para la generación sintética de imágenes magnificadas.

Posteriormente, se estudiará el diseño de la red. Para ello, se plantearán tres arquitecturas distintas basadas en capas convolucionales, capas convolucionales con bloques residuales y capas convolucionales formando un bloque denso. Además, para cada arquitectura se probarán tres algoritmos de optimización: SGD, RMSprop y Adam.

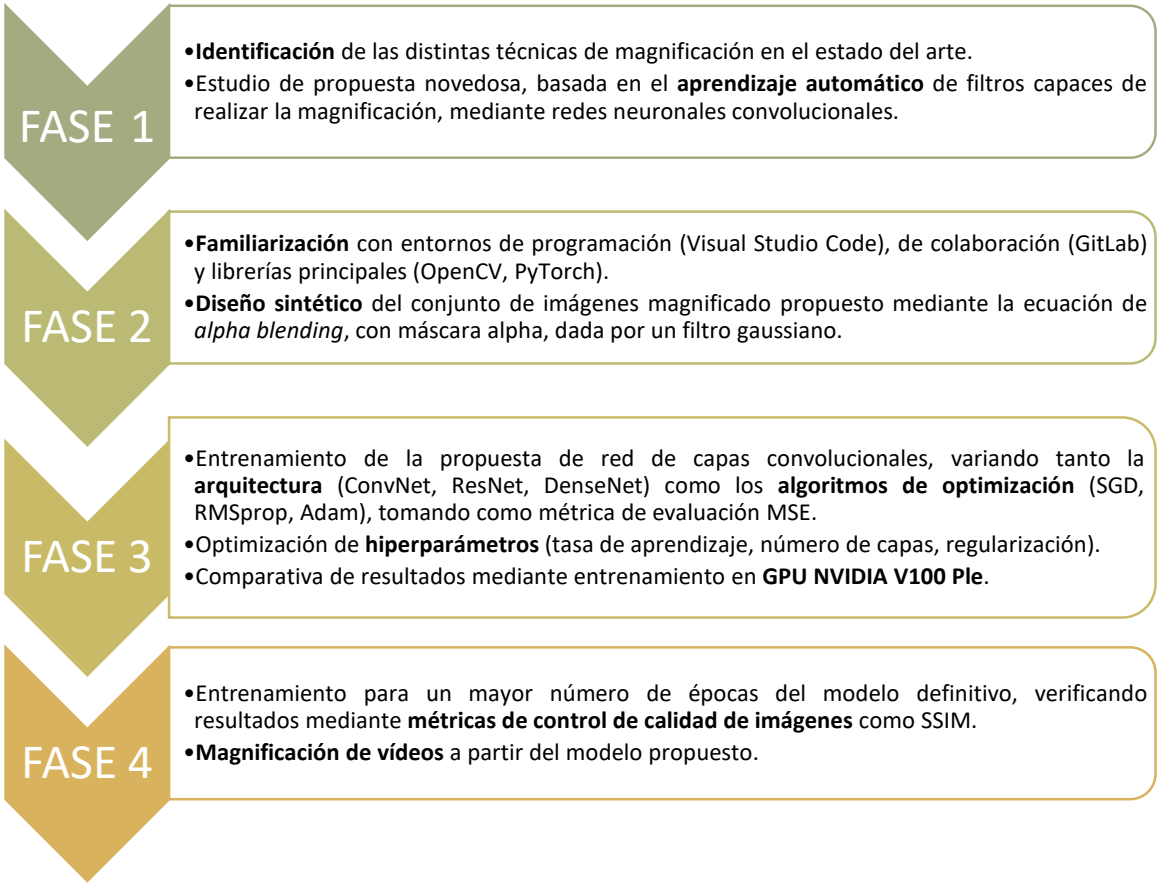
La programación del código se ha realizado en el entorno Visual Studio Code en lenguaje Python. Para ello, se han empleado librerías de alto nivel como PyTorch, para el aspecto del diseño y entrenamiento de la red, y OpenCV, para la manipulación de imágenes. Los archivos se han subido a un *docker* con acceso a una GPU, para realizar los entrenamientos de manera más eficiente. Para la revisión de código, distribución de tareas y el almacenamiento

de archivos, se ha empleado GitLab. El código junto con su explicación está disponible en un repositorio en GitHub.

Tras evaluar resultados en imágenes en base a distintas métricas como MSE o SSIM, se han realizado pruebas para secuencias de vídeo. Se obtienen resultados satisfactorios, siempre que existe un único objeto cuyo movimiento es muy sutil.

Por tanto, el presente proyecto pretende promover el uso de redes neuronales convolucionales para magnificar secuencias de vídeo, como técnica de interés actual.

A continuación, se muestran los objetivos alcanzados en las siguientes fases o paquetes de trabajo. Para una mayor explicación de la distribución de trabajo, se puede consultar el apéndice A.



Capítulo 2

Introducción

El proyecto Revealing Invisible pretende abordar el problema de la magnificación de vídeos sin movimiento aparente. Para ello, se propondrá el uso de redes neuronales convolucionales. Con el objetivo de permitir la comprensión de los procedimientos llevados a cabo en este trabajo, se introducirán las redes neuronales tradicionales a fin de desarrollar un subtipo de las mismas, las redes neuronales convolucionales, que se emplearán como topología fundamental para la aplicación del proyecto: la magnificación.

2.1. ¿Por qué redes neuronales?

Las redes neuronales fueron propuestas por primera vez en 1944 por Warren McCullough y Walter Pitts [36]. La extracción de conocimiento a partir de ellas ha supuesto un gran avance multidisplinar [6, 31]. Pero, ¿qué son las redes neuronales?

Una red neuronal es un conjunto interconectado de unidades de procesamiento simples (neuronas, nodos o células), comunicadas entre sí, donde la capacidad de procesamiento de la red o intensidad de la conexión viene definida por pesos [23]. Para introducir la terminología, se empleará el ejemplo propuesto en [69] (Fig. 2.1, Fig. 2.2).

En este estudio se parte de un conjunto de datos que relaciona la eficacia y la cantidad de dosis de un medicamento. Así pues, el objetivo es aprender una función que indique cuál es la eficacia dada una dosis concreta. Por tanto, se construye una red neuronal como la de Fig. 2.2, donde la neurona de entrada recibe información sobre la cantidad de dosis y la neurona de salida debe ser capaz de predecir la eficacia de la misma.

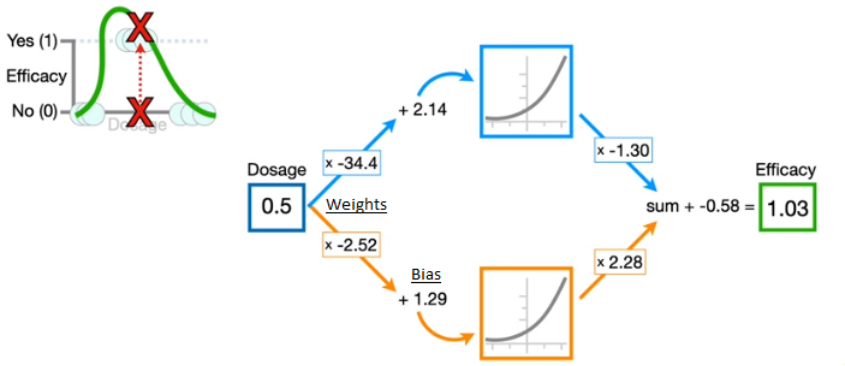


Figura 2.1 Representación didáctica para la comprensión de la terminología de las redes neuronales.

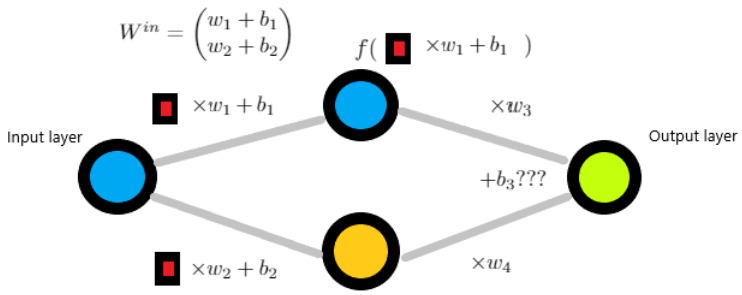


Figura 2.2 Representación didáctica de Fig. 2.1 con notación matemática.

Estructuralmente, entre ambas neuronas se encuentra una capa de neuronas o, más bien, una capa oculta formada por dos neuronas[1]. Las neuronas[2] reciben como entrada los valores correpondientes ponderados o escalados mediante pesos w_{jk} (*weights*), es decir, la conexión entre la neurona i de una capa y la neurona j de otra, viene definida a nivel matemático por un peso, w_{ij}. Además, se suma una cantidad denominada sesgo θ_k (*bias*). Por otra parte, la información que sale de cada neurona es el valor de entrada que recibirá de una o varias neuronas consecutivas tras aplicarle una función, denominada función de activación[3]. En

[1] La imagen 2.2 representa un sistema sencillo, compuesto por una capa de entrada y otra de salida, no obstante, entre ambas pueden existir múltiples capas ocultas (*hidden layers*), donde cada capa puede tener distinto número de neuronas. Cuanto mayor sea el número de capas, mayor será la complejidad de la red neuronal.

[2] En este contexto, las neuronas de las capas ocultas y la capa de salida

[3] Hay diversas funciones de activación: sigmoidal, tangente hiperbólica (problemas de clasificación binaria), *softmax* (clasificación multiclase), RELU, LeakyRelu...etc. Lo más importante es que deben ser escogidas en función de su aplicación.

resumen, cada neurona recibe una entrada de las neuronas vecinas y la utiliza para computar una salida, propagando la información. A nivel matemático [41]:

$$s_k(t) = \sum_j w_{jk}(t) y_j(t) + \theta_k(t) \longrightarrow F_k(s_k(t)) \tag{2.1}$$

donde $s_k(t)$ es la información ponderada y F_k, la función de activación no lineal.

En consecuencia, si se conocen los pesos y sesgos, se puede trazar una arquitectura como la propuesta en ejemplo anterior, que permitirá obtener una función predictora (gráfica verde en la figura 2.1). Pero, otra cuestión muy importante es, ¿cómo se obtienen dichos pesos y sesgos óptimos que dan lugar a la construcción de la función? La respuesta reside en el método de *backpropagation*.

En primer lugar, antes de abordar el concepto de *backpropagation*, es importante recalcar el concepto de función de coste o función error [32] (*loss function*). La red será entrenada con un conjunto de datos de entrenamiento, donde se conoce tanto la dosis como la eficacia (datos observados), con el objetivo de probar la red neuronal entrenada en un conjunto de datos independiente de los utilizados en el entrenamiento (conjunto de validación), donde solo se conoce la dosis y se debe predecir la eficacia (datos predichos). Por consiguiente, el objetivo es encontrar la configuración de la red neuronal que mejor se adapte a los datos, maximizando su capacidad de generalización. Por tanto, la función de coste[4] es una función que relaciona los valores observados con los predichos. Cuanto más óptimos sean los parámetros[5], menor será el valor de la función error.

Es decir, para encontrar el valor de los parámetros, es necesario minimizar la función de coste (problema de optimización). Para encontrar el mínimo es realmente útil el empleo del descenso por gradiente. Consecuentemente, el *workflow* de las redes neuronales consiste en:

- En primer lugar, se inicializan los parámetros de manera aleatoria.
- Posteriormente, se aplica la regla de la cadena, calculando las derivadas de la función de coste, con el objetivo de alcanzar su mínimo, optimizando así los parámetros (*backpropagation*).
- Finalmente, se actualizan los pesos en base al cálculo de derivadas (*gradient descent*)

En el ejemplo propuesto anteriormente, suponiendo que el parámetro desconocido es b_3 (Fig. 2.2), entonces para una función de coste tipo MSE[6], se tendría que:

[4] De nuevo, hay múltiples funciones de coste como MSE (*Mean Squared Error*), L_1,L_2...
[5] El término parámetros se refiere al conjunto de pasos y sesgos
[6] *Mean Squared Error, una métrica de validación.*

$$\mathscr{L} = \sum (observed_i - predicted_i)^2 \tag{2.2}$$

$$\Delta w = \frac{d\mathscr{L}}{db_3} = \frac{d\mathscr{L}}{dpredicted} \frac{dpredicted}{db_3} \tag{2.3}$$

$$w = w + \gamma \cdot \Delta w \tag{2.4}$$

donde γ se denomina tasa de aprendizaje (*learning rate*) y es un hiperparámetro a optimizar. Por tanto, el método de *backpropagation* (ec. 2.3) consiste en realizar la regla de la cadena hacia atrás (cálculo del descenso por gradiente), propagando el error y actualizando los pesos (ec. 2.4), con el objetivo de minimizar la función de coste (ec. 2.2).

En conclusión, a partir del ejemplo didáctico, se puede concluir que una red neuronal es un conjunto de capas (entrada, ocultas y salida) formadas por neuronas, que se conectan entre sí, mediante unos parámetros (pesos y sesgos). Dichos parámetros son desconocidos (inicialmente generados aleatoriamente) y la manera de hallarlos consiste en la minimización de la función de coste, mediante el método de *backpropagation*.

Por tanto, es realmente interesante modificar o adaptar los hiperparámetros, con el objetivo de maximizar la funcionalidad de la red neuronal, variando el número de capas, el número de neuronas por capa, el *learning rate*, las funciones de activación, etc. Además, para tratar de mejorar la capacidad de generalización de la función de coste, se puede añadir un término de regularización[7]. Por ejemplo, la regularización L_2 [75], que viene dada por la siguiente expresión:

$$\mathscr{L} = \sum \mathscr{L}_i + \beta ||w||_2^2 \tag{2.5}$$

donde el primer término ya había sido presentado (MSE en el ejemplo didáctico) y su principal cometido es ajustar el modelo lo máximo posible a los datos. El término de regularización, por su parte, procura hacer los pesos lo menor posibles, reduciendo así la complejidad de la red. Es decir, el término de regularización surge como solución ante un posible sobreajuste (*overfitting*) en el entrenamiento, dando lugar a una alta varianza en el dataset de validación (pérdida de generalización).

Además, existen distintas modificaciones o mejoras sobre el descenso por gradiente. Dichos algoritmos mejorados se denominan optimizadores (*optimizers*). En el apartado 3.2, se describirán los más conocidos, que serán utilizados *a posteriori* en el análisis y resultados. A continuación, se analizan las redes neuronales convolucionales, subtipo de las redes neuronales artificiales (ANNs, por sus siglas en inglés).

[7] Hay distintos tipos de regularización como la norma L_2 o L_1 o *dropout*.

2.2. Redes neuronales convolucionales

Las redes convolucionales (*convolutional neural networks, CNNs*) [44] ofrecen una arquitectura eficiente para extraer las características altamente significativas a gran escala en problemas de alta dimensionalidad [14]. Pero, ¿en qué consiste una red convolucional? ¿En qué se diferencia respecto a las redes neuronales tradicionales descritas previamente[8]?

Para comprender los conceptos de las CNNs, en primer lugar se deben distinguir los distintos elementos que las conforman [63] :

- **Capa convolucional**. La convolución es un tipo especializado de operación lineal, utilizada para la extracción de características, en la que una pequeña matriz de números, llamada *kernel*, se aplica a la entrada, que es otra matriz numérica (tensor) . Se realiza un producto elemento a elemento (Fig. 2.3) entre cada elemento del núcleo y el tensor de entrada y se suma para obtener el valor de salida, en la posición correspondiente del tensor de salida, llamado mapa de características (*feature map*).

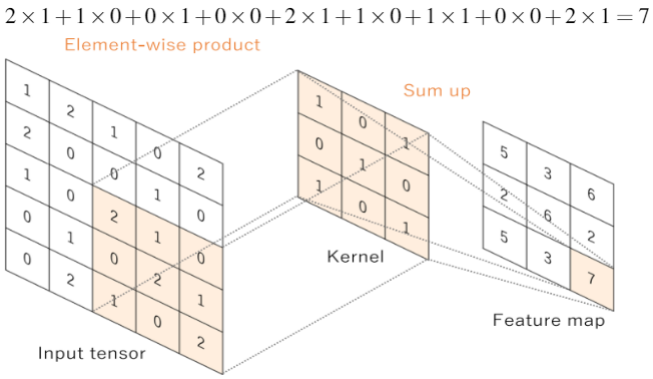


Figura 2.3 Representación gráfica de la operación convolucional [49].

- **Hiperparámetros**. La operación convolucional reduce las dimensiones del tensor de salida respecto al de entrada. Para ello, se emplea la técnica de *padding*, es decir, se añaden filas y columnas con el objetivo de mantener la dimensión. La distancia de actuación del kernel se denomina *stride*. En la arquitectura empleada en el proyecto, se han tomado valores de padding $p = 1$ (donde las filas se han rellenado de ceros, *zero-padding*) y $s = 1$, desplazando el filtro (*kernel*) de 1 en 1 [51]. Otros hiperparámetros a considerar son el número de *kernels* y el tamaño de los mismos.

[8] Las redes neuronales usuales también se denominan *fully-connected*

- **Función de activación no lineal**. Por último, al resultado de la operación convolucional, se le aplica una función de activación. En este proyecto, se ha aplicado ReLU (*Rectified Linear Unit*).

  Así pues, el proceso de entrenamiento de una red formada por capas convolucionales, no es más que la optimización u obtención de los valores de los *kernels* que mejor se adaptan para una tarea determinada a partir de los datos de entrenamiento (en este trabajo, imágenes, es decir, tensores con valores de 0 a 255). Entonces, valores como el tamaño de *padding*, *stride*, dimensión y número de filtros, son hiperparámetros a establecer en el proceso de entrenamiento. A nivel matemático, lo explicado previamente, se resume a [21]:

  $$a^l_{i,j,k} = a((w^l_k)^T x^l_{i,j} + b^l_k) \tag{2.6}$$

  donde el valor del píxel en la posición (i, j) en el k-mapa de características de la capa l es igual a la función de activación no lineal a, aplicada a la operación convolucional sobre el vector de parámetros (pesos w^l_k y sesgos b^l_k).

Pero, ¿cuáles son las principales razones para escoger redes convolucionales (ConvNets) en vez de redes densas (fully-connected) para tratar imágenes? [46]

- Las redes densas aprenden patrones globales en el espacio de características (*feature space*), utilizando todos los píxeles en el caso de imágenes. Sin embargo, las redes convolucionales aprenden patrones locales tras aplicar filtros bidimensionales, sin considerar todos los píxeles en el caso de imágenes.

- Los patrones que aprenden las redes convolucionales son invariantes frente a translaciones, es decir, si una *ConvNet* aprende un patrón en la esquina superior derecha de la imagen, lo reconocerá en cualquier otra región de la imagen. En contraposición, las redes densas tendrían que reaprender el patrón si variara su posición [1].

- Las capas convolucionales aprenden una jerarquía espacial en las imágenes, reconociendo las características o patrones más básicos en las capas iniciales y los conceptos más complejos en las capas más profundas. Por esa razón, permite aplicar *Transfer Learning* para datos de entrada similares tomando tan solo las capas iniciales y aplicando las capas específicas a nuestro problema. Por tanto, en una escala de conectividad y complejidad, las CNN están en el extremo inferior.

En este proyecto se emplearán arquitecturas con capas convolucionales, debido a la gran cantidad de parámetros que se manejan al trabajar con imágenes. Además, revisando

la literatura, las capas convolucionales son ampliamente utilizadas para el procesado de imágenes (*image processing*), como sucede en este caso.

Tras definir los conceptos fundamentales de las redes neuronales, profundizando en las redes neuronales convolucionales, se explicará su aplicación a la magnificación de vídeos en base al estado del arte.

2.3. Magnificación de vídeos

La magnificación del movimiento (*motion magnification*) es un término empleado para referirse a la ampliación de movimientos sutiles o aparentemente estáticos. Dicho término fue acuñado en 2005, por parte de investigadores del *Massachusetts Institute of Technology* (MIT), donde se comenzó realizando un análisis en la escala del píxel, ampliando suavemente el movimiento que el ser humano no podía observar de manera natural.

La magnificación del movimiento aplicado a secuencias de vídeo se basa en descubrir las variaciones temporales supuestamente inexistentes frente a la captura de la cámara o ante el ojo humano. Es decir, la ampliación o magnificación del vídeo, actúa como si se tratara de un microscopio que permite apreciar un movimiento imperceptible de manera visual, logrando caracterizar dichas deformaciones espaciotemporales. Es decir, la magnificación del video es una metodología desarrollada para contemplar y analizar versiones exageradas de pequeños desplazamientos.

¿Cuál es el objetivo de la magnificación de secuencias de vídeo?¿Qué posibles aplicaciones tiene? Un ejemplo es la vibrometría visual. Existe una conexión teórica entre los fundamentos de la mecánica de las vibraciones de los materiales y las técnicas de visión artificial que permite inferir las propiedades de los materiales a partir de pequeños movimientos a menudo imperceptibles en el vídeo. Por tanto, considerando la geometría del material, conocida y fija, puede utilizarse la magnificación para extraer las frecuencias de los modos de vibración, infiriendo así sus propiedades intrínsecas [13].

Otro ejemplo es la medición del ritmo cardíaco. En [3] se describe una aproximación que no necesita contacto, a diferencia del electrocardiógrafo, que permite extraer la frecuencia cardíaca a partir de los cambios sutiles de la secuencia de vídeo del pulso. Para ello, se emplea una combinación de filtrado de frecuencias y PCA[9], que permite identificar la componente de movimiento asociada al pulso, extrayendo los picos para identificar los latidos individuales.

El reconocimiento de expresiones faciales ha sido intensamente estudiado, principalmente por la comunidad de psicólogos para el reconocimiento de patrones. El objeto de investigación más reciente es el problema de reconocer emociones difícilmente apreciables, lo que se

[9] *Principal Component Analysis*

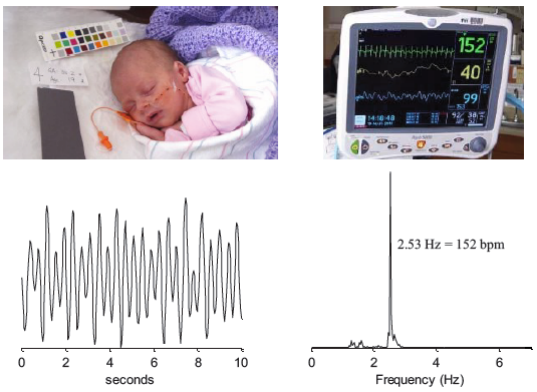


Figura 2.4 Resultados de la magnificación de vídeo de un recién nacido, donde el latido real es de 152 bpm (derecha) y el modelo produce un pico realmente cercano [3].

denominan microexpresiones[10]. La dificultad de reconocer microexpresiones subyace en que su exhibición se da durante un breve periodo de tiempo e implican cambios espaciales minúsculos. Así pues, en el el artículo [79], se propone una metodología para clasificar estos gestos.

En definitiva, existen múltiples aplicaciones como el análisis de comportamiento estructural de los edificios [9], el estudio de la dinámica de las grúas frente a fuertes rachas de viento [78] o las variaciones oculares en sus estudios ópticos [18].

2.3.1. ¿Cómo se magnifican los vídeos?

Las técnicas relativas a la magnificación de vídeo pueden clasificarse en dos categorías: lagrangiana y euleriana. La aproximación lagrangiana extrae directamente el campo de movimiento[11] (*motion field*) para modificar el movimiento de los pixels mientras que la aproximación euleriana descompone los frames de los vídeos, posibilitando su manipulación.

La aproximación euleriana, a su vez, se descompone en tres etapas: la descomposición de frames a una representación alternativa, la manipulación de dicha representación y la reconstrucción de las representaciones alteradas en una secuencia de frames magnificados.

El estado del arte de las técnicas de magnificación de vídeo actuales sugiere el uso de filtros diseñados manualmente (*hand-designed filters*) [66], no siempre óptimos, dado que surgen problemas con el ruido o la creación de artefactos[12]. Por otra parte, las técnicas

[10] Las microexpresiones faciales son expresiones involuntarias, sutiles y rápidas, que tienen aproximadamente una duración de $1/3$ de segundo [17].

[11] En visión artificial, cada punto (y_1, y_2) de la imagen es la proyección de algún punto de la escena 3D de la secuencia de video, pero la posición de la proyección de un punto fijo en el espacio puede variar en el tiempo. Entonces, el campo de movimiento, se define como la derivada temporal de la posición de la imagen.

[12] Distorsión notable debido a métodos de procesado de imágenes.

actuales relativas a la aproximación euleriana, no tienen en cuenta fenómenos como la oclusión [13].

Por tanto, la propuesta definitiva para el desarrollo de la magnificación del video en el presente proyecto, se fundamenta en la novedosa idea propuesta en [50]. Se trata de una técnica de magnificación perteneciente al marco euleriano, totalmente distinta a los habituales filtros diseñados, con tal de evitar esa tendencia de generación de ruido o emborronamiento excesivo (*blurring*). Se trata de un método que aprende los filtros directamente de los datos. Para ello, se genera un conjunto de imágenes sintético, ya que es realmente difícil encontrar una secuencia de vídeo y su respectiva secuencia de vídeo ligeramente magnificada. Para ello, se emplea una arquitectura de capas convolucionales basada en la topología propuesta en [83], donde la red recibe dos imágenes, la imagen original y la misma imagen ligeramente amplificada en alguna región de la misma. La CNN da lugar a la misma imagen original magnificada como *groundtruth*, solo que el factor de magnificación o escala, es superior al de la imagen magnificada de entrada.

Cabe destacar que las principales ideas del proyecto, tanto el *background* de metodologías de filtros diseñados a mano (*hand-crafted filters*), como los nuevos estudios donde se aprenden dichos filtros a partir de CNNs, se han extraído principalmente de las diversas publicaciones del área de estudio *Video Magnification* en el MIT Computer Science and Artificial Intelligence Laboratory.

Con lo cual, el proyecto Revealing Invisible pretende emular la idea propuesta por el estudio mencionado previamente pero realizando un procesado de imágenes distinto para que la red de capas convolucionales aprenda a realizar la tarea de magnificación. Además, también se realizará un estudio alternativo en cuanto al enfoque de la arquitectura, algoritmos de optimización e hiperparámetros, dando lugar a resultados prometedores.

[13] Se produce cuando dos objetos están cerca y parece que se fundan visualmente.

Capítulo 3

Estudio preliminar para la caracterización de la red

En este apartado se desarrolla a nivel teórico las distintas arquitecturas, algoritmos de optimización, hiperparámetros y métricas utilizados para la búsqueda del modelo capaz de realizar la tarea de magnificación.

3.1. Arquitectura & Optimización de hiperparámetros

La arquitectura de una red neuronal es la topología, estructura o patrón de conexionado de una red neuronal [54]. Es decir, la estructura de una red neuronal no es solo el número de capas neuronales, sino también el tipo de conexiones, de neuronas e, incluso, la manera en la que son entrenadas [68].

En la literatura de redes neuronales tenemos múltiples aplicaciones de las mismas, desde redes neuronales para predicción de diagnóstico médico [70], hasta segmentación semántica [53]. Pero, lo más útil, es que, independientemente del proyecto o finalidad de desempeño del sistema neuronal, comúnmente se suele detallar la arquitectura y diseño final en base a unos resultados (métricas de validación). Para el vigente proyecto se ha decidido estudiar tres diseños de arquitecturas: capas convolucionales[1], capas convolucionales con bloques residuales y capas convolucionales formando un bloque denso.

Residual Convolutional Networks

A nivel conceptual, a medida que se aumenta el número de capas convolucionales, más profundo es el modelo, puesto que cada capa añade un mayor nivel de abstracción de características y, por tanto, más niveles de características deberá aprender la red. Ahora

[1]Para esta arquitectura no se van a proporcionar más detalles ya que han sido explicadas con claridad previamente.

bien, un mayor número de capas no implica necesariamente un mejor resultado [73]. Al incrementar el número de capas surgen inconvenientes, como el aumento de la complejidad del sistema (mayor tiempo de cómputo) o el incremento del *overfitting* (sobreajuste o pérdida de generalización), produciéndose el denominado *vanishing gradient* [4].

En [25], se muestra a nivel teórico-matemático el decrecimiento exponencial del gradiente respecto al número de épocas o iteraciones (*error flow scaling factor*). Tal y como se ha explicado en apartados anteriores, para actualizar los pesos se debe calcular el gradiente de la función de coste, con el objetivo de encontrar los parámetros óptimos. Así pues, al aplicar la regla de la cadena, como se trata de un producto de distintos elementos, si bien uno se aproxima a cero, el gradiente se aproximará también a cero y no se dará ninguna actualización en los pesos. Adicionalmente, se comprueba teóricamente que el hecho de aumentar el número de capas o la complejidad del sistema no necesariamente implica mejores resultados.

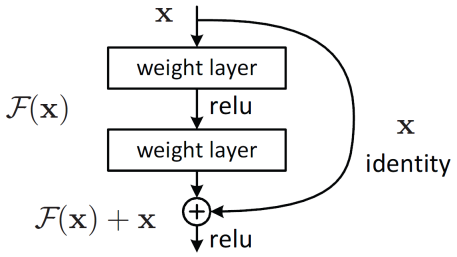


Figura 3.1 Esquematización de redes convolucionales con bloque residual [15].

Las redes convolucionales basadas en bloques residuales [72] surgieron tras su presentación por parte de *Microsoft*, batiendo récords en clasificación, localización, detección de objetos, disminución de la complejidad, etc. Tras diversos estudios, se ha demostrado que el mero hecho de apilar capas convolucionales, aumentando así tanto la complejidad como la profundidad de la red, no siempre implica una mejora en el rendimiento de la misma [16]. En realidad, al aumentar la cantidad de capas, surgen inconvenientes como el desvanecimiento del gradiente [2] y la maldición de la dimensionalidad [3] (*curse of dimensionality*). Del mismo modo, a medida que aumenta la profundidad, en un instante dado la precisión se paraliza y comienza a degradarse, de tal manera que las redes convolucionales más profundas no necesariamente aprenden mejor [24]. Así pues, dichos acontecimientos propician el nacimiento de los bloques residuales en las redes convolucionales (*ResNets*).

[2] En algunos casos, el gradiente se irá desvaneciendo a valores muy pequeños, impidiendo la actualización de los pesos. En el peor caso, esto puede impedir que la red neuronal continúe su entrenamiento.

[3] Expresión de todos los fenómenos que aparecen con los datos de alta dimensionalidad y que tienen la mayoría de las veces consecuencias desafortunadas en el comportamiento y los rendimientos de los algoritmos de aprendizaje. [77, 58]

Es decir, mientras que las redes convolucionales tradicionales conectan la capa l con la $l+1$ mediante la ecuación $x_{l+1} = H_{l+1}(x_l)$, las *ResNets* añaden una conexión de salto que evita las transformaciones no lineales, con una función identidad. El resultado se muestra en la siguiente expresión[4]:

$$x_{l+1} = H_{l+1}(x_l) + x_l \tag{3.1}$$

Una ventaja de las *ResNets* es que el gradiente puede fluir directamente a través de la función de identidad desde las capas posteriores a las capas anteriores. Sin embargo, la función de identidad y la salida de $H(\cdot)$ se combinan mediante la suma, lo que puede impedir el flujo de información en la red.

Como se puede observar en Fig. 3.1, el efecto de las *ResNets* consiste en realizar un "salto"de capas, es decir, se aumenta el número de capas introduciendo dicha conexión residual (capa identidad). Si se tiene como entrada x, tras aplicar un par de capas convolucionales, la tercera capa convolucional recibe $h(x) = f(x) + x$. Por tanto, las ventajas de dicha arquitectura respecto a la tradicional de capas convolucionales son:

- El bloque residual añade una pequeña alteración para conseguir una representación ligeramente distinta, mientras que en las CNNs tradicionales cuando se pasa de x a $f(x)$ no existe ninguna relación. Así pues, autores como [16] afirman que es más sencillo optimizar el mapa residual.

- En el desarrollo del método de *backpropagation*, se evita que se desvanezca el gradiente, mediante las operaciones de suma, que logran balancear los resultados del propio gradiente.

Dense connected convolutional networks

Por último, otra posible alternativa en la elección de la arquitectura de la red neuronal son las denominadas redes convolucionales de bloque denso (*DenseNets*). De hecho, son aplicadas actualmente en diversos campos, obteniendo resultados óptimos respecto a otras arquitecturas [39, 52].

Las DenseNet conectan cada capa con sus anteriores. Mientras que las redes convolucionales tradicionales con L-capas, poseen L-conexiones, las redes convolucionales de bloque denso tienen $\frac{L(L+1)}{2}$ conexiones directas [29]. Para cada capa, los mapas de caracteríticas de todas las capas anteriores se utilizan como entradas en todas las capas posteriores.

Así pues, las ventajas que proporcionan son: reducción o alivio, al igual que ResNet, del problema del desvanecimiento del gradiente; magnificación o mejora de la propagación de

[4]El operador H_l representa la operación de convolución

características; fomento de la reutilización de características; disminución notoria del número de parámetros.

Por consiguiente, las principales diferencias o mejoras respecto a las *ResNets* son la disminución del número de parámetros debido al menor número de conexiones y el mayor flujo de características entre capas.

Para garantizar el máximo flujo de información entre capas convolucionales, la idea propuesta en [30] (ganadora del *best paper award* en CVPR2017) muestra como cada capa obtiene entradas adicionales de todas las capas precedentes, y entrega sus propios mapas de características a todas las capas posteriores. Es decir, el input de cada capa convolucional no es más que la concatenación de todos los mapas de características (*feature maps*) generados por las capas anteriores para un bloque denso dado. Entonces, si bien la salida de la capa $(l-1)$ se denota como x_{l-1}, la capa de salida de la capa l-ésima, tal y como se describe en [86], es:

$$x_l = H_l([x_0, x_1 ... x_{l-1}]) \tag{3.2}$$

donde el operador $H_l(\cdot)$ representa la secuencia de transformaciones consecutivas.

Por otra parte, es importante recalcar uno de los principales inconvenientes computacionales de DenseNet: el requerimiento de memoria[5]. Para ello, se plantean arquitecturas alternativas como *CondeseNet* [28]. Como curiosidad, también son ampliamente utilizadas combinaciones de *ResNet* con *DenseNet* para clasificación de imágenes o detección de objetos.

En base al estudio sobre la literatura relacionada con las tres arquitecturas propuestas, podemos destacar los siguientes aspectos:

- Una arquitectura con capas convolucionales no tiene por qué mejorar su eficiencia a partir del incremento de capas convolucionales ya que, más allá de ciertos problemas como el aumento de la complejidad de la arquitectura o el incremento del tiempo de cómputo, surge además el principal problema: el desvanecimiento del gradiente. Para solventarlo, surgen arquitecturas como las redes convolucionales con bloques residuales (tipo *ResNets*) o de bloque denso (tipo *DenseNets*).

- A nivel matemático, las únicas diferencias entre *ResNet* y *DenseNet* son los valores de entrada que recibe el operador $H_l(\cdot)$, que son sumados y concatenados, respectivamente. Dicha discrepancia matemática da lugar a las siguientes diferencias:

[5]Para programar el diseño de las imágenes de entrada y salida, se realizó en local mediante *Visual Studio*, mientras que la ejecución del programa se realizó en un *docker* que hacía uso de una GPU del Instituto de Física de Cantabria (IFCA), agilizando los procesos de cálculo

Arquitectura	***Equation***
ConvNet	$x_l = H_l(x_{l-1})$
ResNet	$x_l = H_l(x_{l-1}) + x_{l-1}$
DenseNet	$x_l = H_l([x_0, x_1 \ldots x_{l-1}])$

Cuadro 3.1 Operación matemática implementada en las respectivas arquitecturas propuestas de estudio

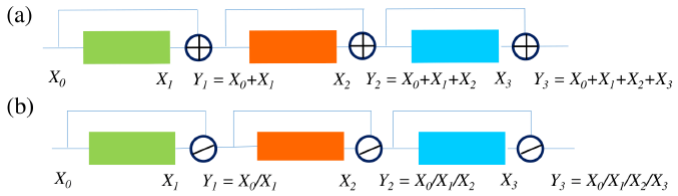


Figura 3.2 Esquema didáctico que refleja la diferencia matemática entre las arquitecturas *ResNet* y *DenseNet* [61].

a) **Compacidad del modelo/ Extracción de características**. Las *Resnets* son fáciles de diseñar ya que tan solo introducen esa suma de la identidad (*identity shortcut*). El inconveniente de la identidad es que provoca el problema de colapso del dominio, que reduce la capacidad de aprendizaje de la red [56]. Para el caso de las redes convolucionales densas, como consecuencia directa de la concatenación de los mapas de características, cualquier capa arbitraria puede ser accedida mediante cualquier capa posterior, es decir, las *DenseNets* posibilitan la reutilización de características, dando lugar a modelos más compactos [29].

b) **Parámetros/ Requerimiento de memoria**. Las *DenseNets* evitan realizar dicha operación de suma y, consecuentemente, preservan las características de las capas precedentes. Sin embargo, tienen un gran inconveniente, pues necesitan grandes requerimientos de memoria GPU, debido a las operaciones de concatenación, incrementando así el tiempo de entrenamiento de la red [85]. Sin embargo, *ResNets* mediante dicha operación suma, logran disminuir el número de parámetros y, consecuentemente, el tiempo de cómputo durante la fase de entrenamiento.

La optimización de hiperparámetros es el problema de minimizar una función de pérdida o coste de un gráfico estructurado en el espacio de configuración. Es decir, la búsqueda de los hiperparámetros más adecuados para cada red en base a los resultados de la métricas de evaluación. Aunque, tal y como se indica en [10, 11], el reto de hallar los hiperparámetros óptimos es un impedimento trascendental para el progreso científico en el marco de redes

neuronales. El estudio de la optimización de los mismos debería formar una capa más de análisis en la optimización de resultados en el campo de las redes convolucionales [5].

La optimización de hiperparámetros es uno de los pasos más relevantes en el desarrollo de la arquitectura de las redes convolucionales. Para ello, hay múltiples maneras que permiten obtener la configuración óptima en base a una métrica dada, como *GridSearch*, *RandomSearch*, *Tree-structured Parzen Estimator Approach (TPE)*, *early-stopping* o propuestas alternativas, como la optimización bayesiana sobre *GPs* [47]. No obstante, en este trabajo se realizará una búsqueda no automática (manual search [20]).

3.2. Optimizers

Los algoritmos de optimización u optimizadores (*optimizers*) ejercen un papel indispensable en la minimización de la función de coste y tienen como objetivo encontrar los parámetros óptimos (sesgos y pesos) durante el proceso de entrenamiento de la red [76]. A continuación, se describirán ciertos optimizadores, dando nociones teóricas sobre los mismos, que permitirán extraer ciertas conclusiones en el apartado de resultados. Para este proyecto se ha decidido estudiar SGD, RMSprop y Adam.

Para comprender de manera sencilla cada uno de los optimizadores y cuál es la idea básica, de nuevo, se empleará el ejemplo didáctico. Suponiendo que se parte de un problema de aprendizaje supervisado donde se poseen los valores de la dosis (x) y se pretende predecir la eficacia (y). La función $l(y,\hat{y})$ cuantifica la diferencia entre el resultado predicho $\hat{y}$ e y, resultado observado. Entonces, el objetivo consiste en hallar una función $f_w(x)$ parametrizada por el vector de pesos[6], que minimice la función de coste $l(y,\hat{y})$.

En definitiva, entrenar la red neuronal no es más que realizar un problema de optimización no convexo sobre la función de coste [37]:

$$min_{w\in\mathbb{R}} Q(z,w) = min_{w\in\mathbb{R}} l(f_w(x),y) \tag{3.3}$$

donde z es el par (x,y). Por tanto, una vez comprendida la idea de optimización, emplear distintos optimizadores va a significar utilizar el método de descenso por gradiente, donde cada optimizador añadirá una serie de modificaciones a dicha fórmula. La expresión general del descenso por gradiente cuyo objetivo es optimizar los pesos de la red neuronal se muestra

[6]Para no ser insistente, de ahora en adelante, cuando se hable de pesos, se referirá al conjunto de pesos y sesgos para aumentar la ligereza del contenido

a continuación, embebida en la ecuación de actualización de parámetros [8]:

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t) \tag{3.4}$$

SGD

Stochastic Gradient Descent es un algoritmo cuya formulación se basa en la aproximación estocástica analizada en [62]. En cuanto a su formulación, se trata de una simplificación del descenso por gradiente donde en cada iteración se estima el gradiente de un elemento z_t elegido de manera arbitraria:

$$w_{t+1} = w_t - \gamma_t \nabla_w Q(z_t, w_t) \tag{3.5}$$

Cada actualización intenta mejorar la predicción para ajustarse a una única muestra, por lo que cada iteración será realmente rápida. Mediante SGD no se emplea todo el conjunto de datos y, consecuentemente, cada paso no tiene por qué ser en la dirección correcta. Por ende, serán necesarios más pasos para converger al mínimo. Es por este motivo que hay ciertos estudios que proporcionan mejoras o consejos para perfeccionar su eficiencia [40, 88]. La velocidad de convergencia de SGD está limitada por la ecuación 3.5. Cuando las ganancias disminuyen muy lentamente, la varianza de la estimación del parámetro w_t, disminuye con la misma lentitud. Sin embargo, si las ganancias disminuyen demasiado rápido, la expectativa de la estimación del parámetro w_t, tarda mucho tiempo en acercarse al valor óptimo [7]. Por tanto, respecto al descenso por gradiente estándar tenemos un cierto *trade-off* porque SGD permite realizar un cálculo rápido por iteración y convergencia lenta. Es decir, la aproximación estocástica introduce una aleatoriedad que induce una variabilidad que causa una convergencia más lenta [34].

Ideas principales. SGD

El algoritmo SGD simplifica el descenso por gradiente. La actualización de pesos se realiza para cada muestra, donde cada iteración es rápida. El inconveniente principal de este método es que la componente aleatoria introduce una cierta varianza que ocasiona fluctuaciones fuertes en la optimización de la función de coste, incrementando el tiempo de convergencia.

RMSprop

Root Mean Square Propagation es otro algoritmo de optimización del descenso por gradiente, propuesto por Geoffrey Hinton [74]. RMSprop sugiere normalizar los gradientes mediante una media móvil exponencial de la magnitud del gradiente para cada parámetro.

Con el fin de entender mejor la media móvil exponencial (*exponential mean average, EMA*) se tomará como ejemplo un modelo en el que se pretende modelizar la tendencia del progreso de cotización de las acciones [57], es decir, una gráfica del valor de la acción frente al tiempo. Por consiguiente, el valor de EMA (v_t) para un día dado t es similar a obtener la media del valor de las acciones en $\frac{1}{1-\beta}$ días, donde β es una valor entre 0 y 1 que representa el ratio ponderado de decaimiento, siendo un hiperparámetro a elegir. En consecuencia, la expresión de EMA es [46]:

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t \tag{3.6}$$

donde θ_t es el valor de la acción en un día dado t y v_t, es el promedio. Entonces, si β es elevado, se adapta mal a cambios bruscos, mientras que, si es bajo, se deja influir por valores atípicos.

Una vez asimilado el concepto de la EMA, que nos proporciona la tendencia de una magnitud dada, se procede a definir el algoritmo de RMSprop:

$$w_t = w_{t-1} - \gamma \frac{\nabla_w Q(z_t, w_t)}{\sqrt{v_t} + \varepsilon} \tag{3.7}$$

donde dicha normalización dada por la media móvil exponencial (el valor ε es muy pequeño, para evitar ceros en el denominador) es la causante de la mejora respecto a SGD en el proceso de optimización. En [12], se muestra una justificación teórica de como RMSprop brinda un cierto equilibrio o regula dicha variación causada en SGD, motivo de la aparición de problemas como la dificultad de convergencia, incrementando el tiempo de ejecución. No obstante, RMSprop también plantea ciertos aspectos adversos de convergencias, es decir, en estudios como [65] se muestra que la convergencia de RMSprop está realmente influenciada por la decisión de los valores que pueden tomar los hiperparámetros.

Ideas principales. RMSprop
El algoritmo RMSprop mejora los problemas de tiempo de cómputo respecto a SGD al introducir la normalización de los gradientes mediante la media móvil exponencial (regula dichas variaciones en la búsqueda del mínimo ocasionadas en SGD). Sin embargo, existen estudios que muestran ciertos problemas de divergencia.

Adam

Adaptative moment es un algoritmo de optimización de primer orden que surge como sustituto del clásico SGD [42]. El método es sencillo de aplicar, eficiente desde el punto de vista computacional, tiene pocos requisitos de memoria, es invariable al reescalado diagonal de los gradientes y se adapta bien a los problemas que son grandes en términos de datos y/o

parámetros. El método también es adecuado para objetivos no estacionarios y problemas con gradientes muy ruidosos y/o dispersos. Además, los hiperparámetros tienen interpretaciones intuitivas y suelen requerir poco ajuste [38]. Adam se puede entender como una mezcla de SGD con momento y RMSprop [43]. Así pues, las ecuaciones que definen Adam son [33]:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_t Q(z_t, w_t) \tag{3.8}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla_t Q(z_t, w_t)^2 \tag{3.9}$$

$$w_t = w_{t-1} - \gamma \frac{\frac{m_t}{1-\beta_1^t}}{\sqrt{\frac{v_t}{1-\beta_2^t}} + \varepsilon} \tag{3.10}$$

donde m_t es la parte relacionada con el primer momento del gradiente (media de decaimiento exponencial sobre los gradientes) y v_t, con RMSprop (segundo momento del gradiente, media de decaimiento exponencial sobre los gradientes cuadrados [64]). Los hiperparámetros β_1 y β_2 representan los coeficientes de atenuación de primer y segundo orden del momento respectivamente. Generalmente, se suele tomar $\beta_1 = 0{,}9$ y $\beta_2 = 0{,}999$ y $\varepsilon = 10^{-8}$.

Ideas principales. Adam
El algoritmo Adam es una combinación de SGD y RMSprop (se podría decir que RMSprop se obtiene a partir de Adam con $\beta_1 = 0$). Así pues, combina la agilidad computacional de SGD con la convergencia de RMSprop.

3.3. Métricas de validación. *Image quality assessment*

La evaluación de la calidad de la imagen (*image quality assessment*, IQA) desempeña un papel importante en el procesamiento de imágenes. Una métrica de calidad tiene tres tipos de aplicaciones [81]:

- **Supervisión de la calidad de las imágenes**. Para realizar un control de calidad de las imágenes.
- **Evaluación de los algoritmos de procesamiento de imágenes**. Es decir, utilizar la métrica de calidad como distintivo para elegir un algoritmo u otro.
- **Optimización de algoritmos**. Pueden emplearse como método de procesamiento para optimizar el ajuste de parámetros.

Las métricas más convencionales son *Peak Signal-to-Noise ratio* (PSNR)[7] y *Mean Square error* (MSE) y vienen dadas por las siguientes expresiones[8] [27]:

$$PSNR(\alpha,\beta) = 10\log_{10}(\frac{Max_{\alpha}^2}{MSE(\alpha,\beta)}) \tag{3.11}$$

donde MSE es,

$$MSE = \frac{1}{CHW}\sum_{i=1}^{C}\sum_{j=1}^{H}\sum_{k=1}^{W}(\alpha_{ijk}-\beta_{ijk})^2 \tag{3.12}$$

Por tanto, ambas métricas operan directamente sobre el valor de intensidad de la imagen [87], sin tener en cuenta las autocorrelaciones[9].

Como consecuencia de la carencia de correlación entre las métricas tradicionales, se han realizado múltiples investigaciones en los desarrollos de métricas basadas en el sistema visual humano (*human visual system*, HVS), es decir, considerando la luminancia, el contraste o el contenido de frecuencia. En base a ello, surgen métricas como SSIM que se basa principalmente en la extracción de información estructural. Su expresión es la siguiente [27]:

$$SSIM(\alpha,\beta) = l(\alpha,\beta)c(\alpha,\beta)s(\alpha,\beta) \tag{3.13}$$

donde,

$$l(\alpha,\beta) = \frac{2\mu_\alpha\mu_\beta + c_1}{\mu_\alpha^2 + \mu_\beta^2 + c_1} \tag{3.14}$$

$$c(\alpha,\beta) = \frac{2\sigma_\alpha\sigma_\beta + c_2}{\sigma_\alpha^2 + \sigma_\beta^2 + c_2} \tag{3.15}$$

$$s(\alpha,\beta) = \frac{2\sigma_{\alpha\beta} + c_3}{\sigma_\alpha\sigma_\beta + c_3} \tag{3.16}$$

donde el primer término es la función comparativa de iluminación que mide la cercanía entre las dos medias de iluminación, μ_α y μ_β. El segundo término es la función de comparación de contraste, donde el contraste es cuantificado en este caso vía σ_α y σ_β. Y el tercer término contiene la covarianza $\sigma_{\alpha\beta}$. Las constantes son empleadas para evitar ceros en el denominador.

La extensión multiescala de SSIM es MS-SSIM [82]. En estudios como [59], se muestra que SSIM y MS-SSIM, obtienen un redimiento estadístico superior (autocorrelación),

[7] Se suele medir en dB.

[8] Dada una imagen α y otra imágen β, donde ambas tienen tamaño (C,H,W) donde C son los canales, H la altura y W, el ancho.

[9] El valor de PSNR tiende a infinito a medida que MSE tiende a cero. Entonces, un menor valor de MSE implicará un mayor valor de PSNR, es decir, una mayor calidad en base a dicha métrica.

respecto a otras métricas de IQA. Sin embargo, SSIM y MS-SSIM comparten una deficiencia común: en una localidad del mapa de características o del mapa de características distorsionado todas las posiciones son consideradas con la misma relevancia [87].

En definitiva, hay múltiples índices [2, 80] para caracterizar la calidad de las imágenes durante el procesado de las mismas, donde MSE, PSNR y SSIM, son las métricas más ampliamente utilizadas debido a su simplicidad [82]. En el caqpítulo 6, también se muestran otras métricas como GMSD, MS-GMSD, HARPSI y VSI[10].

[10]Tan solo se han mostrado por completitud de la información, ya que el análisis de resultados se realizará en base a MSE, PSNR y SSIM

Capítulo 4

Primeros pasos para la creación del modelo

El objetivo del proyecto es magnificar los cambios sutiles o prácticamente imperceptibles para el ojo humano entre cada par de imágenes consecutivas. Pero, ¿de qué manera podemos conseguir dicha finalidad mediante nuestros conocimientos previos de *Machine Learning*?

En primer lugar, la arquitectura escogida para llevar a cabo la tarea de distinguir dichos cambios poco apreciables será una red neuronal convolucional. Así pues, si consideramos un conjunto de imágenes inicial, tendremos que dividir todas las muestras en tres particiones: entrenamiento (*training*), validación (*validation* o *development*) y ejecución (*test*), en unas proporciones de, por ejemplo, 75 % para *training*, 20 % para validación y 5 % para test .

Durante la fase de entrenamiento la red aprenderá los parámetros óptimos asociados, mientras que, en la fase de validación, se comprobará su eficacia en base a criterios, como la métrica de validación, para ver su capacidad de generalización. La red genera la salida y se comprueba la efectividad, comparando la similitud del resultado con las muestras de salida real mediante un cierto criterio (métrica de validación; nos permite discernir como de bueno es el modelo entrenado). Por último, la red escogida como adecuada, se ejecuta con las muestras de test (producción).

Así pues, el concepto fundamental es que tenemos un conjunto de imágenes de entrada y necesitamos un *output* que contenga dichas imágenes de entrada pero con un ligero cambio: una ampliación a nivel local de alguna de sus regiones. De hecho, la explicación que se da a continuación no es más que la dinámica para lograr dicha generación de *output* o *groundtruth*. De esta manera, la red neuronal aprenderá a percibir los cambios imperceptibles, aumentando una cierta área de la misma.

La obtención de pares de vídeos magnificados con movimiento real es un reto. Por lo tanto, utilizamos datos sintéticos que pueden generarse en gran cantidad. Para ello, crearemos la imagen magnificada, facilitando el proceso de entrenamiento y teniendo un mayor control. Como el proceso de magnificación se realiza al vuelo mediante métodos aleatorios para la extracción de las regiones magnificadas se logra un mayor reconocimiento de los movimientos locales. Para ello, los pasos seguidos en la metodología propuesta son los siguientes:

- **Ventana aleatoria**. En primer lugar, se toma una imagen original cuya base y altura son (b,h) respectivamente[1]. Así pues, la idea consiste en extraer una ventana o *patch* de la imagen original. Para ello, en el archivo en el cual se guardan los parámetros de configuración se detallarán las dimensiones del *patch*, (b',h'). Por tanto, el objetivo es extraer una ventana con dichas dimensiones de cualquier región de la imagen original. El centro de la ventana en referencia a la imagen original, se hallará generando un número aleatorio en el intervalo $((0,b-b'),(0,h-h'))$ (NumPy). Por tanto, se extrae una ventana de tamaño fijo y localización aleatoria. Si la ventana se encontrara fuera de los dominios de la imagen original, no se tendrían en cuenta dichos píxeles.

- **Resize**. A continuación, se realizará una magnificación o ampliación de la ventana aleatoria extraída mediante interpolación intercúbica[2] (OpenCV). La amplicación (Fig. 4.1) de la imagen de entrada será en un factor de escala[3] elegido de manera aleatoria. Cuando se trate de la magnificación de la imagen original que recibe la red, el factor de escala será menor que el de la ventana de *groundtruth*, que produzca la red. Es decir, el factor de amplificación es regulable, de tal manera que tendrá un valor para el entrenamiento y otro para el objetivo.

- **Máscara gausiana**. El filtro gaussiano permite realizar un procesado en el dominio espacial, suavizando uniformemente y eliminando ruido de la imagen. Como resultado, genera una imagen cuyo centro es totalmente visible y que se va degradando conforme se aproxima a los bordes (Fig. 4.2). Más adelante, se mostrará el interés de aplicar una máscara gaussiana. Por tanto, aplicar una máscara gaussiana no es más que calcular cuánto vale la función gaussiana de los píxels (recordemos que las imágenes no son

[1] En realidad, también habría que tener en cuenta la tercera dimensión que se corresponde al número de canales. En este caso, habrían 3 canales ya que son imágenes RGB. No obstante, para el ejemplo, se consideran solo base y altura por simplificar.

[2] Es una técnica polinomial para enfocar y ampliar imágenes digitales, es más, consiste en estimar datos desconocidos a partir de los píxeles circundantes. En OpenCV, se utilizan los píxeles vecinos de la cuadrícula 4x4.

[3] Para llevar a cabo buenas prácticas, se ha diseñado un archivo json, externo al programa, para poder modificarlos, sin ensuciar el código.

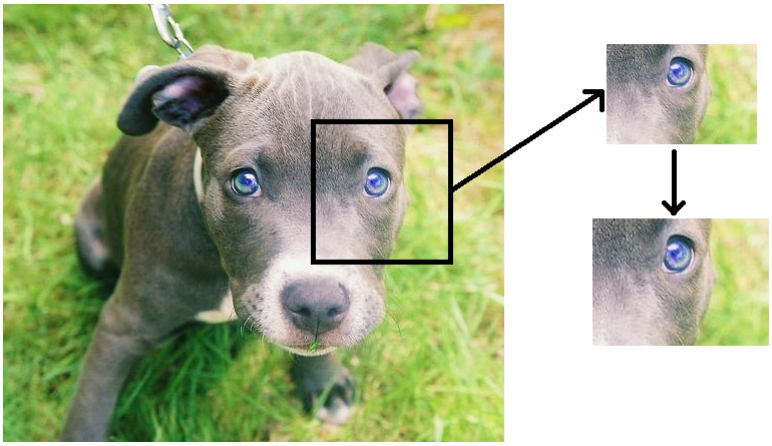


Figura 4.1 Pasos 1 y 2. Extracción de ventana aleatoria y magnificación de la misma.

más que matrices con valores. Si la imagen está en color, tendremos tres matrices, una para cada canal RGB). Por tanto, el concepto fundamental es que queremos lograr que la gaussiana valga 1 en el centro del *patch* magnificado, y vaya dismuyendo su valor cuando alcance el borde[4] (Fig. 4.2). Dicho matemáticamente:

$$k = \frac{\log(g)}{2} \tag{4.1}$$

$$\alpha = e^{k\left(\frac{(x-tx_2/2)^2}{(tx_2/2)^2} + \frac{(y-ty_2/2)^2}{(ty_2/2)^2}\right)} \tag{4.2}$$

donde g es el valor que se desea que tenga la gausiana en los bordes. Así pues, si se considera g = 0.1 para la imagen ampliada, se obtiene (Fig. 4.2):

Figura 4.2 Ventana ampliada con máscara gausiana.

donde, como se puede comprobar, cuanto más nos alejamos del centro, menor es el color de la imagen.

[4]El valor en el borde de la gaussiana también se encuentra en el archivo json.

- **Alpha blending**. Se trata del último paso en el proceso para generar una imagen con una cierta ventana ampliada de una región, es decir, un cambio aparentemente imperceptible. Para ello, lo que se hará será realizar una copia de la imagen original y, fusionar dos imágenes: la copia de la imagen original junto con nuestro patch ampliado centrado en dicha copia. ¿Cómo? Mediante la siguiente expresión matemática:

$$img_2 = (1-\alpha)img_1 + \alpha patch_{amp} \tag{4.3}$$

Esto no se realizará de manera arbitaria, sino centrada. Es decir, si se partía de una ventana aleatoria img[40:50, 40:50], se extrae una ventana 10x10. Por tanto, si se está ampliando a un 30x30, la imagen tendrá que estar centrada en (35:55, 35:55).

El coeficiente α no va a ser un coeficiente fijo, sino que va a depender de los píxels en cuestión. De hecho, dicho coeficiente será el α calculado previamente (máscara gausiana).

Dicho proceso de fusión de imágenes con el coeficiente variable dado por la gausiana tiene como objetivo que, al situarse en el centro del *patch* ampliado $(\frac{tx_2}{2}, \frac{ty_2}{2})$, $\alpha = 1$, con lo que, la nueva imagen (img_2) será igual al *patch*. Sin embargo, al situarse en el borde del *patch*, $\alpha = g$, donde g es un valor pequeño, de tal forma que quien contribuye mayoritariamente en el borde es la propia img_2. Por tanto, el hecho de que el coeficiente sea variable y se corresponda al valor gaussiano, permite realizar un pegado suave que logra preservar los bordes. De esa manera, se logra maximizar cambios sutiles mediante una extracción aleatoria de una ventana, seguida de un *resize* y culminada con α-blending, cuyo coeficiente es variable y viene dado por la gaussiana en el punto. Como resultado de este último paso se obtiene la Fig. 4.3.

Figura 4.3 Resultado final. *Groundtruth* de la imagen de estudio.

Es decir, la metodología llevada a cabo en el presente apartado no es más que un área de estudio de visión artificial, llamada *composition and matting*. Donde el proceso de extracción de un objeto de una imagen original es lo que se conoce como *matting* [67], mientras que el proceso de inserción de una imagen en otra, sin artefactos visibles, se denomina *composition* [60].

Así pues, el método de *alpha blending* es el proceso de superponer una imagen en primer plano (*foreground image*) con transparencia, con una imagen en segundo plano o de fondo (*background image*). Sin embargo, cuando se componen ambas imágenes, la transparencia de la composición[5] viene dada por la expresión mencionada previamente (ec. 4.3). Es decir, si se parte de la imagen extraída, además de sus tres canales RGB, es como si tuviera un cuarto canal que no es más que la propia imagen pero escalada por un factor α (o canal α), que describe la cantidad o fracción relativa de opacidad de cada píxel. En otras palabras, es el valor de α multiplicado por la imagen extraída, *alpha-matted image*). En realidad, aunque se hable de que α es un factor, puede entenderse como un filtro o una imagen que pondera la ventana extraída. Para verlo más claro, véase la ecuación 4.3, de manera gráfica, en la figura 4.4 :

Figura 4.4 Representación esquemática de la ecuación de composición o α-blending: $C = (1-\alpha)B + \alpha F$ [71].

Entonces, si se tiene la ecuación $C = (1-\alpha)B + \alpha F$, el operador atenúa la influencia de la imagen de fondo en un factor $(1-\alpha)$, dándole color y opacidad a la figura F (Fig. 4.4). Por esa razón, más que factor, también se denomina, *alpha mask* o *alpha matte* [48]. Por tanto, la composición centrada entre ambas imágenes, en este caso la ventana aleatoria magnificada y una copia de imagen original, es una combinación píxel a píxel.

La cuestión es que el valor de α no va consistir en un filtro con un valor constante en todos sus elementos sino que va ser una función y, concretamente, va a ser una función gaussiana como la que se muestra a continuación [84]:

$$G(x,y;\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.4}$$

[5]Transparencia es el antónimo de opacidad.

Por tanto, la ecuación de α-*blending*, se escribiría estrictamente como $C(x,y,k) = (1 - \alpha(x,y,k))B(x,y,k) + F(x,y,k)\alpha(x,y,k)$, donde x e y, son altura y ancho respectivamente, y k el canal (hay 3 canales para imágenes RGB). En definitiva, se ha optado por emplear una función, filtro, operador o distribución gaussiana porque permite realizar un suavizado controlado por el parámetro σ (previamente ha sido descrito, solo que embebido en el número $k = \frac{1}{2\sigma^2}$). Una de las principales ventajas de emplear el operador gaussiano, es el suavizado en los bordes controlado por dicho parámetro, que evita problemas como los artefactos en los bordes (*edge artifacts*). Por tanto, dicho operador aporta en el proceso de composición tanto la preservación de los bordes como el pegado suavizado [26, 19].

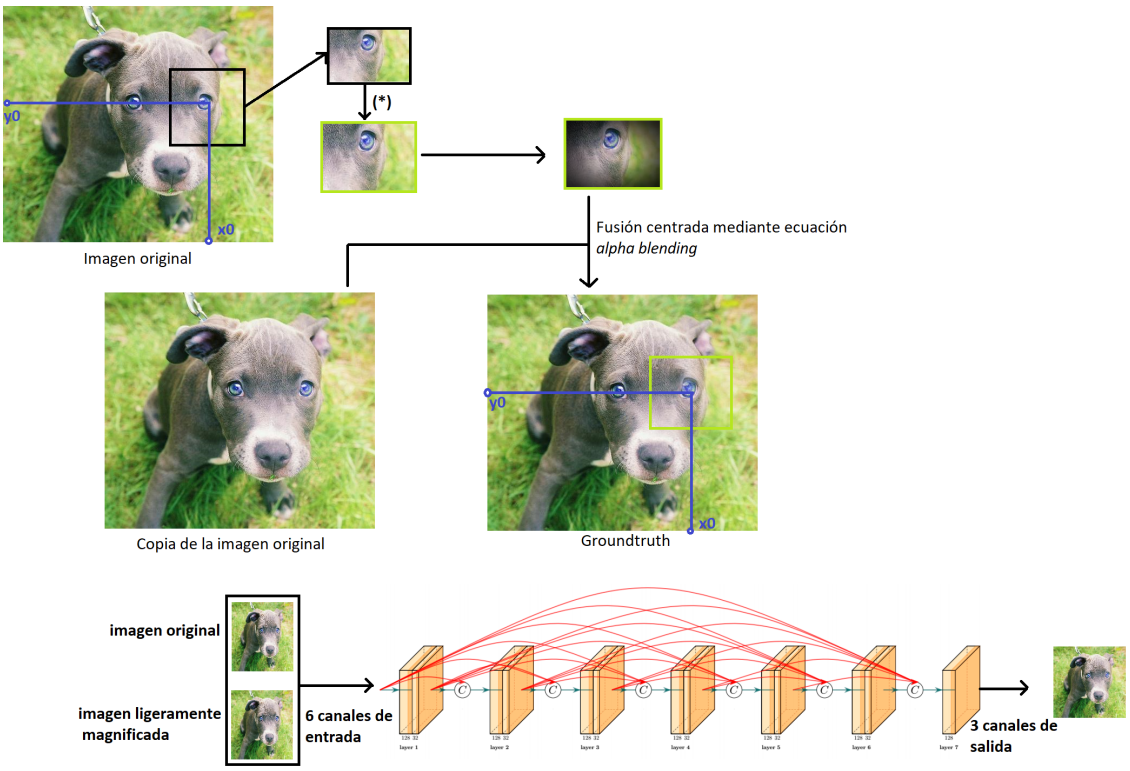


Figura 4.5 Representación esquemática de la metodología para la generación del conjunto de imágenes amplificadas.

Tanto la imagen de entrada ligeramente amplificada, como la imagen amplicada de salida en un factor de ampliado superior, están centradas en las mismas coordenadas.

Capítulo 5

Establecimiento del modelo

En el presente apartado se describen las distintas pruebas llevadas a cabo para la búsqueda del modelo propuesto del proyecto. Para ello, se probarán tres arquitecturas distintas basadas en capas convolucionales, capas convolucionales con bloques residuales y capas convolucionales formando un bloque denso. Cada arquitectura se entrenará con tres algoritmos de optimización distintos: SGD, RMSprop y Adam. Tras escoger la combinación de arquitectura y optimizador adecuada en base a criterios como la evolución de la métrica de validación, se modificarán hiperparámetros como la tasa de aprendizaje, valores de regularización L_2 y número de capas. En este apartado se dan detalles sobre cada prueba. Por último, se muestran los recursos computacionales y, finalmente, los resultados obtenidos acompañados de una discusión en base a las nociones teóricas de capítulos anteriores.

Para las tres arquitecturas planteadas la red neuronal recibe dos imágenes, la imagen original y la imagen original ligeramente magnificada. Sin embargo, la red genera como salida una única imagen magnificada en un mayor factor de escala. Por tanto, recibe 6 canales de entrada, puesto que son imágenes en color (RGB), y produce 3 canales de salida.

En cuanto a los parámetros considerados que tienen todas las pruebas siguientes, cabe destacar que entre las capas convolucionales siempre se considera como función de activación no lineal, *Rectifier Linear Units* (ReLU) [55], para un total de 7 capas convolucionales. Además, las capas convolucionales internas reciben 32 canales de entrada y devuelven 32 canales de salida (salvo en el caso de bloque denso, donde se concatenan). También se utilizan *kernels* de dimensión (3,3), con valores de *stride* y *padding*, iguales a 1. No se han añadido capas intermedias tipo *maxpooling*. En cuanto al tamaño del *patch* (capítulo 3), se ha escogido una ventana de (200,200,3)[1]. Para el factor de escala, se ha decidido tomar un valor aleatorio en el intervalo 10-15 píxeles para la imagen ligeramente magnificada (entrada), y 20-25 píxeles, para el *groundtruth* (salida). Respecto a los valores de la gaussiana en el borde,

[1] (base, altura, canales).

se ha seleccionado $k = 0{,}1$ para la imagen ligeramente magnificada y $k = 0{,}15$ para la imagen de salida. Si bien el lector puede pensar que los parámetros de magnificación escogidos son realmente elevados para la aplicación considerada (revelar pequeños cambios), es necesario destacar que se ha realizado así para comprobar notoriamente los resultados durante las pruebas. Todos los ensayos iniciales (arquitecturas y optimizadores) se han implementado en base a los mismos valores de hiperparámetros (número de capas, tasa de aprendizaje, etc) para realizar una compración lógica [22].

Asimismo, para monitorizar el resultado de la red a lo largo de las épocas, se comprueba el MSE entre los valores de los píxeles del *groundtruth* magnificado con el producido por la CNN (PyTorch).

En el siguiente apartado de resultados se muestran las gráficas obtenidas de la función de coste (MSE) a lo largo de 30 épocas para una arquitectura de capas convolucionales en entrenamiento[2] (*training*) y validación (*validation* o *developpment*). Para ello, se han empleado 30 imágenes en entrenamiento y validación y 10 imágenes para test[3]extraídas de *Stanford Dogs Dataset*. A primera vista puede parecer que tanto el número de imágenes como el número de épocas no es elevado. No obstante, este estudio inicial servirá como paso previo para determinar la arquitectura, puesto que nos permitirá realizar un análisis riguroso en base a los resultados obtenidos y los conceptos explicados en el fundamento teórico previamente, además del algoritmo de optimización.

En el apartado de resultados, se darán más detalles específicos sobre las arquitecturas consideradas. A continuación, se muestra una representación didáctica de la arquitectura de capas convolucionales (Fig. 5.1). En la figura 5.2, se trata de la arquitectura de capas convolucionales de bloques residuales, donde se han adaptado tres bloques residuales cada par de capas convolucionales. La última arquitectura analizada, capas convolucionales formando un bloque denso, apila o concatena capas tal y como se muestra en la figura 5.3.

[2]En el entrenamiento la red aprende los parámetros óptimos mientras que en la fase de validación tan solo lo aplica a un conjunto de imágenes no vistos previamente por la red, evaluando así mediante las métricas la capacidad de generalización, evitando problemas de *overfitting*

[3]Ejecución de la red cuyos resultados en validación son más óptimos sobre un nuevo conjunto de datos

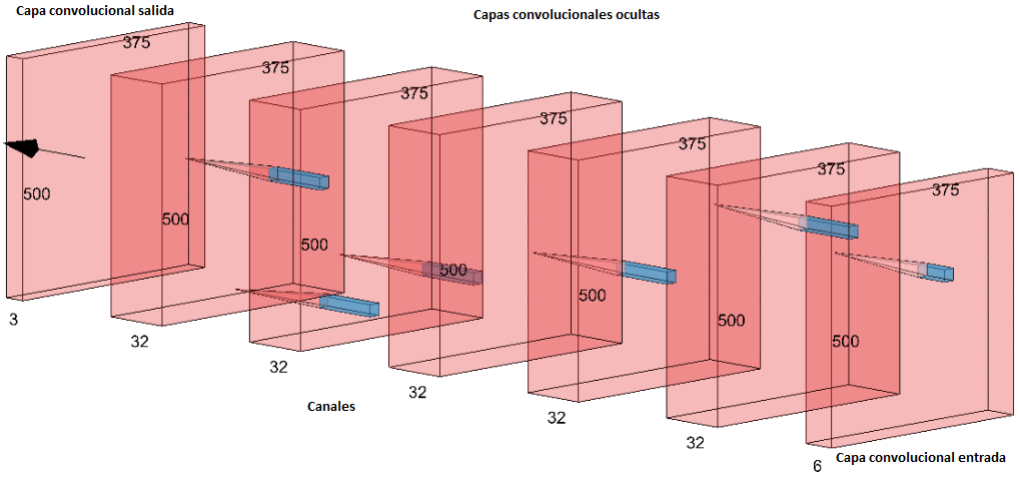


Figura 5.1 Representación gráfica de la arquitectura I. Capas convolucionales [45].

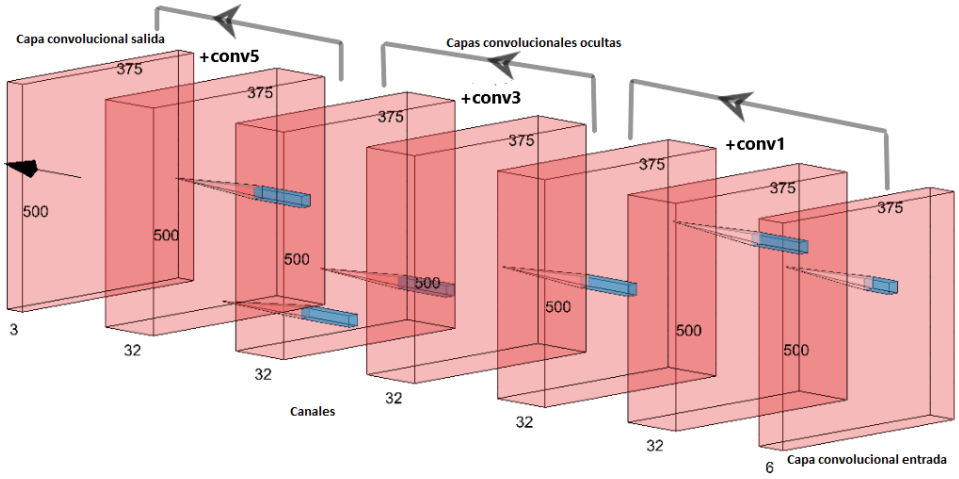


Figura 5.2 Representación gráfica de la arquitectura II. Redes convolucionales con bloques residuales [45].

donde se observan las dimensiones (número de canales, ancho y alto), además del concepto de convolución de la imagen (o tensor de entrada) y el filtro.

Por tanto, el objetivo del apartado es comprobar cuál es la arquitectura cuyos resultados en validación son más adecuados. Para ello, se emplearán las mismas condiciones, posibilitando una comparación lógica. Además, para cada arquitectura, se probarán los *optimizers* descritos previamente: SGD, RMSprop y Adam. De esta manera, se perfilará la arquitectura en conjunto con el optimizador más óptimos y aquella cuyo resultado en validación sea más satisfactorio será considerada para la optimización de los hiperparámetros realizado *a posteriori*.

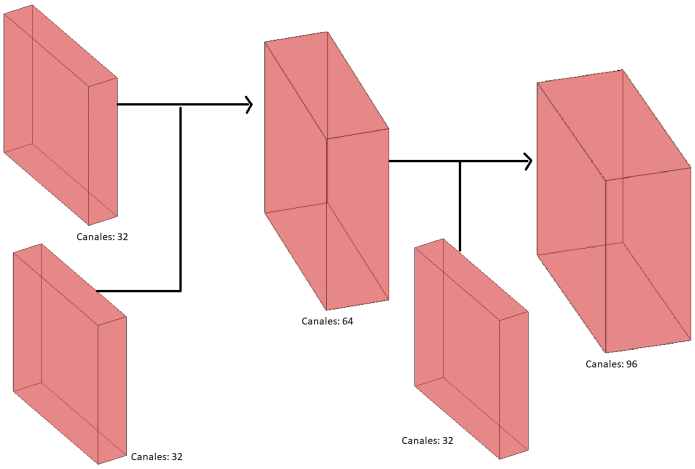


Figura 5.3 Representación gráfica de la arquitectura III. Redes convolucionales con bloque denso [45].

5.1. Recursos computacionales

Para escoger el modelo inicialmente se probaron distintas pruebas para la determinación del modelo en local. No obstante, surgieron problemas como la lentitud de ejecución durante el proceso y el colapso de la memoria. De hecho, en la figura 5.4, se muestra en la parte superior, el mensaje que lanzó el ordenador por problemas de memoria, indicando que se comprara una RAM nueva. Fue entonces cuando se pasó a emplear una GPU perteneciente al grupo de Computación Avanzada y al Grupo de *e-Science*, pertenecientes al Instituto de Física de Cantabria (IFCA-CSIC-UC), que permitió acelerar los procesos de entrenamiento.

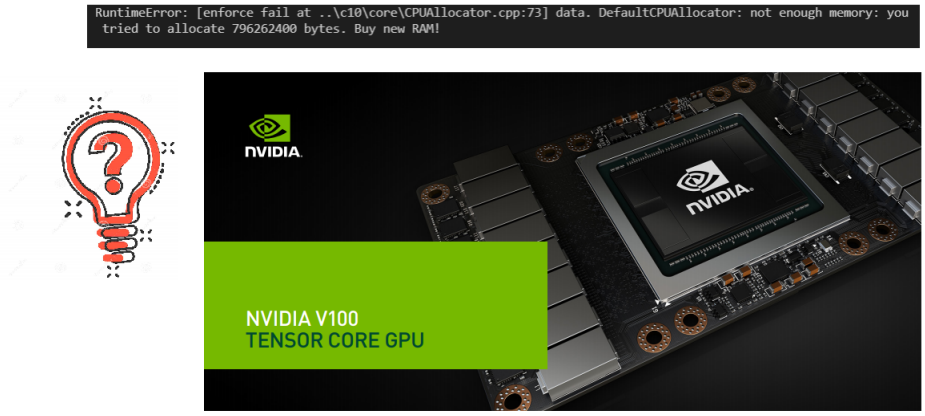


Figura 5.4 Representación de los problemas de memoria en local (superior), solventados con la GPU brindada por el IFCA (inferior).

Por tanto, todas las pruebas realizadas para determinar la estructura de la propuesta de modelo se han realizado en dicha GPU. Se trata de la GPU NVIDIA V100 PCIe, de arquitectura NVIDIA Volta. En cuanto a sus características, tiene 640 núcleos (NVIDIA tensor cores) y 5120 (NVIDIA Cuda Cores). Además, tiene una memoria de 32 GB y 16GB HBM2, con un *memory bandwidth* de 900 GB/segundo. Así como una potencia máxima de consumo de 250W y distintas APIs como CUDA, DirectCompute, OpenCL y OpenACC.

5.2. Arquitectura & *Optimizers*

5.2.1. Capas convolucionales

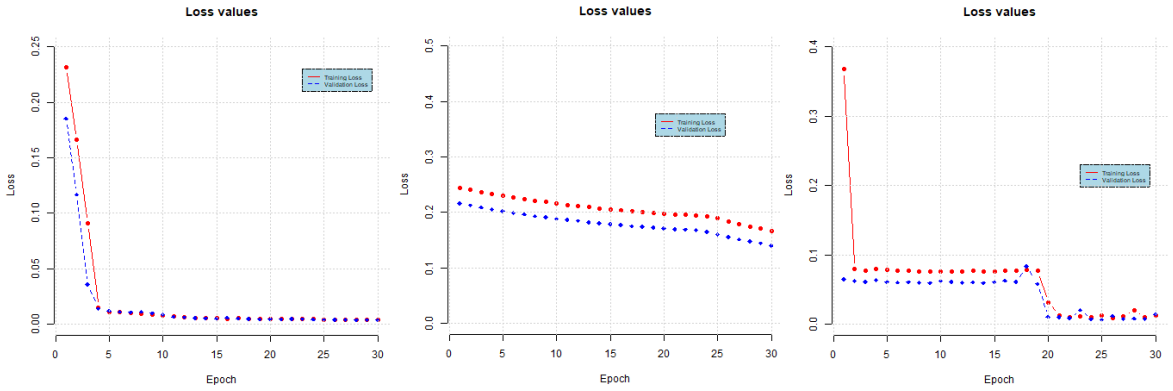


Figura 5.5 MSE en función del número de épocas. La figura de la izquierda se corresponde a Adam, la central a SGD y la de la derecha a RMSprop para arquitectura tradicional de capas convolucionales.

Optimizer	**Función de coste**	**Valor óptimo**
Adam	*Training*	0.0036
	Validation	0.0036
SGD	*Training*	0.1668
	Validation	0.1402
RMSprop	*Training*	0.0095
	Validation	0.0059

Cuadro 5.1 Valores óptimos de la función de coste en las fases de entrenamiento y validación para la arquitectura de capas convolucionales.

ANÁLISIS

Se observa como el *optimizer* Adam alcanza el menor valor de MSE en validación siendo, por tanto, el método cuya capacidad de optimización es superior respecto a los otros optimizadores. En las misma condiciones, SGD consigue un valor de MSE en validación superior en 2 órdenes de magnitud respecto a RMSprop y Adam. Así pues, se observa a nivel experimental el inconveniente del tiempo de convergencia de SGD, especificado en el marco teórico (donde dicha lentitud computacional, puede originarse debido al desvanecimiento del gradiente, degradando y frenando la convergencia y aumentando el tiempo de cómputo).

En cuanto a la capacidad de convergencia, como se puede comprobar, Adam converge aproximadamente en la época 5, mientras que RMSprop necesita 20 épocas y SGD, directamente, no alcanza esa banda de valores de convergencia para MSE. Consecuentemente, Adam logra una mayor convergencia en menor tiempo, que es justo lo esperado a nivel teórico, puesto que combina la rapidez de SGD y la convergencia de RMSprop. Por último, no se observa sobreentrenamiento, probablemente por el reducido número de iteraciones considerado.

5.2.2. Capas convolucionales (bloques residuales)

A continuación, mostramos los resultados de la arquitectura II, basados en *ResNet*. La única diferencia respecto a la arquitectura I reside en la incorporación de 3 bloques residuales, intercalados cada dos capas. Para una mayor visualización, la figura 5.2 puede ser de gran ayuda. Además, también se muestran los mejores valores de la *loss function* para 30 épocas.

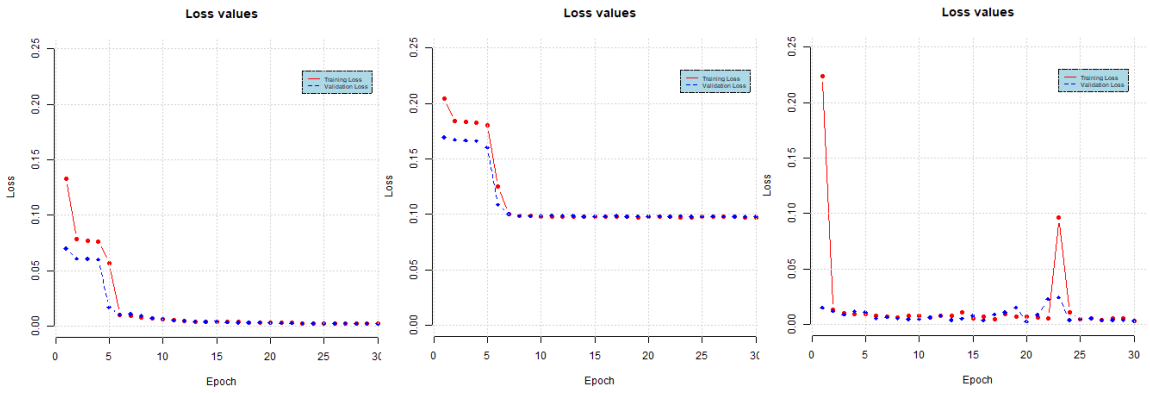


Figura 5.6 MSE en función del número de épocas. La figura de la izquierda se corresponde a Adam, la central a SGD y la de la derecha, a RMSprop para arquitectura de bloques residuales.

Optimizer	Función de coste	Valor óptimo
Adam	*Training*	0.0023
	Validation	0.0025
SGD	*Training*	0.0092
	Validation	0.0107
RMSprop	*Training*	0.0041
	Validation	0.0024

Cuadro 5.2 Valores óptimos de la función de coste en las fases de entrenamiento y validación para la arquitectura de capas convolucionales con bloques residuales.

ANÁLISIS

En este apartado se ha realizado una prueba análoga al apartado anterior, pero modificando la arquitectura mediante la incorporación de bloques residuales. En cuanto al valor óptimo de MSE, se observa que de nuevo el optimizador Adam logra un mejor resultado para una arquitectura tipo *ResNet*. Para SGD, a partir de la quinta época, se puede comprobar en la representación gráfica como la función de coste se degrada y se paraliza en torno a 0.1. Es decir, para bloques residuales el método de optimización SGD no actualiza en absoluto los valores del gradiente, imposibilitando la actualización de los pesos (es probable que se deba al *vanishing gradient*). En esta ocasión, RMSprop alcanza el rango de valores de 10^{-2} en cuanto a la función de coste en un menor número de épocas, sin embargo, se observan ciertas oscilaciones pronunciadas en el rango de épocas 15-25. Por tanto, observamos que RMSprop brinda unos resultados buenos pero con ciertas tendencias a la divergencia o, mejor dicho, muestra cierta inestabilidad, tal y como se comentó en el apartado teórico. En cuanto a la inversión en tiempo computacional durante el proceso de entrenamiento, sin duda, SGD obtiene la matrícula. En definitiva, Adam muestra ser algo más lento que RMSprop, pero más robusto o estable (menor variabilidad), lo que se traduce en menor MSE, sin riesgo de *overfitting* y mayor convergencia.

5.2.3. Capas convolucionales (bloque denso)

Por último, se ha implementado un bloque denso con el objetivo de comparar los resultados de validación con las anteriores configuraciones, comprobando si se obtienen resultados coherentes con la introducción previa. Así pues, tal y como se muestra en Fig. 5.3, el diseño consiste en que la primera capa da como salida un total de 32 canales y la segunda capa convolucional proporciona como salida 32 canales también. Así pues, tras concatenarlas, se obtienen 64 canales y, consecuentemente, la tercera capa recibe 64 canales. Si se continúa

con dicho proceso, se logra apilar capas convolucionales formando un bloque denso. Los resultados para los distintos optimizadores son los siguientes:

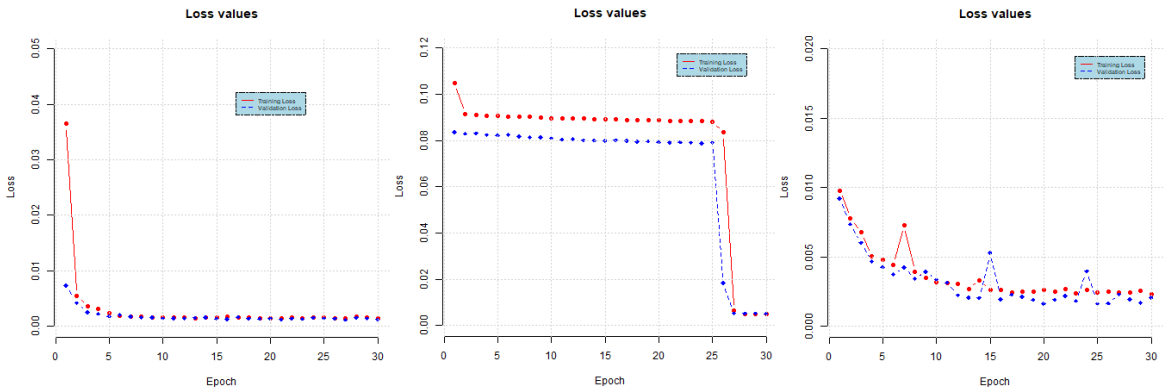


Figura 5.7 MSE en función del número de épocas. La figura de la izquierda se corresponde a Adam, la central a SGD y la de la derecha, a RMSprop para arquitectura de bloque denso.

Optimizer	**Función de coste**	**Valor óptimo**
Adam	*Training*	0.001369
	Validation	0.001189
SGD	*Training*	0.004799
	Validation	0.005043
RMSprop	*Training*	0.002303
	Validation	0.001621

Cuadro 5.3 Valores óptimos de la función de coste en las fases de entrenamiento y validación para la arquitectura de capas convolucionales formando un bloque denso.

ANÁLISIS

Por último, para comprobar si se obtiene una mayor extracción de características, se han realizado estudios análogos pero considerando el conjunto de capas como un bloque denso. Como podemos comprobar, Adam vuele obtener los resultados más óptimos en la función de coste. Además, en este apartado se agravan rotundamente los defectos, tanto de RMSprop como de SGD. En primer lugar, SGD se degrada estancándose en la banda de 0.08, es decir, no se actualizan los pesos correctamente y, con el paso de las épocas, los resultados de MSE no mejoran. Para la representación gráfica de RMSprop, se comprueba cierta inestabilidad en forma de picos diferenciados. No obstante, Adam muestra tanto los mejores resultados de MSE, como una mayor estabilidad y convergencia, sin ningún tipo de degradación, ni *overfitting*.

5.2.4. Discusión de resultados. Establecimiento del modelo

Cuadro 5.4 Resumen de los resultados obtenidos de MSE óptimo en *training* y *validation* para los distintos optimizadores y las distintas arquitecturas propuestas.

Arquitectura	Optimizador	*Training*	*Validation*
Convolucional	**SGD**	0.1668	0.1402
	RMSprop	0.0095	0.0059
	Adam	0.0036	0.0036
Bloques residuales	**SGD**	0.0092	0.0107
	RMSprop	0.0041	0.0024
	Adam	0.0023	0.0025
Bloque denso	**SGD**	0.0047	0.0050
	RMSprop	0.0023	0.0016
	Adam	0.0016	0.0011

ANÁLISIS

A continuación, se van a comparar ciertos aspectos teórico-prácticos para las tres arquitecturas:

- **Arquitectura**

  (a) Para el mismo algoritmo de optimización, podemos comprobar en la anterior tabla como, dado un algoritmo de optimización, los resultados de MSE son más óptimos en el siguiente orden ascendente arquitectónico: bloque denso, bloques residuales y capas convolucionales. Es decir, la arquitectura de bloque denso logra un mejor resultado en términos generales según la métrica MSE.

- **MSE óptimo**.

  (a) Los valores de MSE más óptimos se obtienen para Adam, independientemente de la arquitectura propuesta.

  (b) Los mejores valores obtenidos, tanto en *training* como validación (convergencia más rápida y mejores valores de MSE), se dan para la arquitectura de bloque denso y el *optimizer*, Adaptative Moment Estimation (Adam).

  (c) Las resultados mostrados en las curvas MSE para el algoritmo SGD son inferiores respecto a Adam o RMSprop, independientmente de la arquitectura. El algoritmo RMSprop muestra resultados más susceptibles a la arquitectura, donde se observan ciertas inestabilidades, pese a que la convergencia es rápida. Por último, para todas las arquitecturas, Adam muestra estabilidad (sin picos), mayor rapidez en la convergencia y los resultados más óptimos de MSE.

- **Recursos computacionales**

  (a) El algoritmo de optimización cuyo entrenamiento ha conllevado un mayor tiempo de ejecución ha sido SGD. Tal y como se comentaba en el apartado teórico, SGD es realmente eficiente en base a su cálculo elemento a elemento, sin embargo, dicha variabilidad introducida por la componente aleatoria puede repercutir en la convergencia (actualización ineficiente de los pesos), incrementando considerablemente el tiempo de cómputo. Para el caso de RMSprop y Adam, se comprobaba[4] una diferencia notoria, reduciendo el tiempo de entrenamiento. Aproximadamente, para proporcionar una idea del tiempo requerido, para la misma configuración y tan solo variando el *optimizers*, Adam mostraba un tiempo promedio por época en el entrenamiento de 42 segundos, RMSprop 43 segundos y SGD, 46 segundos.

En base a los distintos aspectos comentados previamente, se considera que los resultados más adecuados se dan para una arquitectura de bloque denso, mediante el algoritmo de optimización de Adaptative Estimation Moment (Adam). En el siguiente apartado, se realizará una búsqueda manual de los hiperparámetros más adecuados, para la arquitectura y optimizador propuestos.

5.3. Optimización hiperparámetros

5.3.1. Learning Rate

Una vez escogida tanto la arquitectura como el método de descenso por gradiente para actualizar los pesos de la red, se ha modificado el valor de la tasa de aprendizaje para dicha configuración. Así pues, a continuación se muestra una tabla con los resultados.

Cuadro 5.5 Función de coste MSE (*training & validation*) para arquitectura de bloque denso y *optimizer*, Adam en función del *Learning Rate*.

Learning Rate	***Training***	***Validation***
0.009	0.0057	0.0067
0.005	0.0014	0.0012
0.001	0.0008	0.0007

[4]Realizando un análisis cualitativo del tiempo medio por iteración/época

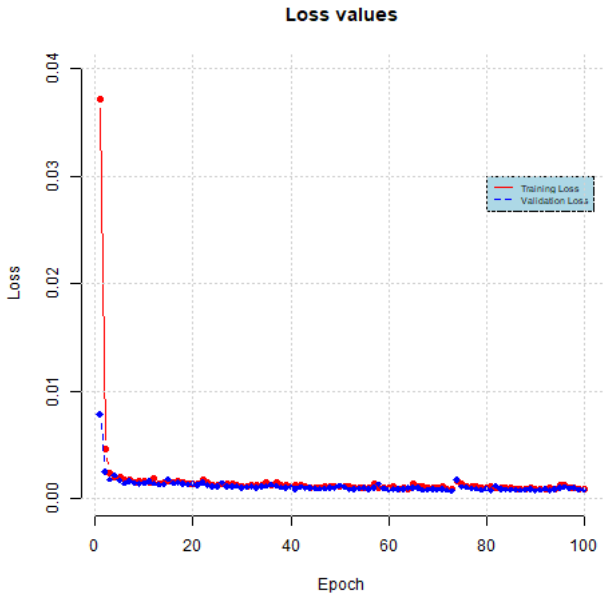


Figura 5.8 MSE en función del número de épocas para arquitectura de bloque denso, optimizador Adam ($\beta_1 = 0{,}9$, $\beta_2 = 0{,}99$) para el *learning rate* óptimo, 0.001.

ANÁLISIS

En cuanto a la modificación de hiperparámetros, se ha iniciado la búsqueda de valores óptimos, comenzando por la tasa de aprendizaje. La tasa de aprendizaje cuyos resultados en entrenamiento y validación son más óptimos respecto a la métrica MSE se corresponde con $\gamma = 0{,}001$. Para dicho valor se comprueba que se alcanza un rango de valores de MSE adecuado para un número inicial de épocas, donde se produce un descenso estable, *mayor convergencia*, menor valor de MSE[5] y no se origina ningún tipo de degradación. Por otra parte, bien es cierto que cuanto mayor es el número de γ, el tiempo de cómputo medio por época es menor (Fig. 5.8). Aun así, la diferencia de eficiencia no es muy apreciable al tomar valores de LR menores. Consecuentemente, para el valor indicado se continuará la búsqueda de hiperparámetros óptima incluyendo regularización L_2.

5.3.2. Regularización L_2

Hay un par de magnitudes que es realmente importante estudiar durante el proceso de entrenamiento y validación: el sesgo y la varianza. El sesgo cuantifica cómo de bien se ajusta el modelo a los datos en el entrenamiento, mientras que la varianza reflejará esa diferencia del promedio del modelo respecto a datos no vistos previamente. Por tanto, si el modelo se ajusta

[5] En *training* y *validation*, respecto al estudio del apartado anterior

perfectamente a los datos en la fase de *training*, su sesgo será alto, pero su capacidad de generalización será baja, es decir, su varianza será elevada. En definitiva, el objetivo siempre es minimizar ambas magnitudes, tanto el sesgo como la varianza. Entonces, dado que no se ha contemplado ningún caso de *overfitting* en las propuestas de modelado anteriores, se ha decidido añadir una regularización L_2, como posible corrector o solución a la alta varianza, modificando los valores de dicha regularización (*weight decay*).

Cuadro 5.6 Función de coste MSE (*training* & *validation*) para arquitectura de bloque denso y *optimizer*: Adam, $\gamma = 0.001$, en función del *weight decay* para regularización L_2.

Weight decay	***Training***	***Validation***
0.001	0.001471	0.001325
0.005	0.001379	0.001261
0.0001	0.001128	0.001044

Mediante la inclusión de una componente de regularización debemos comprender que se intentará disminuir los valores de los pesos, a la vez que la función de coste tratará de ajustarse al modelo, tal y como se estudió en el fundamento teórico. El objetivo es que los valores de los pesos sean menores para reducir la complejidad de la red, maximizando su capacidad de generalización. Para los valores propuestos de *weight decay*, no se ha obtenido ninguna mejora en cuanto a la métrica MSE respecto a los valores de la tabla 5.5. Por tanto, se ha decido que la propuesta de modelo actual siga siendo una arquitectura de bloque denso, con optimizador Adam para un LR de 0.001 y sin inclusión de término de regularización L_2.

5.3.3. Número de capas

Por último, en cuanto a la búsqueda de hiperparámetros óptimos para nuestra red en base a un compendio de criterios (valor MSE, estabilidad, convergencia, etc), se ha decidido incrementar el número de capas convolucionales, donde los resultados de MSE en *training* y *validation* se muestran a continuación:

Cuadro 5.7 Función de coste MSE (*training* & *validation*) para arquitectura de bloque denso y *optimizer*: Adam, $\gamma = 0.001$, en función del número de capas.

Número de capas	***Training***	***Validation***
8	0.000856	0.000836
9	0.000826	0.000709
11	0.000919	0.000760

Si bien antes se tenían 7 capas convolucionales, se han realizado tres entrenamientos distintos para arquitectura de bloque denso, optimizador Adam y *learning rate* de 0.001, con 8, 9 y 11 capas convolucionales. Lógicamente, se observa que a mayor número de capas, mayor tiempo de entrenamiento dada la mayor complejidad de la red. Donde el tiempo promedio por época durante el entrenamiento para N = 8, 9, 11 capas convolucionales ha sido de 42.67, 43.33 y 44.86 segundos respectivamente.

Por otra parte, en la tabla 5.7, podemos comprobar un suceso que ya fue desarrollado en el marco teórico. Cuando incrementamos el número de capas a $N = 8$, los valores óptimos de MSE mejoran respecto a los resultados de la tabla 5.6 para $N = 7$. De igual manera, para $N = 9$, los valores de MSE son mejores que para $N = 8$. Sin embargo, cuando incrementamos el valor a $N = 11$, los valores de MSE no son más óptimos sino todo lo contrario. En resumidas cuentas, el mero hecho de poner más capas, no implica una mejora en la resolución de nuestro problema de optimización no convexo, tal y como se mencionó en el apartado teórico según ciertos aspectos.

5.3.4. Discusión de resultados. Perfilamiento del modelo definitivo

En el anterior apartado relativo a la optimización de hiperparámetros se ha tomado el modelo base propuesto en el apartado 5.2, considerando la arquitectura de bloque denso y el algoritmo de optimización Adam. En el análisis previo sobre los hiperparámetros, se ha determinado el valor óptimo de la tasa de aprendizaje, $\gamma = 0{,}001$, se ha decidido no incluir regularización L_2 y se han tomado 9 capas convolucionales.

5.4. Propuesta definitiva del modelo. Resumen del análisis

En primer lugar, para analizar el rendimiento de la red, se han tenido en cuenta los siguientes aspectos: valores óptimos de la métrica de validación MSE y evolución de la misma frente al número de épocas, tiempo de cómputo, estabilidad y convergencia.

La primera fase de optimización ha permitido analizar los resultados llegando a la conclusión de que, en términos generales, la combinación de topología de bloque denso con el algoritmo de optimización Adam brindaba un mayor rendimiento. Acto seguido, una vez establecida la topología y el *optimizer*, se han modificado los hiperparámetros. Donde se ha encontrado el valor de *learning rate* adecuado, la innecesariedad de incluir un término de regularización y la obtención del número adecuado de capas convolucionales.

Por tanto, el **modelo propuesto** en base al estudio realizado consiste en **9 capas convolucionales**, **sin regularización** y para un ***learning rate*** de 0.001. Además, se ha empleado

la topología de **bloque denso** y el algoritmo de optimización de **Adam**. Además, entre las capas convolucionales, se ha empleado como función de activación **ReLU**. Por último, para realizar la operación de convolución se han tomado **filtros** de dimensión **(3,3)**, con valores de ***padding*** y ***stride*** iguales a **1**.

Para el modelo propuesto, se observan resultados prometedores en base a la métrica MSE, además de otras ventajas como la mejor extracción de características y los esperanzadores resultados en la aplicación a secuencias de vídeo del siguiente apartado. En cuanto a los posibles inconvenientes en el modelo propuesto surge el **requerimiento de memoria**. Es decir, para un gran número de imágenes (o imágenes de grandes dimensiones), se necesitarían mayores recursos computacionales.

En el siguiente apartado, se entrenará el modelo propuesto para un mayor número de épocas, observando los resultados en secuencias de vídeo en aparente movimiento.

Capítulo 6

Revelando lo visible de lo invisible

6.1. Entrenamiento del modelo

Para entrenar el modelo que queremos aplicar, se tomará un número de iteraciones superior al de los apartados anteriores (300 épocas). Al incrementar el número de épocas, se han logrado los valores más óptimos respecto a la métrica MSE (tabla 6.1), así como una rápida convergencia , como podemos observar en la figura 6.1. Además, no hay grandes inestabilidades ya que los picos observados se dan en un rango de valores pequeño (imagen derecha de 6.1).

Cuadro 6.1 Función de coste MSE (*training* & *validation*) para la propuesta de modelo.

Training	***Validation***
0.000445	0.000390

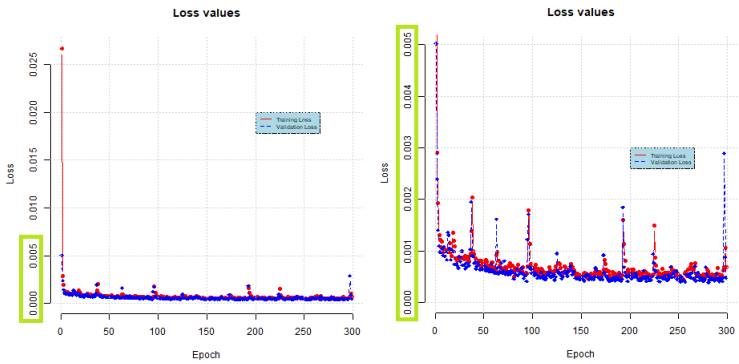


Figura 6.1 MSE en función del número de épocas (300) para el modelo propuesto con arquitectura de bloque denso (9 capas convolucionales), optimizador Adam, $\gamma = 0{,}001$, sin regularización.

Por otra parte, se ha realizado una prueba distinta para la comprobación de calidad de las imágenes generadas (*image quality assessment*). Para ello, se realizó un *script* que tan solo implementaba a nivel matemático el proceso de magnificación detallado en el capítulo 4 para un conjunto de imágenes. Dicho conjunto de imágenes se pasó por la red (*test*) y se almacenaron las imágenes magnificadas. Por tanto, se han aplicado distintas métricas de calidad de imagen entre las imágenes magnificadas a nivel matemático, y las imágenes magnificadas generadas por la red.

En la tabla 6.2 se muestran las distintas métricas de evaluación de calidad de imagen, donde solo se extraerán conclusiones respecto a las métricas explicadas en el apartado 3.3, PSNR y SSIM. El resto de métricas se han mostrado para posibles estudios de interés. Además, cabe destacar que para calcularlas se ha empleado el módulo *piq* [35]. Por tanto, se han tomado 30 imágenes, obteniéndo así los valores de las métricas entre cada par de imágenes magnificadas (obtenida matemáticamente y generada por la red) y se han realizado la media, mediana, varianza y rango intercuartílico de dichos parámetros (tabla 6.2).

Cuadro 6.2 Tabla de métricas de calidad de imágen para un total de 30 imágenes.

Métrica	Media	Mediana	Varianza	Rango intercuartílico
PSNR	28.9508	28.3661	7.4856	3.4184
SSIM	0.9199	0.9212	0.0004	0.0194
MS-SSIM	0.9617	0.9614	0.0001	0.0166
GMSD	0.0833	0.0851	0.0003	0.0202
MS-GMSD	0.0875	0.0896	0.0003	0.0211
HARPSI	0.8246	0.8265	0.0026	0.0605
VSI	0.29854	0.3038	0.0008	0.0205

En primer lugar, los valores típicos que adopta la métrica PSNR están entre 30 y 50 dB. Para nuestro conjunto de imágenes se obtiene un valor promedio de aproximadamente 29 dB. Por tanto, se trata de una calidad en imágenes adecuada ya que se encuentra entre los rangos establecidos en la literatura. Aun así, tal y como se explicó, PSNR solo tiene en cuenta la intensidad, sin tener en cuenta las autocorrelaciones entre píxeles. Para ello, se propone la métrica SSIM que sí que logra captar dicha información estructural. Por tanto, se trata de una métrica que tal y como se definió en la ecuación 3.13, cuantifica la distorsión en base a tres factores: pérdida de correlación, luminancia y contraste. Además, en base a su definición, se encuentra en el rango de valores 0 y 1, donde 0 implica que las imágenes son completamente distintas y 1 que se trata de la misma imagen a nivel estructural. En este proyecto interesa obtener valores altos ya que tan solo se pretende magnificar pequeñas ventanas, donde los cambios entre la imagen original y la imagen magnificada producida por la red deben ser sutiles. Como se puede comprobar, se obtiene un valor de aproximadamente 0.92, lo cual

indica buena calidad de imágenes en el procesado de imágenes propuesto, la magnificación de lo invisible (Fig. 6.2).

Figura 6.2 Representación de una de las imágenes en test para comprobar las métricas de calidad de imagen, donde se remarca la ventana ampliada, en la imagen original (izqda.) y magnificada (dcha.).

En las representaciones posteriores se muestra la imagen original sin ninguna modificación y, al costado, la imagen[1] generada por la CNN propuesta. La red decide magnificar el área aleatoria remarcada posteriormente. Así pues, podemos contemplar en primer plano el funcionamiento de la red aplicado a una imagen antes de aplicarlo a una secuencia de vídeo, que no deja de ser un compendio de imágenes.

6.2. Magnificación de secuencias de vídeo

Para este apartado se han tomado diferentes secuencias de vídeo con distinta dinámica[2], con el objetivo de mostrar tanto la eficiencia de la CNN propuesta como sus limitaciones. De alguna manera, se concibe el proyecto como un trabajo de investigación donde es relevante mostrar tanto los aspectos positivos como los negativos. En cuanto a la dinámica de trabajo para el manejo de vídeos, se ha empleado la librería OpenCV. Donde el *workflow*, consistía en grabar un vídeo sin movimiento aparente, dividir el vídeo en frames, introducir dichos frames a la red y finalmente, unir de manera ordenada los mismos frames magnificados por la red, obteniendo el vídeo magnificado. A continuación, se han grabado 5 vídeos[3] distintos

[1] Se trata de una imagen perteneciente al conjunto de test, donde tan solo se ejecuta la red frente a un conjunto de imágenes previamente no visto y donde no se produce aprendizaje de parámetros, ya que ese paso se realiza en la fase de entrenamiento

[2] Vídeos caseros, ideados sobre la marcha.

[3] Se recomienda ver el vídeo previamente, para entender las ideas indicadas en el análisis.

realizando un análisis de los resultados[4].

Para ver el resultado del proyecto, pinche en:
☞Revealing Invisible

Figura 6.3 Comparativa grabación (real vs. red) vibración de la muñeca.

ANÁLISIS En esta grabación se dispuso el brazo sobre una mesa. El objetivo es magnificar la vibración o el movimiento involuntario de la muñeca del cuerpo humano. Tal y como podemos comprobar en la imagen, no se percibe ninguna diferencia apreciable a primera vista. Sin embargo, es uno de los vídeos que mejores resultados produce. Recordemos que la magnificación es realmente pequeña y, en ocasiones, puede ser realmente costoso ver dicho procesado de imagen de la red. No obstante, en el vídeo se comprueba claramente como el fondo (*background*) no recibe ninguna ampliación, mientras que la muñeca (*foreground*), está en continuo movimiento. Por tanto, se logra muy buenos resultados, pues se magnifica tan solo el objeto susceptible a movimiento [5].

Figura 6.4 Comparativa grabación (real vs. red) de vela encendida.

ANÁLISIS Para este vídeo, se trató de observar el resultado para una vela encendida. De hecho, para evitar fluctuaciones de la llama debidas a las corrientes de aire, se trató de cubrir

[4] En las siguientes imágenes se muestra a la izquierda, un *frame* del video grabado y a la derecha, un *frame* del *frame* resultado de la red.

[5] A lo largo de la toma de secuencias de vídeo, se ha encontrado que si el *background* es fijo y el *foregorund* realiza el pequeño movimiento, los resultados de la red son mejores.

el cuerpo llameante mediante cajas que actuaran como agente protector. En primer lugar, si se observa el vídeo puede parecer que de nuevo la arquitectura propuesta da lugar a resultados satisfactorios ya que magnifica la llama realmente bien. No obstante, si nos fijamos en la imagen anterior, se ha recalcado un problema[6] porque resulta que la red está magnificando el borde vertical de la vela que no se adecúa al objetivo. Lo que ocurre es que aunque la vela aparentemente no fluctúe, los pequeños movimientos generan un cambio de brillo en las distintas regiones de la imagen (contraste) y, por esa razón, la red capta cambios en la propia vela, además de la llama.

Figura 6.5 Comparativa grabación (real vs. red) de taza humeante.

ANÁLISIS En este caso, se tomó una taza con agua que, tras ser calentada en el microondas, emitía vapor de manera continua. Así pues, si se comprueba el vídeo, los resultados son realmente buenos ya que tan solo magnifica el agua en estado gas en movimiento. De hecho, parece que capte el movimiento de las nubes de fondo pero, en realidad, es la magnificación del vapor que debido a su traslucidez produce dicho efecto óptico.

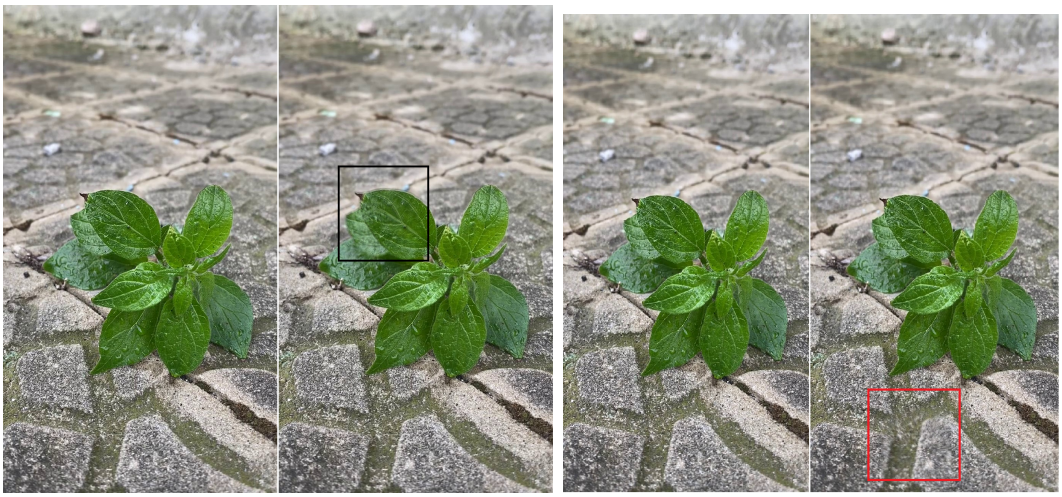


Figura 6.6 Comparativa grabación (real vs. red) de planta bajo la lluvia.

[6]Problema remarcado mediante un marco rojo.

ANÁLISIS Este caso, sin duda, es el que más incertidumbre generó. Se trata de una pequeña planta a la intemperie que recibía tanto los impactos de las gotas de lluvia, como las presiones generadas por las corrientes de aire. El primer par de imágenes capta el movimiento en el contorno de una de sus hojas. Sin embargo, existen unos cuantos frames donde la red capta movimiento en la superficie de la cual nace la planta. Con lo cual, una posible explicación de la amplificación de la red en dicha región indeseada es la situación atmosférica, es decir, las gotas cayendo en forma de calabobos[7]. Este fenómeno nos muestra la importancia de que el *background* en la secuencia de vídeo sea fijo y tan solo varíe el cuerpo susceptible de movimiento (*foregorund*). Por ejemplo, se intentó filmar el movimiento de la cuerda de un tendedero situado en el ventanal de un piso, con la idea de magnificar el movimiento de dicha cuerda ante los impactos del viento paulatino. Sin embargo, la cuerda se magnificaba pero en algunos frames, se observaba cierta magnificación en una esquina. Resulta que estaba captando el movimiento de un coche que circulaba por una rotonda, a la lejanía. Se expone este ejemplo como analogía, donde otro aspecto curioso del proyecto, es tratar de explicar la razón por la cual magnifica una región inesperada (contraste, movimientos no contemplados, etc).

Figura 6.7 Comparativa grabación (real vs. red) de cuerda de mascarilla vibrando.

ANÁLISIS

Para la última grabación, se ha decidido realizar un guiño a la actual situación santiaria causada por la pandemia de la COVID-19. Para ello, se han recortado las cuerdas de una mascarilla, fijando sus extremos y se ha colocado un objeto vibratorio en uno de sus extremos. Así pues, los resultados de la red dan lugar a la magnificación de la cuerda (*foreground*), mientras que la mesa (*background*), no recibía ninguna magnificación.

[7] Lluvia continua cuyas gotas son realmente diminutas.

Capítulo 7

Conclusión

Se ha abordado el problema de **magnificación** de **secuencias de vídeo**. Para ello, se ha planteado una propuesta innovadora respecto al estado del arte, empleando **redes neuronales convolucionales**. En primer lugar, se ha diseñado la **producción sintética** del conjunto de imágenes, donde la CNN recibe dos imágenes, la imagen original y la imagen original ligeramente magnificada, generando como salida la propia imagen magnificada en un factor de escala superior. Por tanto, el objetivo es que los filtros aprendan a realizar la tarea de magnificación. Consecutivamente, se ha confeccionado el **diseño** de la red, evaluando tres **topologías**: capas convolucionales, bloques residuales y bloque denso. También se han considerado tres **algoritmos de optimización**: SGD, RMSprop y Adam. *A posteriori*, se han optimizado distintos **hiperparámetros**: tasa de aprendizaje, regularización y número de capas. Por último, se ha probado el modelo propuesto para **secuencias de vídeo**, donde se obtienen resultados prometedores. Sin embargo, la propuesta es muy sensible ante ligeros movimientos indeseados y presenta **limitaciones** de memoria computacional. Para el desarrollo del trabajo se han utilizado diversas **herramientas**. El código se ha escrito en lenguaje **Python** en el entorno de desarrollo **Visual Studio Code**. Las principales librerías del alto nivel manejadas son **PyTorch**, para la parte estructural, y **OpenCV**, para el tratamiento de imágenes. Los procesos de entrenamiento han sido realizados en un *docker* con acceso a **GPU**. Además, desde Visual Studio Code, se subían los archivos de código a **GitLab** para su revisión. Dicha aplicación también permitía organizar las tareas según la mentalidad **Agile**, además de servir como repositorio. Además, se ha publicado un repositorio para que el lector pueda probar a magnificar sus propios vídeos en GitHub . Así pues, la metodología propuesta se muestra como una **técnica de interés** realmente útil para la magnificación de secuencias de vídeo, dado su poder de extracción de características para una arquitectura realmente sencilla.

Bibliografía

[1] Albawi, S., Mohammed, T. A., and Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6.

[2] Aydin, T. O., Mantiuk, R., Myszkowski, K., and Seidel, H.-P. (2008). Dynamic range independent image quality assessment. *ACM Transactions on Graphics (TOG)*, 27(3):1–10.

[3] Balakrishnan, G., Durand, F., and Guttag, J. (2013). Detecting pulse from head motions in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3430–3437.

[4] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[5] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24. Neural Information Processing Systems Foundation.

[6] Boger, Z. and Guterman, H. (1997). Knowledge extraction from artificial neural network models. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 4, pages 3030–3035. IEEE.

[7] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[8] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer.

[9] Cha, Y.-J., Chen, J. G., and Büyüköztürk, O. (2017). Output-only computer vision based damage detection using phase-based optical flow and unscented kalman filters. *Engineering Structures*, 132:300–313.

[10] Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings.

[11] Coates, A. and Ng, A. Y. (2011). The importance of encoding versus training with sparse coding and vector quantization. In *ICML*.

[12] Dauphin, Y., Vries, H., Chung, J., and Bengio, Y. (2015). Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *arXiv*, 35.

[13] Davis, A., Bouman, K. L., Chen, J. G., Rubinstein, M., Durand, F., and Freeman, W. T. (2015). Visual vibrometry: Estimating material properties from small motion in video. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 5335–5343.

[14] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*.

[15] Deshpande, A. (2019). The 9 deep learning papers you need to know about. understanding cnns part 3.

[16] Diaz-Alejo, L. M. (2020). Análisis comparativo de arquitecturas de redes neuronales para la clasificación de imágenes. *Universidad Internacional de la Rioja*.

[17] Ekman, P. (2009). Lie catching and microexpressions. *The philosophy of deception*, 1(2):5.

[18] Elgharib, M., Hefeeda, M., Durand, F., and Freeman, W. T. (2015). Video magnification in presence of large motions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4119–4127.

[19] Endo, Y., Kanamori, Y., Fukui, Y., and Mitani, J. (2012). Matting and compositing for fresnel reflection on wavy surfaces. In *Computer Graphics Forum*, volume 31, pages 1435–1443. Wiley Online Library.

[20] Feurer, M. and Hutter, F. (2019). Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham.

[21] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377.

[22] Gunther, S., Ruthotto, L., Schroder, J. B., Cyr, E. C., and Gauger, N. R. (2020). Layer-parallel training of deep residual neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):1–23.

[23] Hardesty, L. (2017). Explained: Neural networks ballyhooed artificial-intelligence technique known as "deep learning" revives 70-year-old idea.

[24] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[25] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

[26] Hofmann, C., Sawall, S., Knaup, M., and Kachelrieß, M. (2014). Alpha image reconstruction (air): A new iterative ct image reconstruction approach using voxel-wise alpha blending. *Medical physics*, 41(6Part1):061914.

[27] Hore, A. and Ziou, D. (2010). Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE.

[28] Huang, G., Liu, S., Van der Maaten, L., and Weinberger, K. Q. (2018). Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2752–2761.

[29] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[30] Huang, G., Liu, Z., and Weinberger, K. Q. (2016). Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*.

[31] Ibnkahla, M. (2000). Applications of neural networks to digital communications–a survey. *Signal processing*, 80(7):1185–1215.

[32] Janocha, K. and Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*.

[33] Jiang, X., Hu, B., Satapathy, S. C., Wang, S.-H., and Zhang, Y.-D. (2020). Fingerspelling identification for chinese sign

language viaalexnet-based transfer learning and adam optimizer. *Scientific Programming*.

[34] Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26:315–323.

[35] Kastryulin, S., Zakirov, D., and Prokopenko, D. (2019). PyTorch Image Quality: Metrics and measure for image quality assessment. Open-source software available at https://github.com/photosynthesis-team/piq.

[36] Kelly, J. G. (2001). Talking nets: An oral history of neural networks.

[37] Keskar, N. S. and Socher, R. (2017). Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*.

[38] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[39] Kohl, M., Walz, C., Ludwig, F., Braunewell, S., and Baust, M. (2018). Assessment of breast cancer histology using densely connected convolutional networks. In *International Conference Image Analysis and Recognition*, pages 903–913. Springer.

[40] Konečnỳ, J. and Richtárik, P. (2013). Semi-stochastic gradient descent methods. *arXiv preprint arXiv:1312.1666*.

[41] Kröse, B., Krose, B., van der Smagt, P., and Smagt, P. (1993). An introduction to neural networks.

[42] Kumar, A., Sarkar, S., and Pradhan, C. (2020). Malaria disease detection using cnn technique with sgd, rmsprop and adam optimizers. In *Deep Learning Techniques for Biomedical and Health Informatics*, pages 211–230. Springer.

[43] Lancho, G. (2019). Detección de armas en videos mediante técnicas de deep learning. *Universidad Pública de Navarra*.

[44] LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer.

[45] LeNail, A. (2019). Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747.

[46] Lloret, L. (2021). Convnets (machine learning i). *Instituto de Física de Cantabria (IFCA)*.

[47] Murugan, P. (2017). Hyperparameters optimization in deep convolutional neural network/bayesian approach with gaussian process prior. *arXiv preprint arXiv:1712.07233*.

[48] Nayak, S. (2017). Alpha blending using OpenCV (C++/Python). Technical report.

[49] Nishio, M. (2018). Convolutional neural networks: an overview and an application to radiology. *ResearchGate*.

[50] Oh, T.-H., Jaroensri, R., Kim, C., Elgharib, M., Durand, F., Freeman, W. T., and Matusik, W. (2018). Learning-based video motion magnification. *arXiv preprint arXiv:1804.02684*.

[51] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

[52] Pan, W., Qin, J., Xiang, X., Wu, Y., Tan, Y., and Xiang, L. (2019). A smart mobile diagnosis system for citrus diseases based on densely connected convolutional networks. *IEEE Access*, 7:87534–87542.

[53] Paszke, A., Chaurasia, A., Kim, S., and Culurciello, E. (2016). Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*.

[54] Pedro Larrañaga, I. I. and Moujahid, A. (2021). Tema 8. redes neuronales. *Departamento de Ciencias de la Computaci´on e Inteligencia Artificial Universidad del Pa´ıs Vasco–Euskal Herriko Unibertsitatea*.

[55] Pereira, S., Pinto, A., Alves, V., and Silva, C. A. (2016). Brain tumor segmentation using convolutional neural networks in mri images. *IEEE transactions on medical imaging*, 35(5):1240–1251.

[56] Philipp, G., Song, D., and Carbonell, J. G. (2018). Gradients explode-deep networks are shallow-resnet explained.

[57] Pinzón, J. E. D. (2018). Modelos media móvil simple y media móvil exponencial como pronóstico de la acción de isa. *FACE: Revista de la Facultad de Ciencias Económicas y Empresariales*, 18(1):44–52.

[58] Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. (2017). Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519.

[59] Ponomarenko, N., Lukin, V., Zelensky, A., Egiazarian, K., Carli, M., and Battisti, F. (2009). Tid2008-a database for evaluation of full-reference visual quality assessment metrics. *Advances of Modern Radioelectronics*, 10(4):30–45.

[60] Porter, T. and Duff, T. (1984). Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259.

[61] ProgrammerSought (2021). Resnet or densenet? introducing dense shortcuts to resnet resnet or densenet? the plug-and-play ds rising point artifact is here!

[62] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

[63] Saad Albawi, A. M. (2017). Understanding of a convolutional neural network. *ResearchGate*.

[64] Scornet, E. (2020). Deep learning-optimization. Technical report.

[65] Shi, N., Li, D., Hong, M., and Sun, R. (2021). Rmsprop converges with high proper hyper-parameter.

[66] Simoncelli, E. P. and Freeman, W. T. (1995). The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *Proceedings., International Conference on Image Processing*, volume 3, pages 444–447. IEEE.

[67] Smith, A. R. and Blinn, J. F. (1996). Blue screen matting. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 259–268.

[68] (StatQuest), J. S. Arquitectura de redes neuronales.

[69] (StatQuest), J. S. Neural networks part 8: Image classification with convolutional neural networks.

[70] Street, W. N. (1998). A neural network model for prognostic prediction. In *ICML*, pages 540–546.

[71] Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

[72] Targ, S., Almeida, D., and Lyman, K. (2016). Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*.

[73] Tarrés Benet, L. (2019). Clasificación de lesiones en la piel con un ensemble de redes neuronales residuales. B.S. thesis, Universitat Politècnica de Catalunya.

[74] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[75] Van Laarhoven, T. (2017). L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*.

[76] Vani, S. and Rao, T. M. (2019). An experimental approach towards the performance assessment of various optimizers on convolutional neural network. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 331–336. IEEE.

[77] Verleysen, M. and François, D. (2005). The curse of dimensionality in data mining and time series prediction. In *International work-conference on artificial neural networks*, pages 758–770. Springer.

[78] Wadhwa, N., Rubinstein, M., Durand, F., and Freeman, W. T. (2013). Phase-based video motion processing. *ACM Transactions on Graphics (TOG)*, 32(4):1–10.

[79] Wang, Y., See, J., Oh, Y.-H., Phan, R. C.-W., Rahulamathavan, Y., Ling, H.-C., Tan, S.-W., and Li, X. (2017). Effective recognition of facial micro-expressions with video motion magnification. *Multimedia Tools and Applications*, 76(20):21665–21690.

[80] Wang, Z. and Bovik, A. C. (2002). A universal image quality index. *IEEE signal processing letters*, 9(3):81–84.

[81] Wang, Z., Bovik, A. C., and Lu, L. (2002). Why is image quality assessment so difficult? In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages IV–3313. IEEE.

[82] Wang, Z., Simoncelli, E. P., and Bovik, A. C. (2003). Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee.

[83] Wu, H.-Y., Rubinstein, M., Shih, E., Guttag, J., Durand, F., and Freeman, W. (2012). Eulerian video magnification for revealing subtle changes in the world. *ACM transactions on graphics (TOG)*, 31(4):1–8.

[84] Yu, T., Song, K., Miao, P., Yang, G., Yang, H., and Chen, C. (2019). Nighttime single image dehazing via pixel-wise alpha blending. *IEEE Access*, 7:114619–114630.

[85] Zhang, C., Benz, P., Argaw, D. M., Lee, S., Kim, J., Rameau, F., Bazin, J.-C., and Kweon, I. S. (2021). Resnet or densenet? introducing dense shortcuts to resnet. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3550–3559.

[86] Zhang, K., Guo, Y., Wang, X., Yuan, J., and Ding, Q. (2019). Multiple feature reweight densenet for image classification. *IEEE Access*, 7:9872–9880.

[87] Zhang, L., Zhang, L., Mou, X., and Zhang, D. (2011). Fsim: A feature similarity index for image quality assessment. *IEEE transactions on Image Processing*, 20(8):2378–2386.

[88] Zinkevich, M., Weimer, M., Smola, A. J., and Li, L. (2010). Parallelized stochastic gradient descent. In *NIPS*, volume 4, page 4. Citeseer.

Apéndice A

Distribución de tareas

Para desempeñar un *workflow* de trabajo adecuado, se estructuraban las tareas para ser desarrolladas en *sprint*[1]. Para ello, se habilitó un repositorio en GitLab donde poder subir las últimas ediciones de código, realizar un esquema sobre la posible distribución de tareas (*milestones, labels, lists*, etc). En cuanto a las competencias adquiridas gracias al proyecto, se pueden destacar la inculcación de **buenas prácticas** en programación (jerarquía de clases, definición multidisciplinar de funciones, expresividad de las vriables definidas, etc), adquisición de nuevas herramientas de trabajo (*Visual Studio*, *GitLab*, clúster de computación, etc), organización en la distribución de tareas (similar a los *work packages*) y perseverancia ante la no resolución de inconvenientes.

Tarea 1
Descripción: El objetivo consistía en realizar la producción sintética de las imágenes. Es decir, generar la ventana aleatoria, magnificarla y realizar el α*-blending* en la copia tomando el coeficiente como la máscara gaussiana. Además, se debían configurar los inputs y salidas. Todo ello, mediante buenas prácticas programando, es decir, claridad en la definición de variables, disposición de parámetros en documentos *json* aparte y programación estructurada (funciones y jerarquía de clases).
Nº sprints: 3 (01/02/21 - 15/03/21)

[1] Un *sprint* equivale a 2 semanas según la mentalidad Agile

Tarea 2
Descripción: Diseño de tres arquitecturas: ConvNet, ResNet y DenseNet. Evaluando las métricas óptimas de entrenamiento, validación y curvas en función del número épocas.
Nº sprints: 2 (15/03/21 - 15/04/21)
Aportaciones: En este apartado se decidió aportar mayores pruebas en base a lo aprendido en el máster, extendiendo lo requerido en esta tarea, donde para cada arquitectura se optó por estudiar tres *optimizers* distintos: Adam, RMSprop y SGD. Además, también se incorporó el estudio de hiperparámetros como el *learning rate*, *weight decay* y número de capas.

Tarea 3
Descripción: Evaluación de la arquitectura final para distintas métricas de calidad de la imagen (PSNR, SSIM, MS-SSIM,etc).
Nº sprints: 1 (15/04/21 - 1/05/21)
Aportaciones: Para extender el estudio, se adicionaron más métricas pertenecientes a la rama *image quality assessment*.

Tarea 4
Descripción: Aplicación visual del proyecto a distintas secuencias de video.
Nº sprints: 1 (1/05/21 - 15/05/21)
Aportaciones: Innovación en la grabación de secuencias de video y publicación de las mismas en YouTube (https://youtu.be/ZV32120yU7c).

Tarea 5
Descripción: Consolidación de la memoria
Nº sprints: 3 (15/05/21 - 15/06/21)