Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención en Tecnologías de la Información)

Superresolution

Autor:
D. Miguel Pérez Alonso

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención en Tecnologías de la Información)

Superresolution

Autor:
D. Miguel Pérez Alonso
Tutor:
D. Benjamín Sahelices Fernández

Agradecimientos

Para todos los que me han apoyado en la realización de este proyecto, en especial a mi familia y a mi tutor. Gracias familia por estar ahí animándome de forma incansable.

A mi tutor de la universidad: Benjamín, por enseñarme durante estos años de la carrera, y por introducirme en este campo tan apasionante e interesante que es el de la aplicación de la inteligencia artificial, concretamente en el mundo de las imágenes, así como todo el apoyo y tiempo personal empleando para la culminación de este proyecto.

Resumen

El trabajo de fin de grado es un proyecto de carácter teórico-práctico que consiste en el análisis comparativo de diferentes métodos existentes de interpolación con y sin el uso de inteligencia artificial. El objetivo de estos métodos es redimensionar el tamaño de una imagen sin perder los detalles.

En la memoria se encuentran el análisis de las imágenes obtenidas como resultado de la aplicación de diversos métodos (ESRGAN, interpolación del vecino más cercano, interpolación bilineal, interpolación bicúbica, interpolación de Lanczos), mediante el uso de diferentes métricas como puede ser la métrica proporción máxima de señal de ruido (PSNR).

Palabras claves: **superresolution**, **interpolación**, **inteligencia**, **artificial**, **Python**, **PyTorch**.

Índice general

Índice general **1**

Índice de figuras **2**

Índice de tablas **3**

1. Introducción **4**
1.1. Estructura del trabajo . . . 5

2. Planificación **6**
2.1. Plan de fases . . . 6
2.2. Planificación de las actividades . . . 7
2.3. Recursos . . . 8
2.4. Costes . . . 9

3. Nociones básicas sobre redes neuronales **10**
3.1. Introducción . . . 10
3.2. Neurona de McCulloch-W.Pitts (1943) . . . 12
3.3. Tipos de funciones de activación . . . 14
3.3.1. Función de transferencia . . . 14
3.3.2. Función de paso . . . 14
3.3.3. Combinación lineal . . . 14
3.3.4. Rectificador . . . 14
3.3.5. Algoritmo de pseudocódigo . . . 14
3.4. Entrenamiento . . . 14

4. Estado del arte **16**
4.1. Fotografía digital . . . 16
4.2. Interpolación del vecino más cercano . . . 17
4.3. Interpolación bilineal . . . 17
4.4. Interpolación bicúbica . . . 18
4.5. Interpolación de Lanczos . . . 19
4.6. Redes generativas antagónicas. ESRGAN . . . 19

5. Experimentación **23**
5.1. Metodología . . . 23
5.1.1. *Dataset* . . . 23
5.1.2. Métricas . . . 24
5.2. Replicación . . . 26
5.2.1. Entrenamiento . . . 26
5.2.2. Test . . . 28
5.2.3. Resultados de la replicación . . . 30
5.3. Generación de resultados propios . . . 31
5.3.1. Creación de los *datasets* . . . 32

5.3.2. Entrenamiento . . . 33
5.3.3. Test . . . 33
5.4. Comparación con las técnicas clásicas . . . 34
5.4.1. Implementación de las técnicas . . . 34
5.4.2. Imágenes aleatorias . . . 35
5.4.3. Resultados . . . 35

6. Conclusiones **39**
6.1. Líneas de trabajo futuras . . . 39

Bibliografía **41**

Índice de figuras

2.1. Cronograma 7

3.1. Representación de una neurona 11
3.2. Representación del perceptrón 11
3.3. Esquema de una red neuronal simple [59] 12
3.4. Representación gráfica de una neurona y los tipos de conexiones [60] 13
3.5. Ejemplo de funciones lógicas [60] 13

4.1. Comparación de los métodos con una imagen real [26] 17
4.2. Ejemplo.- Interpolación del vecino más cercano 17
4.3. Ejemplo.- Interpolación bilineal [43] 18
4.4. Ejemplo.- Interpolación bicúbica 18
4.5. Comparación distintas formas de interpolación [44] 18
4.6. Gráfica de la ventana de Lanczos $sinc(x/a)$ [45] 19
4.7. Representación de los distintos sistemas [84] 20
4.8. Esquema del discriminador [84] 20
4.9. Esquema del generador [84] 20
4.10. Arquitectura del generador [84] 20
4.11. Esquema de la interacción del generador y el discriminador 21
4.12. Arquitectura de la red [67] 22

5.1. Diagrama de casos de uso 25
5.2. Resultado de *test.py* en el set5 29
5.3. Resultado de *test.py* en el set14 30
5.4. Comparación resultados en baboon.png 31
5.5. Comparación métodos en comic.png 31
5.6. Diagrama de casos de uso 34
5.7. Comparación métodos en animal.png 35
5.8. Comparación métodos en comida.png 36
5.9. Comparación métodos en monumento.png 37
5.10. Comparación métodos en paisaje.png 37
5.11. Comparación métodos en persona.png 38

Índice de tablas

2.1. Fases del proyecto 7
2.2. Duración de las fases del proyecto 7
2.3. Recursos 8

5.1. Descripción del caso de uso 02 26
5.2. Comparativa I (en decibelios) 30
5.3. Comparativa II 30
5.4. Test del propio entrenamiento 33
5.5. Descripción del caso de uso 02 34
5.6. Comparación métricas animal 36
5.7. Comparación métricas comida 36
5.8. Comparación métricas monumento 37
5.9. Comparación métricas paisaje 38
5.10. Comparación métricas persona 38

Capítulo 1

Introducción

Con el objeto de dar cumplimiento a la legislación vigente sobre la ejecución de los Trabajos de Fin de Grado, TFG, y en concreto, lo establecido en RESOLUCIÓN de 11 de abril de 2013, del Rector de la Universidad de Valladolid, por la que se acuerda la publicación del reglamento sobre la elaboración y evaluación del trabajo de fin de grado (aprobado por el Consejo de Gobierno, sesión de 18 de enero de 2012, «B.O.C. y L.» n.º 32, de 15 de febrero, modificado el 27 de marzo de 2013). Normativa de la UVA, se redacta el presente trabajo.

Este proyecto es una de las propuestas presentadas por la Universidad de Valladolid. Asimismo se barajó la posibilidad de realizar, entre otras propuestas de trabajos de investigación el proyecto denominado *SmartFiller* consiste en aplicar técnicas de aprendizaje automático (*Machine Learning*) para reconstruir partes eliminadas dentro de una imagen, y rellenando automáticamente los huecos (partes eliminadas) de la imagen, así como el proyecto de *Superresolution* que se desarrolla.

Después de valorar los dos proyectos, se decide por la realización del proyecto *Superresolution* por considerar que es un campo muy apasionante, que suscita la curiosidad sobre los actuales métodos en la realización de ampliación de una imagen, sin pérdida de resolución y ser de gran utilidad en el día de hoy para los usuarios. Este proyecto ha sido pilotado y tutorizado por D. Benjamín Sahelices Fernández.

En el campo de la fotografía digital, la calidad se degrada considerablemente cuando se quiere ampliar el tamaño de visualización o de impresión más allá de lo que permite su resolución.

Ante esta situación, se han desarrollado numerosos algoritmos convencionales que implementan herramientas matemáticas para interpolar la información de las imágenes y generar una salida de un mayor tamaño que la original (interpolación bilineal, bicúbica,...).

Estas funciones, y en mayor o menor medida, todos ellos acaban generando artefactos no deseables en una imagen, como pueden ser:

- Exceso de suavizado en los contornos: lo que se traduce en una falta de nitidez en la imagen y en el detalle de los bordes de los objetos de la imagen.
- Pérdida de detalle en los pequeños patrones: al utilizar modelos de interpolación, algunos patrones de la imagen original se pierden por considerarse "outliers".
- Los algoritmos tradicionales no son capaces de reconstruir partes no existentes o poco definidas de la imagen original, aun existiendo información contextual que permitiría extraer una aproximación aceptable. Por ejemplo, en una imagen donde aparezca un rostro muy lejano, en la que no se aprecian los detalles de los distintos rasgos de la cara (boca, nariz, ojos...) y que al ampliarse se obtiene solamente un rostro emborronado donde no se evidencian los detalles.

En los últimos años, gracias al avance de las redes neuronales artificiales, se han llevado a cabo experimentos que permiten aprender características de las imágenes pudiendo ir más allá de una simple interpolación.

Por ejemplo, en los modelos GAN (modelos generativos adversariales), se entena un componente generador y un componente discriminador que compiten para conseguir modelos de gran calidad a partir de los datos de entrada, pudiendo reconstruir elementos no existentes a priori en la imagen original, a partir de la información obtenida durante el proceso de entrenamiento.

Este proyecto realiza un análisis comparativo sobre los diversos métodos utilizados para la ampliación del tamaño de una imagen entre los cuales habrá métodos clásicos y métodos modernos que se ayudan de los avances de la inteligencia artificial; cuyos objetivos son:

- Analizar los métodos.
- Experimentación con los mismos.
- Comparación de resultados.

1.1. Estructura del trabajo

Al desempeñar un análisis comparativo se ha realizado una parte teórica y otra práctica. En la parte teórica se han buscado y analizado qué métodos existen y su funcionamiento, mientras que en la parte práctica se han ejecutado y se han comparado los resultados.

Incluyendo los apartados básicos de un proyecto, la estructura del trabajo es el siguiente:

- **Capítulo 1: Introducción**
  Presentación del contenido que se desarrolla en el transcurso del presente documento.
- **Capítulo 2: Planificación**
  Se analiza la organización que se ha seguido durante la realización del proyecto y los recursos utilizados.
- **Capítulo 3: Nociones básicas sobre redes neuronales**
  Se describe nociones básicas acerca de las redes neuronales como pequeña introducción al tema.
- **Capítulo 4: Estado del arte**
  Se describe la problemática de las imágenes, junto con las posibles soluciones que se han ido encontrando, por los diferentes métodos clásicos y con inteligencia artificial.
- **Capítulo 5: Experimentación**
  Se detalla el proceso de experimentación mediante la realización de diversas pruebas y la comparación de los resultados obtenidos.
- **Capítulo 6: Conclusiones**
  Se sintetizan las diferentes conclusiones obtenidas y las futuras líneas de trabajo que tiene este proyecto.
- **Bibliografía**
  Se recopilan todas las fuentes consultadas en la elaboración del trabajo.

Capítulo 2

Planificación

En este capítulo se describe los distintos aspectos de la planificación del trabajo, considerados relevantes, que han sido utilizandos a lo largo de la realización del proyecto, teniendo en cuenta, las fases implementadas, las actividades desarrolladas, así como la cuantificación económica del desarrollo.

2.1. Plan de fases

Como se indicó brevemente en el apartado 1.1 'Estructura del trabajo' del capítulo 'Introducción' las principales fases del proyecto son dos: la parte teórica y la parte práctica.

La parte teórica fue la primera fase a realizar en el proyecto. En la cual se ha buscado información acerca de la temática de este trabajo de fin de grado, desde la adquisición de los conocimientos básicos hasta la comprensión de dichos métodos para en la fase siguiente, fase práctica o fase de experimentación, poder analizar los resultados que se van a obtener. Se desgrana esta fase en las siguientes tareas:

* **Búsqueda de los diferentes métodos**: ha consistido en buscar los diferentes métodos de *superresolución* existentes.
* **Comprensión de los métodos**: una vez seleccionado un método, éste se analiza para entender su funcionamiento y se experimenta, además se ha buscado el vocabulario asociado con el objetivo de conseguir un mejor entendimiento del método.
* **Replicación ESRGAN**: obtener resultados muy aproximados a los conseguido por los propios desarrolladores, siguiendo todos y cada uno de los pasos dados.

La segunda fase del proyecto es la fase práctica o de experimentación. Se ha experimentado con los diferentes métodos, obteniendo unos resultados que posteriormente se han analizado, visualizando cuál se aparece más a la imagen original y la calidad que posee.

* **Experimentación**: ejecución del método ESRGAN con diferentes imágenes de entrenamiento, de validación y test. Además de la ejecución con los distintos métodos a unas imágenes concretas.
* **Búsqueda de métricas**: se encuentra un valor o conjuntos de valores que permitan la comparación de los diversos resultados de forma objetiva e inequívoca.
* **Análisis de los resultados**: se realiza un análisis comparativo entre los diversos resultados obtenidos al aplicar los métodos en una imagen.
* **Elaboración conclusiones**: se sintetizan todas las ideas obtenidas mientras se realizaba el proceso de desarrollo de la experimentación, además de las posibles conclusiones parciales que se han ido surgiendo entre tanto.

La siguiente tabla recoge de forma ordenadas y a golpe de vista las fases del proyecto y las tareas implicadas en cada fase.

Fases	Tareas
Fase de Documentación	Búsqueda de los diferentes métodos
	Comprensión de los métodos
	Replicación ESRGAN
Fase de Experimentación	Experimentación
	Búsqueda de métricas
	Análisis de los resultados
	Elaboración conclusiones

Tabla 2.1: Fases del proyecto

2.2. Planificación de las actividades

Identificadas todas las tareas a realizar en el proyecto, se hace una estimación del tiempo que va a llevar su realización. Todas las duraciones de las distintas fases, medidas en semanas, están recogidas en la siguiente tabla:

Fase	Duración
Fase de Documentación	**3 semanas**
Búsqueda de los diferentes métodos	1 semana
Comprensión de los métodos	2 semanas
Replicación ESRGAN	2 semanas
Fase de Experimentación	**8 semanas**
Experimentación	6 semanas
Búsqueda de métricas	2 semanas
Análisis de los resultados	4 semanas
Elaboración conclusiones	2 semanas

Tabla 2.2: Duración de las fases del proyecto

Conocidas las diferentes duraciones de las tareas, se planifican a lo largo del tiempo, teniendo en cuenta el orden de estas y la dependencia o no con otras tareas.

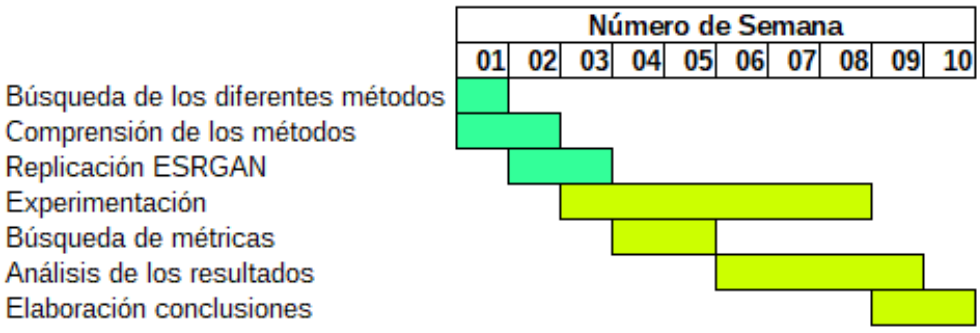


Figura 2.1: Cronograma

Viendo la distribución de las tareas a lo largo de las semanas, se concluye que en el transcurso de 10 semanas se habrán realizado de manera óptima todas las fases que componen dicho proyecto.

2.3. Recursos

Todos los proyectos necesitan recursos para su desarrollo y elaboración, aunque estos varían en función del tipo de proyecto que sea. En el transcurso de este proyecto los principales recursos que se van a utilizar se encuentran recogidos en la tabla situada a continuación:

Recurso	**Tipo**
Ordenador	Hardware
Latex	Software
Python	Software
PyTorch	Software
Numpy	Software
OpenCV	Software

Tabla 2.3: Recursos

Se procede a la explicación de cada recurso indicando sus características, el contexto en que se necesitan, indicando si existe alguna posible alternativa en caso de que se produzca algún fallo o error de gran relevancia para la elaboración del proyecto.

El **ordenador** portátil se utiliza para la búsqueda de información, ejecución del programa además de la redacción del presente documento. Las características son VAIO con Intel Core 2 DUO T8300 @ 2.40GHz, 3.8 GB de memoria, 229 GB de disco y sistema operativo Ubuntu 18.04. La posible alternativa debido a la antigüedad del portátil es una máquina virtual solicitada al tutor con una memoria de 4 GB, disco de 20 GB y sistema operativo Ubuntu 18.04.

Con el programa **Latex** se escribe la presente documentación. En el caso de que éste no hubiera funcionado, se hubiera utilizado cualquier otro programa capaz de procesar textos como puede ser Microsoft Word, LibreOffice Writer u otro similar.

El lenguaje de programación **Python** para la codificación y ejecución de todos los programas, desde los que implementan los diferentes métodos hasta los que calculan las diferentes métricas. La alternativa a este lenguaje de programación son los utilizados para el desarrollo de programas para la inteligencia artificial como son el R, el C++ y Java.

La librería **PyTorch** que utiliza en el aprendizaje profundo automático a través de optimizaciones de la CPU y GPU. En el trabajo es parte del programa de que permite la escalada de una foto mediante inteligencia artificial. Una posible alternativa puede ser *TensorFlow*, herramienta desarrollada por Google para el aprendizaje automático.

El empleo de la librería **Numpy** se limita, por ser un requisito que debe poseer el sistema donde se quiere instalar el PyTorch además de ser usada en el método con inteligencia artificial.

La librería **OpenCV** permite la lectura de imágenes y sus transformaciones; su utilización se dirige a temas de visión artificial, es decir, adquisición, procesamiento y análisis. Sin embargo se puede usar SimpleCV como posible alternativa ante cualquier imprevisto.

Como se puede observar, la mayoría de los recursos usados en el desarrollo del proyecto son del tipo software. Las versiones vienen especificadas por el método con redes neuronales que son Python 3 (Anaconda), PyTorch versión igual o superior a la 1.0 . En el caso de que fallaran el python y todas las librerías asociadas a este, habría que convertirlo todo a otro lenguaje de programación, modificándose completamente con la finalidad de adecuarse a las posibles alternativas de las librerías.

2.4. Costes

Sabiendo los recursos que se van a requerir a lo largo del tiempo, mientras se realiza el propio trabajo, se estima el coste que supone la adquisición de dichos recursos:

- El ordenador de características parecidas al utilizado por el autor del trabajo, cuyo coste es 645,00 €.
- La máquina virtual es facilitada por el tutor de la universidad. El precio es 0 € aunque existen diversas páginas que facilitan una y el precio va en función de cuanto se utilicen (se calculará el precio que costaría más adelante).
- El software de Latex al poseer licencia pública tiene un coste de 0 €.
- El lenguaje de programación Python posee su propia licencia la cual indica que es software libre. El coste de adquisición es nulo (0 €).
- Al ser la librería PyTorch de código abierto se puede adquirir de manera totalmente gratuita.
- La librería Numpy y OpenCV poseen licencia de software libre permisiva. Significa que se puede ser redistribuida como libre por lo que su precio es gratuito.

Se calcula el precio que hay que pagar a los trabajadores en este proyecto que será sólo una única persona. Para saber el precio por hora se consulta el Convenio de Consultoría ("BOE número 57, de 6 de marzo de 2018"). El salario que se debería pagar al trabajador oscilaría entre los 9 y 10 euros la hora. El número de horas que dura el desarrollo de este proyecto son 300 horas, según lo indicado en la guía docente del trabajo final de grado de la Universidad de Valladolid.

Conocidos todos los datos se calcula el coste del proyecto:

$$645\ € \text{ (Recursos)} + 300 \text{ horas} * 9,5\ €_{hora} \text{(promedio del salario)} = 3495\ €$$

La realización del proyecto ascendería a 3495 € si se concede la petición de solicitud de la máquina virtual.

Si la petición es denegada, se calculará el precio de contratar una máquina virtual. De todas las opciones facilitadas por Azure Microsoft [25] la que más se ajusta al proyecto es 'Uso general – Dv3' cuyo precio es $0,0106€_{/hora}$. Se suponen dos casos, uno en el que se usa en todo el transcurso del proyecto (300 horas) y el otro solo la mitad (150 horas).

- Todo el proyecto: 300 horas $* 0,0106\ €_{/hora} = 3{,}18\ €$.
- Mitad del proyecto: 150 horas $* 0,0106\ €_{/hora} = 1{,}59\ €$.

El coste del proyecto son 3495 € con la petición de la máquina virtual. Alquilando la máquina virtual durante todo el proyecto son 3498,18 € mientras que si se utiliza la mitad del tiempo son 3496,59 €.

Capítulo 3

Nociones básicas sobre redes neuronales

Cuando hablamos de una red neuronal artificial nos referimos a modelos matemáticos que emulan el comportamiento de una neurona biológica de los seres vivos. Se habla de estructuras similares a las cerebrales porque están interconectadas y trabajan juntas, íntimamente unidas entre sí. "Aprenden" de las experiencias anteriores, obtienen una retroalimentación y se adaptan a las realidades cambiantes. Así haremos una breve descripción de los elementos que componen una neurona fisiológica que nos permitirá entender el comportamiento de la artificial.

En este capítulo trataremos el primer modelo neuronal moderno, la neurona definida por McCulloch - W.Pitts en 1943, que sirve de inspiración a muchos otros modelos actuales. Así como los elementos de las distintas capas de entrada, capas ocultas y capas de salida.

Estos modelos matemáticos son variados y, según el algoritmo utilizado, podemos tener distintas clasificaciones como son la tipología de la red, las funciones utilizadas y el tipo de aprendizaje.

3.1. Introducción

Una Red Neuronal Artificial (RNA) es un modelo matemático que imita el comportamiento biológico y estructural del cerebro de los seres vivos. Al ser una agrupación de millones de neuronas interconectadas trabajando en conjunto, hacen que el cerebro se considere como un sistema altamente complejo. Con la experiencia se van creando nuevas conexiones y/o reforzando las ya existentes.

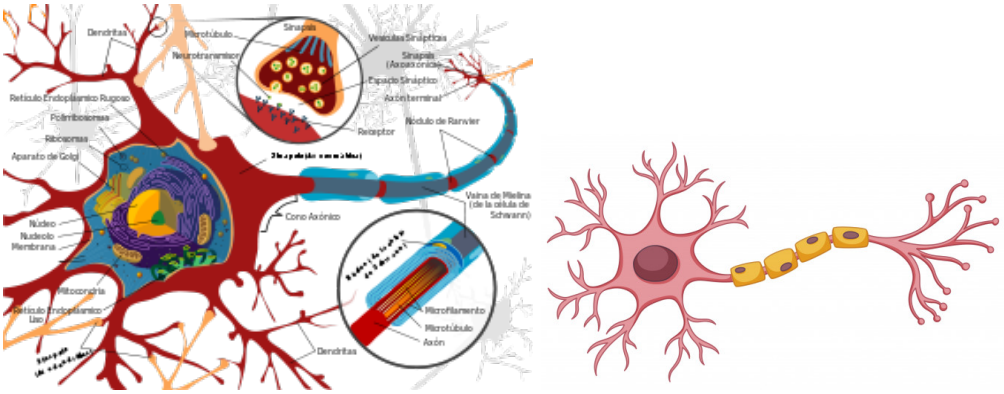
[54]

La unidad fundamental del sistema nervioso es la **neurona**. La neurona biológica fue descubierta por Ramón y Cajal en 1888 y consta de las siguientes partes:

- **Soma**: es el cuerpo celular donde se encuentra el núcleo. Funciona como si fuera un procesador.
- **Dendritas**: son las densas ramificaciones terminales que reciben la información procedente de otras. Actúan como los canales de entrada de la información.
- **Axón**: prolongación delgada de forma tubular del soma que transmite la información hacia otra neurona. Actúa como el canal de salida de la información.

La **sinapsis** es la zona de conexión no física entre neuronas, es decir, no se tocan entre ellas. Para transmitir la información entre neuronas se utilizan unas sustancias químicas específicas denominadas neurotransmisores, mientras que la información dentro de una neurona se transmite como un impulso eléctrico.

Al existir muchos tipos de neuronas, éstas se clasifican según diversos criterios como pueden ser según su función (sensorial, motora) o según la dirección del impulso nervioso (aferente, eferente). Todas ellas actúan de la misma forma y tienen el mismo esquema.

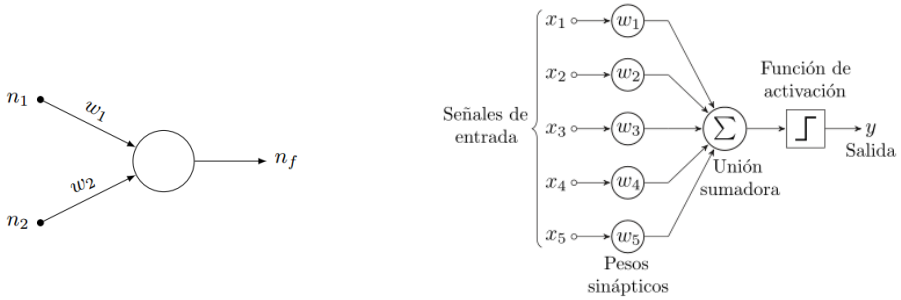


(a) Real [55] **(b)** Simplificada [56]

Figura 3.1: Representación de una neurona

El análisis de este modelo neuronal, desde el punto de vista biológico y con el fin de poder explicar el funcionamiento del cerebro y todos los procesos asociados, es de especial interés para los grupos de estudio y los primeros modelos neuronales, permitiéndonos desarrollar sistemas "inteligentes". El modelo matemático propuesto por McCulloch fue el primer ejemplo de esta forma de proceder. Cada una de estas neuronas recibe un conjunto de entradas, las combina y, finalmente genera una señal de activación.

En la primera generación de los modelos neuronales se hacía referencia a los **perceptrón** o compuertas de umbral, se trata de un tipo de red neuronal fundamental que usa un tipo concreto de neurona y función de activación, y se caracterizan por su salida digital, teniendo un funcionamiento similar al de una neurona fisiológica.



(a) Simplificado [57] **(b)** Detallado [63]

Figura 3.2: Representación del perceptrón

En la figura 3.2(a) vemos el esquema de un perceptrón que tiene dos entradas de la capa anterior (n_1 y n_2) con diferentes pesos (w_1 y w_2). Este valor es el que permite dar mayor o menor importancia a una conexión frente a otras y, una vez realizada la entrada con su función de activación concreta, se nos transforma en una salida n_f acorde a los parámetros introducidos. En la figura 3.2(b) se ve el perceptrón con más detalle con una función sumadora.

Los neuronas se agrupan en unidades estructurales llamadas capas, que trasmiten la información en un solo sentido. Existen tres tipos de capas diferentes:

- **Capa de entrada**: Primera capa, en la se reciben datos procedentes del entorno (programa).
- **Capa oculta**: Situada entre la capa de entrada y la capa de salida, se genera el procesamiento interno de la red.
- **Capa de salida**: Última capa de la red neuronal, en la que se producen las salidas para el programa a partir de la información que recibe en la entrada.

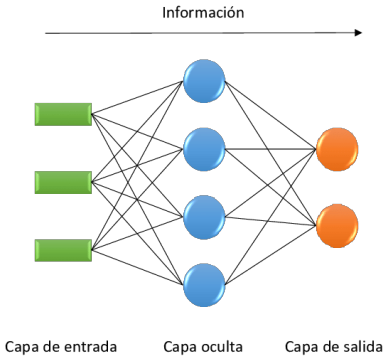


Figura 3.3: Esquema de una red neuronal simple [59]

En la figura 3.3 se observa cada una de las conexiones de las neuronas precedentes. En la capa de entrada se comunican con todas y cada una de las conexiones de la capa oculta, interactuando todas con todas, con un gran peso computacional dado el elevado número de combinaciones. Una vez que pasan por la capa oculta permiten obtener una salida más sencilla, reduciendo el tamaño según va avanzado en la estructura. Esta capa oculta es representativa y no es única, pudiendo ser varias. Cada una de ellas se especializa en un elemento significativo de la imagen. Según se va acercando al final de la capa, éstas tienen mayor grado de abstracción del modelo establecido.

Red neuronal convolucional

Son algoritmos utilizados en el aprendizaje automático. Se basa en los operadores del procesamiento de imágenes. Está formada por una capa de entrada, varias capas ocultas y una capa de salida, por donde los "kernels" utilizados pueden ser "aprendidos" y no fijados manualmente. Es muy similar a la transformación que lleva a cabo una neurona de McCulloch-W.Pitss, que describiremos en el siguiente punto. Las capas ocultas pueden ser convolucionales (el peso de sus entradas es igual para ellas, además de reducir el número de parámetros). Su utilización principal es el procesamiento de imágenes y lenguaje natural.

3.2. Neurona de McCulloch-W.Pitts (1943)

El modelo de McCulloch - W. Pitts fue el primer modelo neuronal moderno desde el punto de vista computacional, y sirvió de inspiración a otros modelos actuales. Se basaba en los conocimientos que había en el año 1943 de las neuronas biológicas, pero no en sus aspectos fisiológicos y/o morfológicos, sino en la forma de actuar totalmente "matemática".

El potencial de este modelo, sigue vigente por las ideas que propone. Aprovecha la capacidad de representación, tanto lógica como digital, fácilmente implementable en los actuales dispositivos (siendo uno de los escasos modelos digitales) y permite trabajar en tiempo discreto, parte de cinco consideraciones del comportamiento propio de las neuronas:

- La actividad neuronal es un proceso de todo/nada.
- Un cierto número concreto de sinapsis debe ser excitado dentro de un periodo de adición latente, en orden a excitar una neurona en cualquier intervalo. Este número es independiente de la actividad previa y la posición de la neurona.
- El único retardo significativo dentro del sistema es el retardo sináptico.
- La actividad de cualquier sinapsis inhibitoria previene absolutamente la excitación de la neurona en ese intervalo de tiempo.
- La estructura de la red no cambia con el tiempo.

Este tipo de neurona es un dispositivo binario que puede recibir entradas de las sinapsis excitadoras y de las sinapsis inhibitorias. Si está activada la sinapsis inhibidora, la neurona no se puede activar con ningún valor.

Existe un lapso de tiempo fijo para la integración de las entradas sináptica, basado en el retardo sináptico. Confiere a la neurona su carácter de trabajo en tiempo concreto.

Los autores inician trabajos con arreglos lineales. Consideran que no hay vías de retroalimentación entre neuronas, y realizan una serie de simplificaciones en este lapso de tiempo, llamado umbral de disparo en la neurona, que adopta valores discretos y que se mantienen constantes.

Estas premisas les permiten demostrar el funcionamiento de la neurona y, posteriormente, mediante la realización de arreglos y conexiones, reproducir los resultados con umbrales cambiantes en el tiempo.

La neurona responde a la actividad sináptica en el tiempo de integración, contemplando incluso las células presinápticas. Si no existen sinapsis inhibitorias activas, la neurona suma sus entradas sinápticas y verifica si la suma es igual o mayor a su nivel de umbral. Si es así, la neurona se vuelve activa; de lo contrario, permanece inactiva.

En el caso de que exista sinapsis inhibitoria activa, la neurona permanece de igual forma, inactiva.

Figura 3.4: Representación gráfica de una neurona y los tipos de conexiones [60]

A través de la figura 3.5, podemos comprender la forma de funcionamiento:

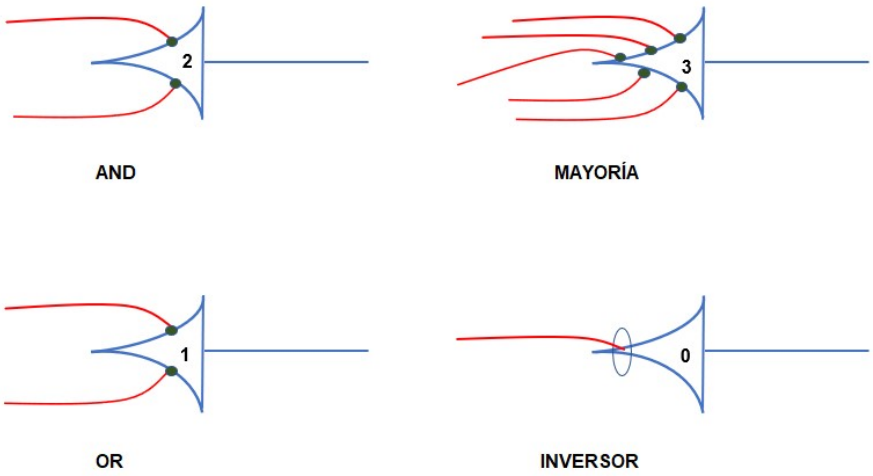


Figura 3.5: Ejemplo de funciones lógicas [60]

AND: Con dos sinapsis excitatorias a la entrada, considerando unitario el peso de cada una de éstas e indicado que la neurona tiene un umbral de dos unidades, la salida se activará si ambas están activas. En el caso de que alguna de las dos entradas estuviera inhibida, esta salida no se producirá.

OR: De igual forma, una única entrada nos puede excitar el umbral y darnos una salida.

Inicialmente se partió de unas aproximaciones de igual peso, todo/nada, no era posible tomar registros intracelulares, etc.., pero se pueden realizar proposiciones muy complejas. No obstante McCulloch y Pitts consideraban que el poder computacional venía porque estas neuronas simples están insertas en un sistema nervioso interactuante.

3.3. Tipos de funciones de activación

A continuación se realiza una breve descripción de una serie de tipos de funciones utilizadas en las redes neuronales.

3.3.1. Función de transferencia

La función de transferencia o de activación de una neurona se elige para tener una serie de propiedades que mejoren o simplifiquen la red que contiene la neurona. Obedece a la fórmula:

$$u = \sum_{i=1}^{n} w_i x_i$$

donde u se refiere en todos los casos a la suma ponderada de todas las entradas a la neurona (n entradas), w es un vector de pesos sinápticos y x es un vector de entradas.

3.3.2. Función de paso

La salida y de esta función de transferencia es binaria, dependiendo de si la entrada cumple un umbral específico θ. Se envía la "señal" y la salida se establece en uno si la activación alcanza el umbral.

$$y(x) = \begin{cases} 1 \text{ si } x \geq \theta \\ 0 \text{ si } x < \theta \end{cases}$$

Esta función a menudo aparece en muchos otros modelos. Es especialmente útil en la última capa de una red destinada a realizar una clasificación binaria de las entradas.

3.3.3. Combinación lineal

En este caso, la unidad de salida es una suma ponderada de sus entradas más un término de sesgo que permite realizar una transformación afín a los datos.

3.3.4. Rectificador

El rectificador es una función de activación definida como la parte positiva de su argumento donde x es la entrada a una neurona.

$$f(x) = x^+ = max(0, x)$$

Esta función de activación, también conocida como función de rampa, fue introducida por primera vez en una red dinámica por Hahnloser. En 2011 se demostró que permite un mejor entrenamiento de redes más profundas, en comparación con las funciones de activación utilizadas con anterioridad.

3.3.5. Algoritmo de pseudocódigo

Un pseudocódigo es una descripción informal, simple pero de alto nivel, del principio operativo de un programa u otro algoritmo de una unidad lógica de umbral que toma entradas booleanas (verdadero/falso), devolviendo un única salida cuando se activa.

3.4. Entrenamiento

El entreanmiento es el proceso por el cual la red neuronal va actualizando los pesos, con el fin de optimizar con la máxima eficiencia la tarea designada.

Los conceptos que hay que tener en consideración son:

- **Paradigma**: información disponible para la propia red.
- **Regla**: principio que dirige el aprendizaje.

- **Algoritmo**: procedimiento numérico que se aplica en el ajuste de los pesos.

Los paradigmas de aprendizaje más fundamentales son el supervisado y el no supervisado. El **Aprendizaje supervisado** consiste en minimizar los errores de la salida generada respecto a la salida deseada y conocida, mientras que en el **Aprendizaje no supervisado** la red extrae rasgos y patrones para generar una salida que es desconocida.

El proceso de aprendizaje de las redes neuronales artificiales es un proceso complejo que suele llevar altos tiempos de computación, haciendo el proceso muy lento. En él se pueden aplicar diversos algoritmos como puede ser minimizando los errores (**Minimización del error**), fijando parámetros aleatorios (**Boltzman**), aplicando la ley de Hebb (**Hebb**) o aprendiendo de las neuronas que generan un valor más cercano al deseado en la salida (**Competitivo**).

Finalmente cabe destacar que el proceso de aprendizaje o entrenamiento no es simple, utilizando grandes recursos desde el punto de vista computacional.

Capítulo 4

Estado del arte

En los siguientes apartados realizaremos un estudio acerca de los diversos modelos matemáticos de interpolación aplicados a la fotografía digital, con un foco más exhaustivo en el modelo con red neuronal ESRGAN por ser el modelo que se replica en el presente proyecto.

4.1. Fotografía digital

En el campo de la fotografía digital, la resolución de las imágenes es una propiedad importante que viene asociada al número de píxeles que la componen y permite la observación de los detalles existentes, además de la calidad visual. Pero cuando las imágenes se quieren ampliar de forma considerable, más allá de su propia resolución, éstas van degradando su calidad.

Algunas operaciones permiten extraer información útil y generar una imagen optimizada. Una de esas operaciones que soluciona el comentado problema de pérdida de calidad es el escalado. Escalar una imagen consiste en cambiar el tamaño físico de la imagen, ya sea para agrandarlo o reducirlo. Es decir, se adapta el número de píxeles que forma dicha imagen a esas determinadas dimensiones

Al aumentar el tamaño de la imagen, ésta tendrá un mayor número de píxeles. Pero, ¿cómo saber qué valor van a tomar los nuevos píxeles a partir de la que se parte? Una de las respuestas a esa pregunta puede venir de la interpolación, pero no es la única. Es una solución que se aplica en los métodos clásicos.

La interpolación es un proceso matemático por el cual se aplican fórmulas concretas para calcular valores (desconocidos) a partir de otros datos conocidos.

Un ejemplo de interpolación lineal sería:

1. Se tienen los puntos $M_1(2, 3)$ y $M_2(5, 8)$.

2. Se aplica la siguiente fórmula con esos puntos:

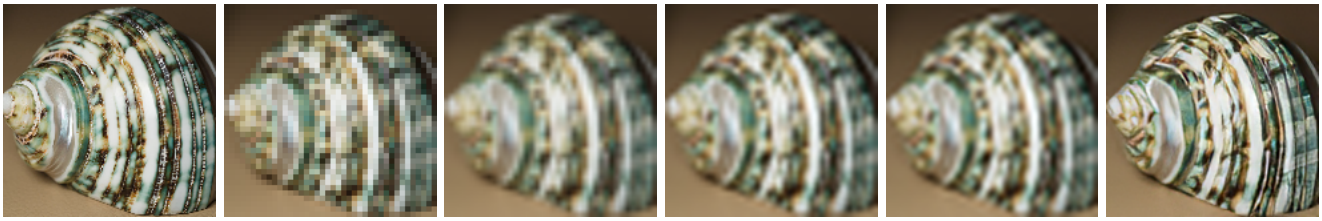
$$y = \frac{(x - x_1)}{(x_2 - x_1)}(y_2 - y_1) + y_1 = \frac{(x - 2)}{(5 - 2)}(8 - 3) + 3 = \frac{5}{3}x - \frac{1}{3}$$

3. Si se quisiera saber qué valor tendría y en el punto 3,5:

$$y = \frac{5}{3} * 3{,}5 - \frac{1}{3} = 5{,}5$$

4. El punto obtenido sería el M_i(3,5 , 5,5)

Las operaciones de interpolación aplicadas en imágenes digitales para cambiar el tamaño que vamos a analizar en los siguientes apartados son: vecino más cercano, bilineal, bicúbica, Lanczos y ESRGAN. La figura 4.1 refleja visualmente los resultados de dichos métodos.

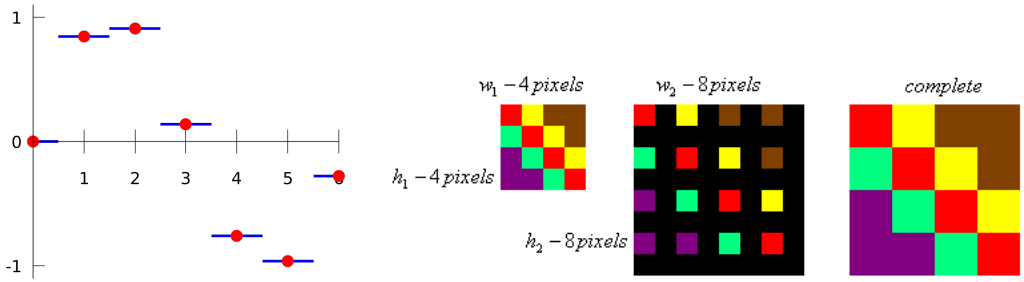


(a) Original **(b)** V. cercano **(c)** Bilineal **(d)** Bicúbica **(e)** Lanczos **(f)** ESRGAN (IA)

Figura 4.1: Comparación de los métodos con una imagen real [26]

4.2. Interpolación del vecino más cercano

La interpolación del vecino más cercano, en inglés *Nearest Neighbour Interpolation*, también es conocido como interpolación proximal. Posee uno de los enfoques más simples para el aumento de tamaño mediante el uso de la interpolación.



(a) Una dimensión [40] **(b)** Dos dimensiones [41]

Figura 4.2: Ejemplo.- Interpolación del vecino más cercano

Los espacios en negro del ejemplo de dos dimensiones (figura 4.2.b) son los píxeles cuyo valor hay que interpolar. En esta técnica, los espacios vacíos (negros) tendrán los mismos valores que el píxel vecino más cercano. En otras palabras, se aumenta el tamaño de cada píxel. Su algoritmo es rápido y su implementación es bastante cómoda aunque se generan unas imágenes de baja calidad, además de producir irregularidades no deseadas en los contornos de la imagen.

4.3. Interpolación bilineal

La interpolación bilineal, en inglés *Bilinear Interpolation*, tiene en cuenta los cuatros píxeles conocidos más cercanos en una vecindad de 2x2 píxeles para calcular el valor del píxel interpolado.

Un ejemplo sería:

- Se parte de cuatro puntos

  $Q_{11} = (x_1, y_1)$

  $Q_{21} = (x_2, y_1)$

  $Q_{12} = (x_1, y_2)$

  $Q_{22} = (x_2, y_2)$

- Se escoge una coordenada, en este caso la y

- Se interpolan linealmente los dos valores que tienen la misma coordenada

  Q_{11} y $Q_{21} \rightarrow R_1 = (x, y_1)$

  Q_{12} y $Q_{22} \rightarrow R_2 = (x, y_2)$

- Con los valores obtenidos, se vuelve a aplicar la interpolación lineal y se obtiene el punto interpolado P

  R_1 y $R_2 \rightarrow P = (x, y)$

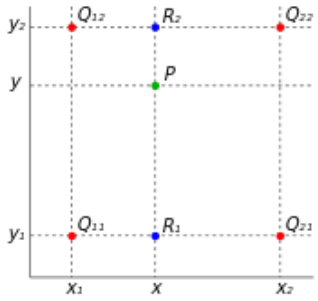


Figura 4.3: Ejemplo.- Interpolación bilineal [43]

Este tipo de interpolación conlleva un uso mayor de recursos y tiempo que la interpolación del vecino más cercano. Aunque los resultados son mejores, se produce un suavizado de los detalles reflejándose sobre todo en la escasa definición de los bordes en los objetos que forman la imagen. Por eso se dice que tiene un defecto "borroso".

4.4. Interpolación bicúbica

La interpolación bicúbica, en inglés *Bicubic Interpolation*, consiste en asignar el valor resultante de aplicar la interpolación entre los 16 píxeles más cercanos distribuidos en una vecindad de 4x4 al píxel que se quiere interpolar, teniendo en cuenta que los píxeles más cercanos al interpolado tienen mayor peso.

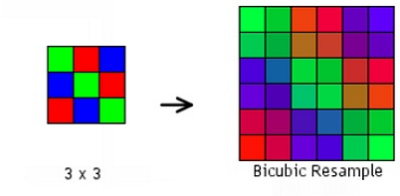
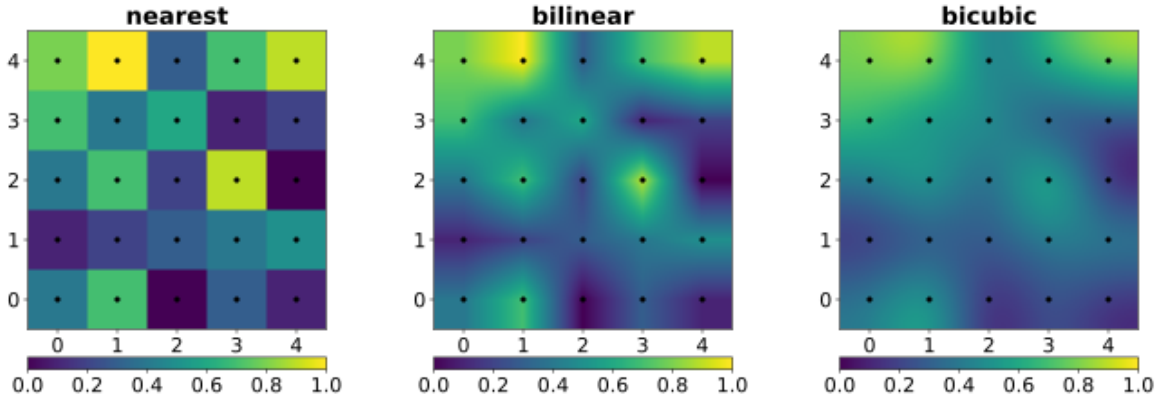


Figura 4.4: Ejemplo.- Interpolación bicúbica

Se utiliza como estándar en programas de edición fotográfica ya que se obtienen mejores resultados, mayor nitidez que con los otros métodos vistos con anterioridad, aun requiriendo más tiempo de procesamiento.



(a) Vecino más cercano **(b)** Bilineal **(c)** Bicúbica

Figura 4.5: Comparación distintas formas de interpolación [44]

4.5. Interpolación de Lanczos

La interpolación de Lanczos, en inglés *Lanczos Interpolation*, fue creada por Cornelius Lanczos. Es una fórmula matemática que se usa para interpolar suavemente el valor de una imagen. Para eso se mapea cada muestra de la imagen original a una copia traducida y escalada con el kernel de Lanczos, que es una función seno cardinal $sinc(x) = \frac{\sin(x)}{x}$.

El kernel de Lanzcos se define como:

$$L(x) = \begin{cases} sinc(x)sinc(x/a) & \text{si } -a < x < a \\ 0 & \text{en otro caso} \end{cases}$$

$L(x)$ es el efecto de cada muestra de entrada en los valores interpolados. El parámetro a es un valor entero positivo que determina el tamaño del kernel, es decir, cuántos píxeles se toman para la interpolación del valor. En el caso de dos dimensiones es $L(x, y) = L(x)L(y)$.

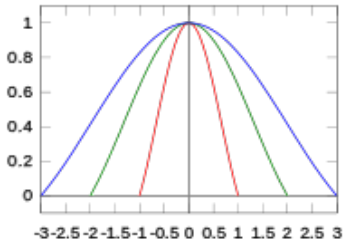


Figura 4.6: Gráfica de la ventana de Lanczos $sinc(x/a)$ [45]

El valor interpolado del píxel x se calcula según la siguiente expresión:

$$S(x) = \sum_{i=[x]-a+1}^{[x]+a} s_i L(x - i)$$

donde a es el tamaño del filtro, $[x]$ el valor truncado de x y s_i una muestra vectorial unidimensional. Es una fórmula matemática para interpolar de forma suave el valor de una imagen digital. En función del orden del kernel se toma un valor u otro de píxeles cercanos.

La imagen interpolada obtenida por este método es mejor que las anteriores porque se conservan los detalles y se generan menos artefactos y/o errores geométricos. Pero requiere altos tiempos computacionales y es inadecuado para su uso comercial.

4.6. Redes generativas antagónicas. ESRGAN

También denominadas *Generative Adversarial Networks* (GAN), es un tipo de red neuronal, pilar fundamental de la Inteligencia Artificial. Se presentaron por primera vez por el investigador Lan Goodfellow el 10 de junio de 2014. Esta investigación se encuentra publicada en la siguiente url: https://arxiv.org/abs/1406.2661. Significó una importante evolución en la inteligencia artificial del método de aprendizaje no supervisado, ya que, el modelo tiene conocimiento a priori de los posibles datos aleatorios y se ajusta a dichas observaciones para su construcción.

Los modelos generativos se fundamentan en que son capaces de aprender de la distribución de probabilidad de un conjunto de entrenamiento a través de varios sistemas:

- Un sistema que puede crear datos a partir de los originales, aumentando el conjunto de datos.
- Un sistema de aprendizaje por refuerzo, donde el generador es un simulador del entorno.

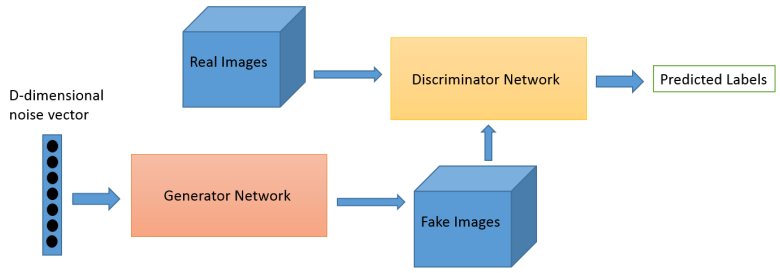


Figura 4.7: Representación de los distintos sistemas [84]

Las redes generativas adversas están compuestas por dos redes neuronales competitivas entre sí, que son:

- **Discriminador** o **Modelo Discriminativo D**: clasifica la imagen como real o falsa y estima la probabilidad de que una muestra provenga desde los datos de entrenamiento.

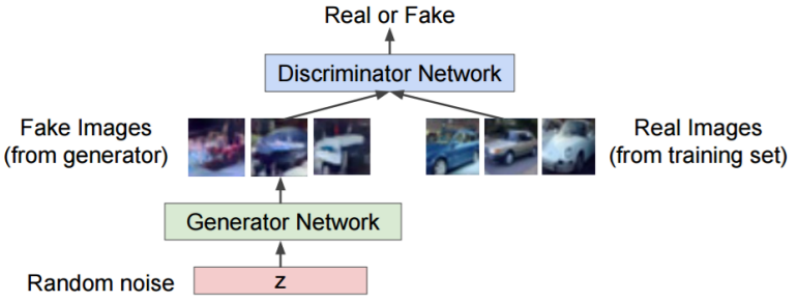


Figura 4.8: Esquema del discriminador [84]

- **Generador** o **Modelo Generativo G**: es el que recibe como entrada un vector aleatorio, hace un captura de la distribución de datos e intenta aplicarla en imágenes basadas en las reales, creando una nuevas imágenes con el objeto de engañar al Discriminador.

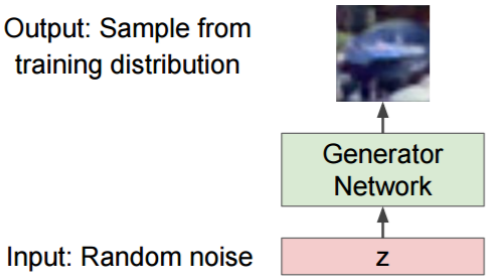


Figura 4.9: Esquema del generador [84]

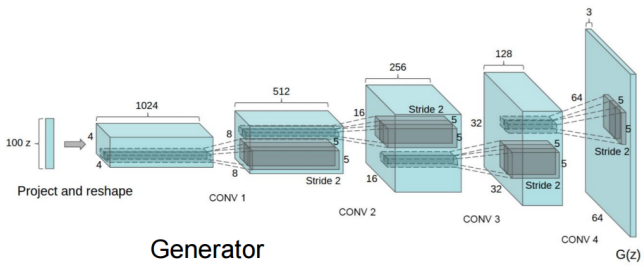


Figura 4.10: Arquitectura del generador [84]

Funcionamiento

Se trata de una red adversativa generativa, esto significa que una red compite con la otra y ambas son entrenadas con un conjunto de datos idénticos. Dichos datos inducen al generador a crear mejores imágenes basadas en los datos originales de entrenamiento, mientras que el discriminador debe discernir si esta imagen es real o generada, alcanzando un equilibro entre ambas.

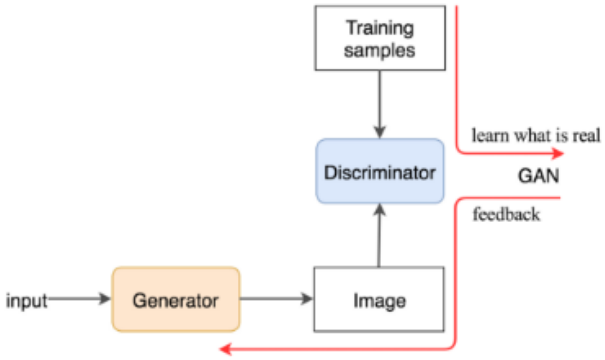


Figura 4.11: Esquema de la interacción del generador y el discriminador

Al emplearse un planteamiento competitivo el generador intenta crear variaciones lo más realistas posible a partir de los datos procesados, mientras que el discriminador debe diferenciar si esas muestras que está procesando son los datos originales de entrenamiento o unas falsas inventadas por el generador intentando simular una imagen real.

Problemas

El manejo de estas redes GAN, al intentar alcanzar un equilibrio entre ambas redes, puede generar una serie de dificultades. Este sistema de competición provoca cierta inestabilidad que se traduce en una gran facilidad para "atascarse". Además de complicar el entreno simultáneo de las distintas redes debido al equilibrio que debe existir en el dominio de ambas, evitando que una domine sobre la otra.

Otro problema sucede cuando el generador explota la debilidad que ha descubierto en el discriminador, produciendo bastantes imágenes similares independientemente del valor de entrada. Este problema se llama **colapso del modelo**. Para solucionarlo basta con robustecer el discriminador, modificando su configuración interna de las capas o ajustando su tasa de entrenamiento. Existen múltiples investigaciones y publicaciones que intentan solventar los problemas de este tipo de redes.

ESRGAN

La *Enhanced Super-Resolution Generative Adversarial Network* (ESRGAN) es una mejora de la *Super-Resolution Generative Adversarial Network* (SRGAN) capaz de generar texturas realistas en una imagen con la super-resolución. Sin embargo, se producían artefactos desagradables que modificaban la calidad visual de la propia imagen generada.

Para minimizar la aparición de dichos "errores" se modifican los componentes fundamentales con la intención de mejorar la calidad perceptiva de la super-resolución. Así se consigue una mejor calidad visual debido a unas texturas más realistas y naturales.

- **Arquitectura de red**

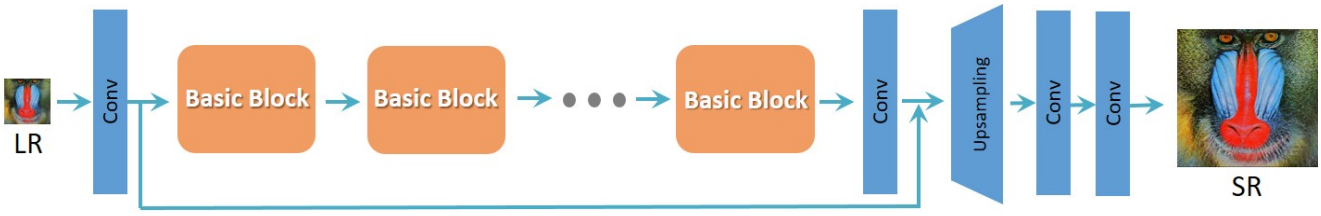
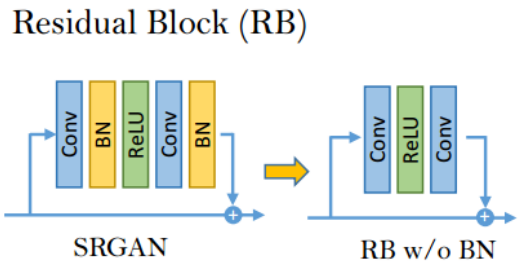


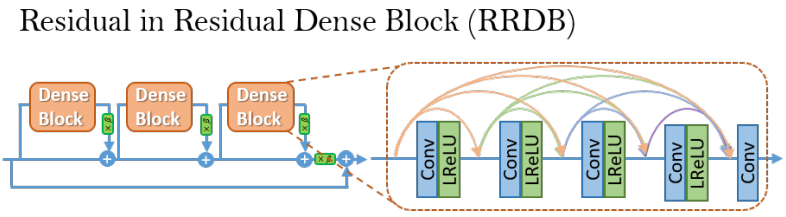
Figura 4.12: Arquitectura de la red [67]

Las modificaciones son las siguientes:

- **Eliminación de las capas *Batch Normalization***. Normalizan las características usando la media y la varianza de un lote durante el entrenamiento y las estimadas de todo el conjunto de datos del entrenamiento. Durante las pruebas introduciendo desagradables artefactos y limitando la capacidad de generación sobre todo, cuando las estadísticas de los datos de prueba y de entrenamiento difieren bastante. La eliminación de estas capas estabilizan el entrenamiento, además de aumentar el rendimiento y la capacidad de generalización, reduciendo la complejidad computacional y el uso de memoria. [74]



- Sustitución del bloque original por uno ***Residual-in-Residual Dense Block***. Este nuevo bloque emplea una estructura más profunda y compleja que el bloque residual original para proporcionar una estructura similar que se beneficia de las conexiones densas, mejorando la capacidad de dicha red. [67]



- **Mejoras del discriminador**: se realiza la modificación del discriminador, basándose en la red generativa adversa relativista, donde se intenta predecir la probabilidad de que una imagen real sea relativamente más realista que una falsa. El original estimaba la probabilidad de que una imagen fuese real y natural.
- **Pérdida de percepción**: se define previamente en las capas de activación de la red pre-entrenada, minimizando la distancia entre dos características activadas. Con ello se pretende ser más efectivo y se restringen las características antes de la activación. De ahí que la red VGG, en la que se basa el reconocimiento de materiales, se centra más en las texturas que en el propio objeto.

Capítulo 5

Experimentación

5.1. Metodología

La metodología seguida en este proyecto se estructura en las siguientes secciones:

1. Replicación del proyecto original.
2. Generación de resultados propios.
3. Comparación con las técnicas clásicas.

La replicación del proyecto original ha consistido en reproducir todos los pasos seguidos por los propios autores (Xintao Wang y Ke Yu entre otros) para la obtención de resultados, es decir, imitarlos. Para lograrlo, primero se entrenó la red neuronal ESRGAN con los mismos *dataset* que usaron. Un *dataset* son imágenes preparadas. Después se probó el entrenamiento en la propia red y se obtuvieron unos resultados que serán comparados con los del proyecto original. Comprobando si estos no distan tanto de los resultados obtenidos.

Una vez replicado el proyecto original, se procedió a la obtención de resultados propios, que se entrenaron, probaron la red neuronal y se compararon con los resultados de la utilización de un *dataset* propio. Con las imágenes resultantes se procede a realizar un análisis comparativo con los resultados de aplicar las técnicas clásicas en el *dataset* creado.

5.1.1. *Dataset*

Un *Dataset* es un conjunto de imágenes preparadas que se utilizan para el entrenamiento de la red neuronal y el test de la misma. Como mínimo está formado por 2 carpetas que son:

- **Carpeta HR**: contienen las imágenes de alta calidad denominadas "*Ground Truth*". Cuya funcionalidad dependerá de la fase en la que se esté utilizando:
  - Fase de entrenamiento: se toma la imagen como referencia de la salida que se espera.
  - Fase de test: se compara con los resultados obtenidos al aplicar las distintas técnicas de escalado.
- **LR4**: incluye la versión redimensionada a la cuarta parte del tamaño de las imágenes de la carpeta HR. Su tamaño es el cociente del tamaño original entre 4.

Aunque existe la posibilidad de que en ese *dataset* existan más carpetas. En algunos contienen las carpetas LR2, LR3 y LR8 en las que incluyen imágenes redimensionadas con un factor 2, 3 y 8 respectivamente.

Los *datasets* que se usan con más frecuencia en el tema de la Superresolución son los siguientes:

- **Set5**: se utiliza para probar los modelos y consta de 5 imágenes.
- **Set14**: es otro *dataset* que se utiliza para probar los modelos, concretamente en la fase de entrenamiento. Como su propio nombre indica está formado por 14 imágenes.

- **DIV2K**: se usa para entrenar los modelos, consta de un conjunto de 900 imágenes RGB con una resolución de 2K, de las cuales, 800 se usan para el entrenamiento del modelo y las otras 100 para la validación. Todas estas imágenes son de alta definición por lo que es *dataset* en total ocupa 4.0 GB. Estas imágenes son de temática diversa desde paisajes naturales hasta personas. Este *dataset* se utilizó en NTIRE 2017 *Challenge on Single Image Super-Resolution* (Desafío de superresolución en una sola imagen). [82] [83]

5.1.2. Métricas

Para poder comparar las distintas imágenes obtenidas, como resultado de la aplicación de las diferentes técnicas, es necesario encontrar métricas adecuadas. Teniendo en cuenta que en ocasiones las imágenes resultantes se asemejan y no se aprecian las diferencias a simple vista.Las métricas elegidas son:

- Error cuadrático medio.
- Proporción máxima de señal a ruido.
- Índice de similitud estructural.
- Índice de calidad basado en la variación local.

Error cuadrático medio

El *Mean Squared Error* o Error Cuadrático Medio (**MSE**) es un estimador que mide el promedio de los errores al cuadrado de la diferencia existente entre los pixeles de la imagen original y los de la imagen interpolada.

La fórmula general es:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2$$

aunque adaptada al caso de imágenes es la siguiente:

$$MSE = \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m}(Y_{ij} - \widehat{Y}_{ij})^2$$

donde m y n son las dimensiones de la imagen, Y_{ij} es la señal de la imagen original mientras la señal de la imagen interpolada es $\widehat{Y}_{ij}$.

Es una métrica indirecta porque cuanto menor sea el valor, mejor es el resultado, cuanto menor sea el error más se asemejarán las imágenes. Además al tratarse de una medición sobre errores es recomendable que su valor sea lo menor posible.

Proporción máxima de señal de ruido

La *Peak Signal-to-Noise Ratio* o Proporción Máxima de Señal de Ruido (**PSNR**) define mediante una escala logarítmica la relación existente entre la energía máxima posible de una señal y el ruido que afecta a su representación fiable. Mide cuánto se desvía la imagen interpolada (posiblemente de menor calidad) de la imágen original de alta calidad. Su unidad es el decibelio dB.

Para calcular el PSNR se usa la formula siguiente:

$$PSNR = 10log_{10}(\frac{MAX_I^2}{MSE}) = 20log_{10}(\frac{MAX_I}{\sqrt{MSE}})$$

donde MSE es el error cuadrático medio y MAX_I es el valor máximo del píxel en la imagen original, también es el resultado de $2^B - 1$ donde B es el número de bits que se utilizan para codificar el pixel.

Al ser una métrica directa, cuanto más alto sea el PSNR, mejor será la calidad de la imagen reconstruida porque el valor que toma el error (MSE) es bajo. Los valores típicos están comprendidos entre 30 y 50 dB.

Índice de similitud estructural

El *Structural Similarity index* o Índice de Similitud Estructural (**SSIM**) evalúa el grado de similitud entre dos imágenes fijándose en las posibles degradaciones provocadas por la modificación de algunas características de la imagen como son:

- La luminancia o brillo (l): luz reflejada por los componentes dentro de la imagen.
- El contraste (c): diferencia relativa en la intensidad entre dos puntos en una imagen.
- La estructura (s).

Para calcular el SSIM se usa la siguiente forma:

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

Donde:

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

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

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

$$C_1 = (K_1L)^2, C_2 = (K_2L)^2, C_3 = C_2/2$$

donde L es el rango dinámico del valor del píxel y las constantes escalares K_1 y $K_2 << 1$.

Es una métrica directa, cuanto mayor sea el valor, más se parecen las imágenes entre sí. Los valores que toma la métrica están comprendidos en el intervalo 0 y 1. El valor 0 indica que las imágenes son completamente diferentes mientras que si son completamente iguales el valor es 1.

Índice de calidad basado en la variación local

El *Quality Index based on Local Variance* o Índice de calidad basado en la variación local (QILV) se basa en la suposición que gran cantidad de la información estructural de la imagen se codifica en la distribución de su varianza local.

La fórmula es:

$$QILV(I,J) = \frac{2\mu_{V_I}\mu_{V_J}}{\mu_{V_I}^2 + \mu_{V_J}^2} * \frac{2\sigma_{V_I}\sigma_{V_J}}{\sigma_{V_I}^2 + \sigma_{V_J}^2} * \frac{\sigma_{V_IV_J}}{\sigma_{V_I}\sigma_{V_J}}$$

donde I, J son las imágenes a comparar.

El primer término realiza una comparación entre la media de las variaciones locales de las imágenes. El segundo término compara las desviaciones estándar de las variaciones locales. El tercer término introduce información cruzada de las imágenes.

Los valores de está métrica están comprendidos entre el 0 y el 1. Cuanto menor sea el valor del índice, mayor será el desenfoque en la imagen interpolada.

Implementación de las métricas

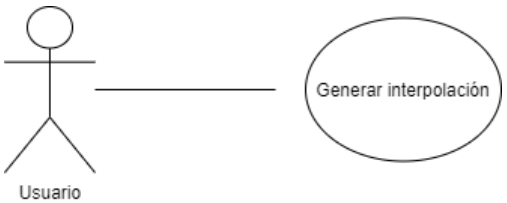


Figura 5.1: Diagrama de casos de uso

CdU-01	Calcular métricas	
Versión	1	
Actor	Usuario	
Descripción	El sistema deberá comportarse tal y como se describe en este caso de uso cuando el usuario desee calcular la métrica en una imagen.	
Precondición	Las imágenes que se pasan como parámetros deberán tener las mismas dimensiones.	
Secuencia normal	**Paso**	**Acción**
	1	El Usuario ejecuta el programa
	2	El sistema genera las métricas correspondientes
Postcondición	Ninguna	
Prioridad	Alta	
Estado	Desarrollado	

Tabla 5.1: Descripción del caso de uso 02

El funcionamiento del programa durante la ejecución es el siguiente: se calcula las métricas entre dos imágenes que se han pasado como parámetros, siendo la primera imagen la original mientras que la segunda será la resultante de la interpolación. El resultado se escribirá por pantalla.

En vista de obtener la funcionalidad deseada del programa se buscan herramientas software disponibles para su elaboración. Se localiza una librería de código abierto llamada *Open Computer Vision Library* o **OpenCV** que permite leer imágenes y calcular valores. También se encuentra una función que permite calcular la métrica del SSIM en el repositorio PyPI [64].

5.2. Replicación

El proyecto de la red ESRGAN con el que se ha experimentado está depositado en un directorio Github que tiene la siguiente dirección https://github.com/xinntao/ESRGAN. Desarrollado por Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu... quedaron primeros en la PIRM2018-SR. La PIRM2018-SR [65] fue una competición organizada por *European Conference on Computer Vision* el día 14 de Septiembre del 2018 en Munich, Alemania [66] como reto del "Taller y desafío en la restauración de la imagen perceptiva y manipulación".

5.2.1. Entrenamiento

El primer paso para la replicación fue mirar cómo se entrena la red, buscar que ficheros hay que modificar y los *datasets* utilizados. Se encontró la siguiente información:

1. Preparar el *dataset*
2. Preparar el modelo preentrenado PSNR
3. Modificar el fichero option/train/train_ESRGAN.yml
4. Ejecutar el siguiente comando:

```
$ python train.py -opt options/train/train_ESRGAN.yml
```

Una vez encontrados los pasos a realizar, se empezó con el primero. Se comenzó con la replica del experimento, se necesitó saber qué *datasets* habían utilizado los autores para entrenar la red. Dicha información se localiza en el fichero modificado en el paso 3. Se observa que utilizan el *dataset* DIV2K y el Set14. Los enlaces para su descarga se encuentran documentados en la información del proyecto (https://drive.google.com/drive/folders/1pRmhEmmY-tPF7uH8DuVthfHoApZWJ1QU. Descargados los ficheros correspondientes se empezaron a generar los *datasets*.

Antes de generar el *dataset* DIV2K como las imágenes de este *dataset* son muy grandes ya que sus dimensiones rondan entorno a los 2040 x 1356 píxeles se tuvieron que tratar antes de procesarlas. Para eso se dividió la imagen

en subimágenes de menor tamaño que el programa más adelante va a poder procesar. Estas imágenes se obtienen de ejecutar el fichero extract_subimgs_single.py que previamente se han modificado las rutas donde se encuentran y donde se va a guardar las imágenes.

```
$ python codes/scripts/extract_subimgs_single.py
```

El resultado obtenido son unas imágenes de tamaño 480 x 480 píxeles. Es decir, la imagen original se divide como si se tratase de un puzle de piezas cuadradas con las dimensiones anteriormente mencionadas (480 x 480 píxeles). El nombre de las subimágenes se corresponde con el siguiente formato:

```
''Nombre original'' + _s + indice + .png
```

donde el índice indica el número de subimagen respecto a la original. Como no se disponen de las imágenes reducidas de tamaño (las imágenes LR) hay que ejecutar el siguiente programa:

```
$ python codes/scripts/generate_mod_LR_bic.py
```

que previamente se ha modificado las rutas de modificar de *sourcedir*, *saverdir*. Finalizada la ejecución, se obtienen varias carpetas: HR, Bic, LR. En la carpeta HR se encuentran las imágenes de tamaño original mientras que en la capeta LR se encuentran las de tamaño reducido. Es muy importante que el número de imágenes contenidas en estas carpetas sea el mismo. Debido a la cantidad de imágenes con las que se van a formar el *dataset*, se necesita aplicar una técnica que favorezca la lectura rápida de las imágenes que es *Lightning Memory-Mapped Database* (LMDB). Al ejecutar el siguiente fichero (codes/scripts/create_lmdb.py) de forma independiente a las dos carpetas de imágenes (HR, LR) se obtiene una carpeta .lmdb, previamente se tienen que modificar los directorios de img_folder y lmdb_save_path.

Lightning Memory-Mapped Database es una biblioteca software de administración de base de datos basada en B-árboles (árbol-B) proporcionando un alto rendimiento. La base de datos no relacional es del tipo clave-valor. Es compatible con el multiprocesamiento porque permite el acceso de lectura / escritura proporcionando un uso simultáneo de los diferentes subprocesos. Debido a la política usada en las páginas de datos que es copia en escritura para no sobrescribir páginas de datos activas y escribiendo las modificaciones de forma serializada.

La generación del *dataset* Set14 es más sencillo que el *dataset* DIV2K. Se modifican las rutas donde se encuentran las imágenes del Set14 y se ejecuta el fichero generate_mod_LR_bic.py.

Con el paso de la preparación del modelo preentrenado orientado al PSNR se usa el modelo facilitado por los propios desarrolladores del proyecto, el fichero RRDB_PSNR_x4.pth.

Una vez generados todos los *dataset* requeridos se modifica el fichero de configuración añadiendo las direcciones correctas de la ubicación de los distintos datasets y el modelo preentrenado. Después de modificar el fichero, se ejecuta el siguiente comando dando comienzo al entrenamiento de ESRGAN.

```
$ python train.py -opt options/train/train_ESRGAN.yml
```

Durante el entrenamiento se iban imprimiendo mensajes por el terminal cada 32 minutos hasta el momento en el que se llegaban a las 4100 iteraciones de la primera época. Una época es un periodo de tiempo en el cual la red evoluciona a partir de una secuencia inicial de ejemplo hasta la respuesta adecuada. Justo después de la finalización de la época, los diferentes valores de los pesos se modifican y se reinicia la red para que el entrenamiento de la siguiente época no dependa del final de esta.

Después de ese momento no se recibía nada por pantalla. El terminal no se actualizaba con ningún nuevo mensaje incluso después de haber pasado más de doce horas. Como se mencionó anteriormente que la red generativa y la red discriminadora se pueden atascar fácilmente durante las iteraciones. Se decidió detener la ejecución y volver a ejecutar para comprobar si realmente se debía a un "atasque" en dicha iteración. Pero resultó que sucedía lo mismo. No pasaba de la iteración número 4100 además, tampoco generaba ningún modelo salvo los 2 ficheros de log. En los cuales el único que solo contenía información era el log de entrenamiento.

Ante esta situación de aparente bloqueo, se decide hacer una traza en el código del fichero ejecutado train.py para averiguar dónde está el posible fallo y poder solucionarlo. Se fueron poniendo mensajes a lo largo del fichero además de modificar el número de iteraciones en el fichero de configuración (options/train/train_ESRGAN.yml) y las imágenes del **dataset** de entrenamiento y de prueba. Al final la ejecución se descubre donde está el fallo y el motivo por el cual no se guardaban la información en los diferentes ficheros del logs y tampoco se creaba los modelos de entrenamiento de la red. Se debía a que no se realizaba el segundo bucle *for* de la zona de entrenamiento.

Se soluciona con la modificación de un parámetro de esa función teniendo en cuenta la documentación y el código de una versión anterior. Se hacen varias pruebas diferenciándose en el número de imágenes que constituyen los distintos *datasets* y el número de iteraciones. La primera prueba está formada por el primer fragmento de la primera imagen del *dataset* DIV2K y una imagen del *dataset* Set5 para el entrenamiento y la validación y una sola iteración. Finalizada la ejecución, se obtiene:

- **Log del entrenamiento**: se detalla la configuración, los parámetros y las distintas capas de la red neuronal. Se registra el PSNR de ese momento además de indicar el momento en los que se está guardando los ficheros del modelo y los estados del entrenamiento.
- **Log de la validación**: se registra el valor del PSNR indicando la época y la iteración en la cual se calcula.
- **Carpeta *models***: se guardan dos modelos. Uno para la red generativa (G) y el otro para la red discriminadora (D).
- **Carpeta *training_state***: se guardan copias del estado en el que se encuentra la red en fichero .state.
- **Carpeta *val_images***: se guarda el resultado de la(s) imagen(es) de validación en función del estado de la red neuronal.

Comprobado varias veces su correcto funcionamiento, se decide aumentar el número de imágenes de los *dataset*, el número de iteraciones, la frecuencia de generación de los modelos y las copias de seguridad del estado de la red neuronal.

El *dataset* de entrenamiento está formado por la primera imagen del *dataset* DIV2K (40 subimágenes) y el *dataset* de contraste es el Set5. El número de iteraciones son 10000. Después de unos días ejecutándose, se interrumpe la ejecución porque no hay espacio suficiente en el disco del portátil debido a la frecuencia tan alta que se guardaban los resultados que se había configurado a 10. Gracias a los estados intermedios guardados como copia de seguridad en el transcurso del entrenamiento se puede continuar desde ese punto y no hace falta volver a ejecutar el entrenamiento desde el principio.

Para lograrlo se modificó el fichero de configuración options/train/train_ESRGAN.yml y el campo denominado resumen_state con la ruta donde se encuentra el fichero con extensión .state que se usará como punto de inicio. Justo después de configurarlo, se modifica el programa train.py porque el entrenamiento era distribuido y el ordenador en el cual se va a realizar el entrenamiento no posee tarjeta NVIDIA debido a su antigüedad, por lo que no es posible la utilización de CUDA.

```
# Original
resume_state = torch.load(opt['path']['resume_state'],
                          map_location=lambda storage, loc: storage.cuda(device_id))
# Modificacion
resume_state = torch.load(opt['path']['resume_state'])
```

En el fichero original se cargan los modelos guardados y se almacenan en los distintos dispositivos CUDA. La modificación consiste en no cargar el modelo en los distintos dispositivos que formarían parte en el entrenamiento distribuido. Se logra con la eliminación de los parámetros de almacenamiento y localización.

Después de la modificación del fichero se procede a la ejecución del programa de entrenamiento original en el que se utilizan el *dataset* DIV2K y el *dataset* Set14 como punto de partida y validación respectivamente. Con el paso de días llega un momento en donde los logs no se actualizan ni se generan más modelos por lo que se vuelve a ejecutar para observar si el entrenamiento termina en algún momento porque en las pruebas ejecutadas anteriormente con pocas imágenes sí que finalizaba. Repetido en varias ocasiones los intentos y al suceder lo mismo, se decide usar el último modelo que se ha guardado aunque no sea el modelo definitivo del entrenamiento.

5.2.2. Test

Para la realización del test o prueba del modelo generado en el entrenamiento de la red se siguen los siguientes pasos encontrados en la documentación del proyecto:

1. Modificar el fichero option/test/test_ESRGAN.yml
2. Ejecutar el siguiente comando:

```
$ python test.py -opt options/test/test_ESRGAN.yml
```

Las modificaciones que se deben realizar en el fichero option/test/test_ESRGAN.yml son en los siguientes campos: *petrain_model_G* se debe poner la ruta donde se ubica el modelo generativo (archivo_G) creado durante el entreno de la red neuronal, en los campos *dataroot_GT* y *dataroot_LQ* se escribirá la ubicación donde se encuentran las imágenes con alta definición (HR) y las reducidas (LR) respectivamente. Los *dataset* utilizados en esta parte son los del Set5 y el Set14. Solamente hay que generar el Set5 puesto que el Set14 ya ha sido utilizado anteriormente en la parte del entrenamiento.

Para generar el Set5 hay que repetir los pasos vistos anteriormente en la sección del entrenamiento:

1. Buscar las imágenes.
2. Reducir el tamaño.
3. (Opcional) Convertir al formato .lmdb el resultado.

Se buscan las imágenes que forman el *dataset* Set5 pero como no se disponer de las imágenes reducidas, se procede a su reducción. Se modifican las diferentes rutas de los campos *sourcedir* y *saverdir* en función de los datos actuales y se ejecuta el programa generate_mod_LR_bic.py.

```
$ python codes/scripts/generate_mod_LR_bic.py
```

De forma opcional se puede reducir el tiempo de ejecución de la prueba (test) del modelo generado con la conversión al formato lmdb de las imagenes del *dataset*. Previamente a la ejecución se ha mdificado las rutas de *img_folder* y *lmdb_save_path*.

```
$ python codes/scripts/create_lmdb.py
```

Inmediatamente después de haber configurado el fichero de configuración (las ubicaciones de los *dataset* y el modelo de la red neuronal) se procede a la comprobación del modelo obtenido en el entrenamiento con la ejecución del siguiente comando.

```
$ python test.py -opt options/test/test_ESRGAN.yml
```

Los resultados de la ejecución son los cálculos de la métrica PSNR y SSIM de cada imagen que compone el *dataset* además de cuantificar las métricas del *dataset* mediante un promedio. También se calcula con el canal Y (Channel Y) que representa el brillo de la imagen, esto es, la parte acromática de la imagen. Las figuras 5.2 y 5.3 muestran los resultados obtenidos de la ejecución del comando anterior en los *datasets* Set5 y Set14.

```
20-03-05 12:52:44.605 - INFO:
Testing [set5]...
20-03-05 12:54:07.603 - INFO: baby_ESRGAN          - PSNR: 31.037373 dB; SSIM: 0.840471; PSNR_Y: 32.715592 dB; SSIM_Y:
0.874291.
20-03-05 12:54:36.649 - INFO: bird_ESRGAN          - PSNR: 31.551234 dB; SSIM: 0.881768; PSNR_Y: 34.442114 dB; SSIM_Y:
0.938578.
20-03-05 12:55:00.296 - INFO: butterfly_ESRGAN     - PSNR: 26.735045 dB; SSIM: 0.887240; PSNR_Y: 28.403570 dB; SSIM_Y:
0.913681.
20-03-05 12:55:28.648 - INFO: head_ESRGAN          - PSNR: 30.154689 dB; SSIM: 0.734923; PSNR_Y: 31.940304 dB; SSIM_Y:
0.780457.
20-03-05 12:55:58.000 - INFO: woman_ESRGAN         - PSNR: 29.702688 dB; SSIM: 0.907566; PSNR_Y: 31.452941 dB; SSIM_Y:
0.928043.
20-03-05 12:55:58.017 - INFO: ----Average PSNR/SSIM results for set5----
        PSNR: 29.836206 dB; SSIM: 0.850394

20-03-05 12:55:58.018 - INFO: ----Y channel, average PSNR/SSIM----
        PSNR_Y: 31.790904 dB; SSIM_Y: 0.887010
```

Figura 5.2: Resultado de *test.py* en el set5

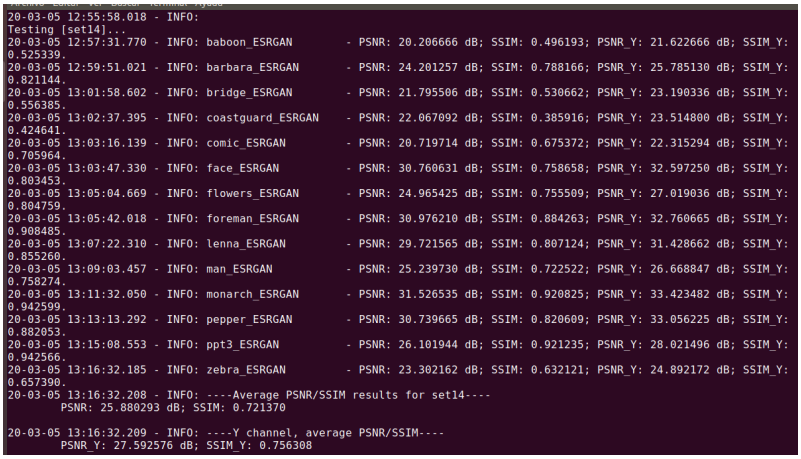


Figura 5.3: Resultado de *test.py* en el set14

5.2.3. Resultados de la replicación

Obtenido el modelo replicado después del entrenamiento de la red neuronal ESRGAN se dispone a compararlo con el modelo original de Xintao. Para lograrlo se van a observar las métricas en los dos momentos siguientes:

- En la ejecución del test al modelo.
- En la ejecución del modelo con el ESRGAN.

Se han elegido esos momentos porque el primero se realiza una vez finalizado obtenido el modelo en el entrenamiento de la propia red obteniendo conocimientos que permiten averiguar si la red se puede mejorar más o no. Y el segundo es una aplicación de los distintos modelos en el ESRGAN. Los valores de las métricas obtenidas se han recogido en las tablas contiguas:

PSNR	**ESRGAN (Xintao)**	**ESRGAN (Replicado)**
Set5	34,447550	29,836206
Set14	28,095125	25,880293
Baboon	29,923	29,273
Comic	30,459	29,665

Tabla 5.2: Comparativa I (en decibelios)

SSIM	**ESRGAN (Xintao)**	**ESRGAN (Replicado)**
Set5	0,940293	0,850394
Set14	0,818644	0,721370
Baboon	0,695	0,524
Comic	0,820	0,716

Tabla 5.3: Comparativa II

Observando los diferentes valores obtenidos del modelo del ESRGAN replicado y del modelo de Xintao no difieren excesivamente. Mantienen valores similares en la métrica PSNR (Proporción máxima de señal a ruido) lo que significa que la imagen generadas por los dos modelos no se desvían tanto de la imagen original aunque en el caso de la métrica SSIM (Índice de similitud estructural) los valores distan un poco más por lo que el propio modelo genera imágenes menos similares a la original que el modelo de Xintao.

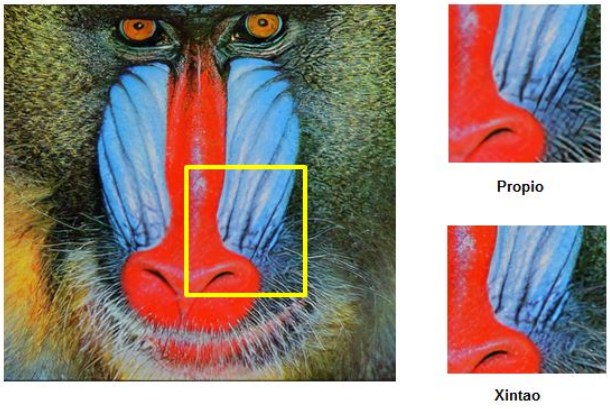


Figura 5.4: Comparación resultados en baboon.png

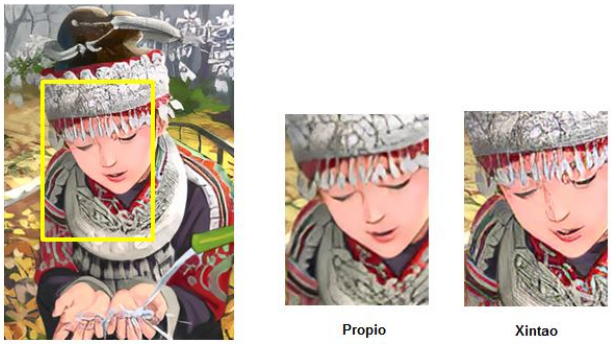


Figura 5.5: Comparación métodos en comic.png

Aunque se obtengan imágenes con pequeñas diferencias visualmente hablando se concluye que, a pesar de la escasa diferencia de la calidad, con el modelo replicado se pueden obtener resultados parecidos a los generados por el modelo de Xintao.

5.3. Generación de resultados propios

Se procede a la obtención de resultados propios generados por la red neuronal entrenada con *datasets* construido con las imágenes captadas por diversos dispositivos fotográficos:

- Cámara de fotos Olympus con 8 megapíxeles.
- Cámara de fotos Panasonic con 14 megapíxeles.
- Móvil BQ Aquaris M5 con 13 megapíxeles.
- Móvil Samsung Galaxy A6 con 16 megapíxeles.

En temas relacionados con la superresolución se generan *datasets* compuestos por numerosas imágenes. Estas imágenes son utilizadas a lo largo de un proceso dividido en las etapas: entrenamiento, validación y test. La composición de dicho *dataset* sigue estas proporciones: un 70 % para el entrenamiento, un 20 % para la valoración y el 10 % restante para el test. Con cifras concretas, si el *dataset* está constituido por 24000 imágenes, 17000 serían las de entrenamiento, 5000 para validación y 2000 para test.

Al ser un número de imágenes bastante elevado se reduce a 100 imágenes propias aunque en el entrenamiento se mantiene el *dataset* DIV2K, utilizado anteriormente en el entrenamiento de la replicación del ESRGAN siguiendo lo realizado por Xintao y sus colaboradores. Sabiendo los porcentajes de composición del *dataset* mencionados anteriormente, se concluye que el *dataset* tiene la estructura siguiente:

- 70 imágenes para el entrenamiento más las que forman el *dataset* DIV2K
- 20 imágenes de validación

- 10 imágenes de test. La mitad serán propias y la otra mitad de los Set5 y Set14.

5.3.1. Creación de los *datasets*

Teniendo claro cuantas imágenes hay en cada *dataset* y la función que se le asigna se deciden hacer 4 *datasets*:

- **Entrenamiento**
- **Validación**
- **TestSet1**: formado por 3 imágenes propias y 2 de los *dataset* set5 y set14.
- **TestSet2**: consta de 5 imágenes de las que 2 son imágenes propias y 3 de los *dataset* set5 y set14.

Se sale por la ciudad realizando diferentes fotografías a los objetos, estructuras, naturaleza, vehículos... que se encuentran en la vía pública como pueden ser unos columpios en un parque infantil, una motocicleta, escaparate, las ramas de unos árboles, el paisaje de las vías del tren... Después de realizar las fotos se criban seleccionando las más curiosas y evitando imágenes muy similares entre sí.

Con las imágenes seleccionadas se clasifican según la función que van a desempeñar entre los distintos *dataset* y se guardan en la correspondiente carpeta. Como las imágenes se guardan en formato .jpeg y no las reconoce los diferentes scripts se cambian de formato (.png) con el siguiente comando:

```
$ mogrify -format png *.*
```

Dataset Entrenamiento

Se ejecuta el programa *extract_subimags_single.py* para subdividir las imágenes en subimágenes más pequeñas. Adaptándolas a un tamaño que el programa pueda procesar debido a las dimensiones de las fotografías originales.

```
$ python extract_subimgs_single.py
```

Con dicha transformación pasamos de 170 imágenes (de las cuales 70 son propias y 100 del *dataset* DIV2K) a 12334 subimágenes (8190 de las imágenes propias y 4144 del DIV2K). Para generar las imágenes reducidas a la cuarta parte que no se disponen de ellas se ejecuta *generate_mod_LR_bic.py*.

```
$ python generate_mod_LR_bic.py
```

Las carpetas resultantes HR y LR por la aplicación de lo anterior se transforman al formato .lmdb utilizando *create_lmdb.py*:

```
$ pyhton create_lmdb.py
```

Los distintos programas se han modificado ajustando las rutas donde se ubican las diferentes carpetas utilizadas.

Dataset Validación

El procedimiento de validación o test seguido es similar al visto en el apartado anterior "*Dataset* Entrenamiento". Las dimensiones de las imágenes que forman este *dataset* en el caso de Xintao son las mismas que se obtienen en la ejecución de *extract_subimags_single.py*. Se ejecuta dicho programa para subdividir las imágenes en subimágenes más pequeñas adaptando sus dimensiones a unas que el propio programa puede manejar.

```
$ python extract_subimgs_single.py
```

Con dicha transformación pasamos de 20 imágenes a 2340 subimágenes. De todas ellas se deja la subimagen con más variedad cromática y de formas de cada imagen además de tener en cuenta su calidad. Para la generación de las imágenes reducidas a la cuarta parte se ejecuta *generate_mod_LR_bic.py*.

```
$ python generate_mod_LR_bic.py
```

Las carpetas resultantes HR y LR por la aplicación de lo anterior se transforman al formato .lmdb utilizando:

```
$ pyhton create_lmdb.py
```

Dichos programas habrán sido modificados con la ubicación correcta de los directorios donde se ubican las imágenes.

***Dataset* Test**

Se generan las imágenes reducidas mediante la ejecución del programa *extract_subimags_single.py*. Y luego se convierten las carpetas al formato .lmdb. Previamente se habrán adaptado los distintos programas a las carpetas correspondientes.

```
$ python generate_mod_LR_bic.py
$ pyhton create_lmdb.py
```

5.3.2. Entrenamiento

Seguimos los mismos pasos vistos durante la replicación del proyecto que son:

1. Preparar el *dataset*
2. Preparar el modelo preentrenado PSNR
3. Modificar el fichero option/train/train_ESRGAN.yml
4. Ejecutar el siguiente comando:

```
$ python train.py -opt options/train/train_ESRGAN.yml
```

El paso uno y dos ya están realizados. Mi propio *dataset* se construyó en el apartado anterior y el modelo preentrenado PSNR es el mismo que se utilizó para obtener los mismos resultados de Xintao. Se modifica el fichero *option/train/train_ESRGAN.yml* indicando la ubicación de las imágenes que forman el *dataset* (imágenes de alta definición y reducidas). Y se ejecuta:

```
$ python train.py -opt options/train/train_ESRGAN.yml
```

Finalizada la ejecución se obtiene el modelo entrenado de la red neuronal que se utiliza en los próximos apartados.

5.3.3. Test

Obtenido el modelo de la red neuronal a través de la ejecución del entrenamiento de la red ESRGAN con el *dataset* propio de entrenamiento, se comprueba los resultados del entrenamiento testeandolo con los dos *datasets* propios fabricados. Se siguen los pasos vistos previamente:

1. Modificar el fichero option/test/test_ESRGAN.yml
2. Ejecutar el siguiente comando:

```
$ python test.py -opt options/test/test_ESRGAN.yml
```

Anteriormente a la ejecución se ha modificado el fichero con las rutas donde se encuentra los *datasets* y el modelo generado por la ejecución del entrenamiento. Se obtiene lo siguiente:

	PSNR	SSIM
TestSet1	27,784398	0,798979
TestSet2	30,872281	0,851519

Tabla 5.4: Test del propio entrenamiento

5.4. Comparación con las técnicas clásicas

A continuación de la obtención del modelo generado en el entrenamiento con imágenes propias. Se realizó un análisis comparativo de los resultados obtenidos de la ejecución de las técnicas clásicas y con la red neuronal generativa ESRGAN.

5.4.1. Implementación de las técnicas

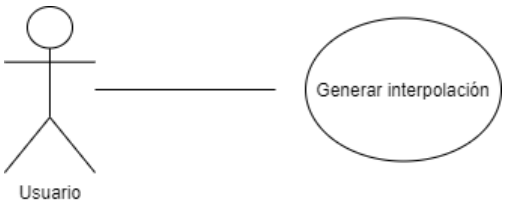


Figura 5.6: Diagrama de casos de uso

CdU-02	**Generar interpolación**	
Versión	1	
Actor	Usuario	
Descripción	El sistema deberá comportarse tal y como se describe en este caso de uso cuando el usuario desee generar la interpolación de las imágenes.	
Precondición	El usuario ha guardado las imágenes en la carpeta correspondiente.	
Secuencia normal	**Paso**	**Acción**
	1	El actor Usuario ejecuta el programa
	2	El sistema genera la interpolación de las imágenes
Postcondición	El usuario obtendrá las imágenes generadas por el sistema en una carpeta.	
Prioridad	Alta	
Estado	Desarrollado	

Tabla 5.5: Descripción del caso de uso 02

En vista de obtener la funcionalidad deseada del programa, se buscan herramientas software disponibles para la elaboración de dicho programa. Se localiza una librería de código abierto que permite interactuar con imágenes ya sea leyendo, escribiendo y/o modificando estás además, dispone de una función que dependiendo del parámetro modifica el método de interpolación de uno a otro. Esta maravillosa librería se denomina *Open Computer Vision Library* o **OpenCV**.

En el apartado "Lectura y escritura de archivos de imagen" (*Image file reading and writing*) de la documentación oficial se hallan las funciones:

- *imread*: carga un archivo de imagen.
- *imwrite*: guarda una imagen en el archivo que se especifica pudiendo escogerse el formato con la modificación de su extensión.

Mientras que en el apartado "Transformación geométrica de la imagen" (*Geometric Image Transformations*) de la documentación se ubica la función más importante del programa, la que modificará la imagen aplicando diferentes interpolaciones. Dicha función es *resize*, la cual permite modificar las dimensiones de la imagen mediante la aplicación de los métodos de interpolación determinados por las banderas implementadas. Esta función se ajusta al trabajo porque dispone de todas las interpolaciones anteriormente vistas (vecino más cercano, bilineal, bicúbica, lanczos).

Después de buscar y leer la documentación de todas las funciones a aplicar, se empieza a codificar el programa en lenguaje *python*. El funcionamiento del programa durante la ejecución es el siguiente: primero se crea la carpeta

donde se guardarán los resultados en el caso de que no exista. A continuación se procederá a la lectura de las imágenes de las cuales se calcula la resultante de su interpretación y se guarda en la carpeta.

5.4.2. Imágenes aleatorias

Las imágenes utilizadas en el análisis fueron seleccionadas al azar de diversas páginas de Internet para conseguir obtener resultados independientes sin estar sesgados. Puesto que al utilizar las mismas imágenes del proceso de entrenamiento de la propia red neuronal pueden modificar los resultados, ya que la red neuronal está entrenada para generar dichas imágenes, provocando unas conclusiones posiblemente erróneas.

Las imágenes seleccionadas han sido recogidas de sitios web libres de derechos recopilados en https://www.40defiebre.com/donde-conseguir-imagenes-contenidos. Se han escogido las siguientes categorías:

- **Animal** [108]
- **Comida** [109]
- **Monumento** [110]
- **Paisaje** [111]
- **Persona** [112]

5.4.3. Resultados

Después de obtener los resultados de aplicar las diferentes técnicas seleccionadas en las distintas imágenes, se calculan las métricas para poder establecer un análisis comparativo entre ellas.

Se ha comprobado analíticamente que las imágenes generadas por el método de inteligencia artificial (ESRGAN) tienen mejor calidad empíricamente y visualmente que las imágenes generadas a través de los métodos clásicos vistos en el presente trabajo

Animal

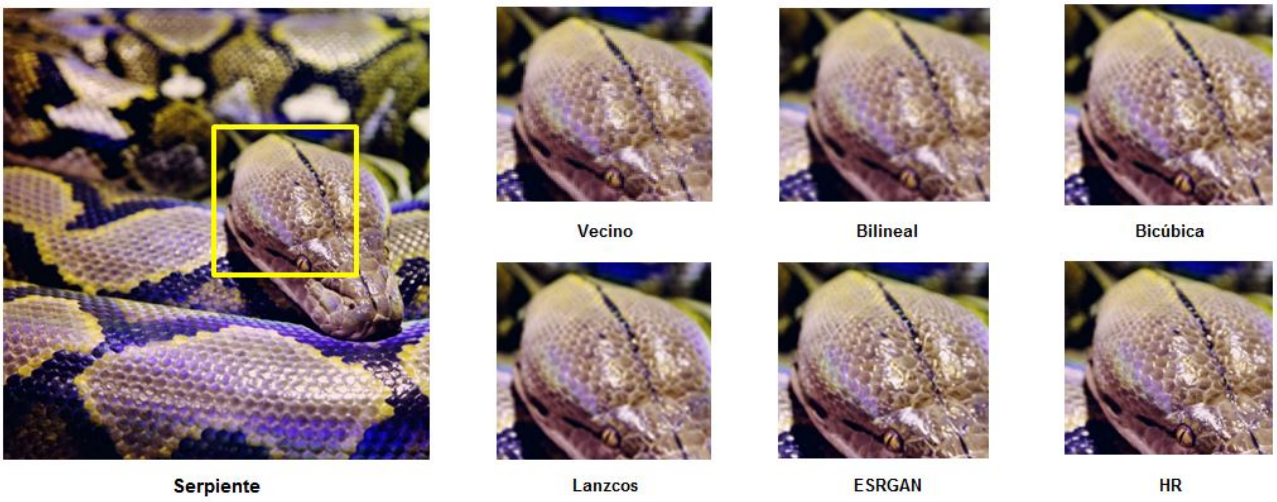


Figura 5.7: Comparación métodos en animal.png

	MSE	PSNR	SSIM	QILV
Vecino	47,909	31,327	0,777	1,000
Bilineal	40,732	32,031	0,829	0,999
Bicúbica	34,879	32,705	0,863	1,000
Lanczos	33,450	32,887	0,871	1,000
ESRGAN	36,573	34,499	0,865	1,000

Tabla 5.6: Comparación métricas animal

En esta secuencia, se observa que las imágenes entre sí no se desvían de la original, por no degenerarse con la introducción de ruidos, como lo atestigua los datos del PSNR. Asimismo, las imágenes que más se asemeja a la real es la de ESRGAN y Lanzcos, siendo las que tienen mayor valor en el SSIM. Tampoco se observan desenfoques de las imágenes, los valores obtenidos del QILV están muy próximos a la unidad.

Comida

Figura 5.8: Comparación métodos en comida.png

	MSE	PSNR	SSIM	QILV
Vecino	31,679	33,123	0,787	0,999
Bilineal	30,725	33,256	0,807	0,998
Bicúbica	29,249	33,470	0,828	0,999
Lanczos	28,660	33,558	0,833	0,999
ESRGAN	34,072	32,807	0,859	1,000

Tabla 5.7: Comparación métricas comida

En este grupo de imágenes, se observa que la imagen que más se parece a la de alta definición (HR), es la obtenida por ESRGAN teniendo en consideración el valor del índice de similitud estructural (SSIM). También se observa que el desenfoque de todas las imágenes es mínimo.

Monumento

Figura 5.9: Comparación métodos en monumento.png

	MSE	PSNR	SSIM	QILV
Vecino	49,575	31,178	0,654	0,999
Bilineal	47,881	31,329	0,663	0,999
Bicúbica	45,779	31,524	0,692	0,999
Lanczos	45,521	31,549	0,698	0,999
ESRGAN	49,222	31,209	0,706	1,000

Tabla 5.8: Comparación métricas monumento

En esta otra serie de imágenes, ESRGAN es la que más se asemeja a la original. Aunque el valor de métrica índice de calidad basado en la variación local (QILV) es cercano a la unidad, se aprecia cierto desenfoque en las imágenes.

Paisaje

Figura 5.10: Comparación métodos en paisaje.png

	MSE	PSNR	SSIM	QILV
Vecino	75,741	29,338	0,513	0,995
Bilineal	76,273	29,307	0,499	0,991
Bicúbica	73,859	29,447	0,553	0,995
Lanczos	73,476	29,469	0,564	0,995
ESRGAN	79,227	29,142	0,541	1,000

Tabla 5.9: Comparación métricas paisaje

La imagen del ESRGAN es similar a la de alta definición a pesar de que en esta se genera mucho error (MSE). En este caso los valores del PSNR se encuentran cerca de la horquilla de valores habituales que son entre 30 y 50 decibelios.

Persona

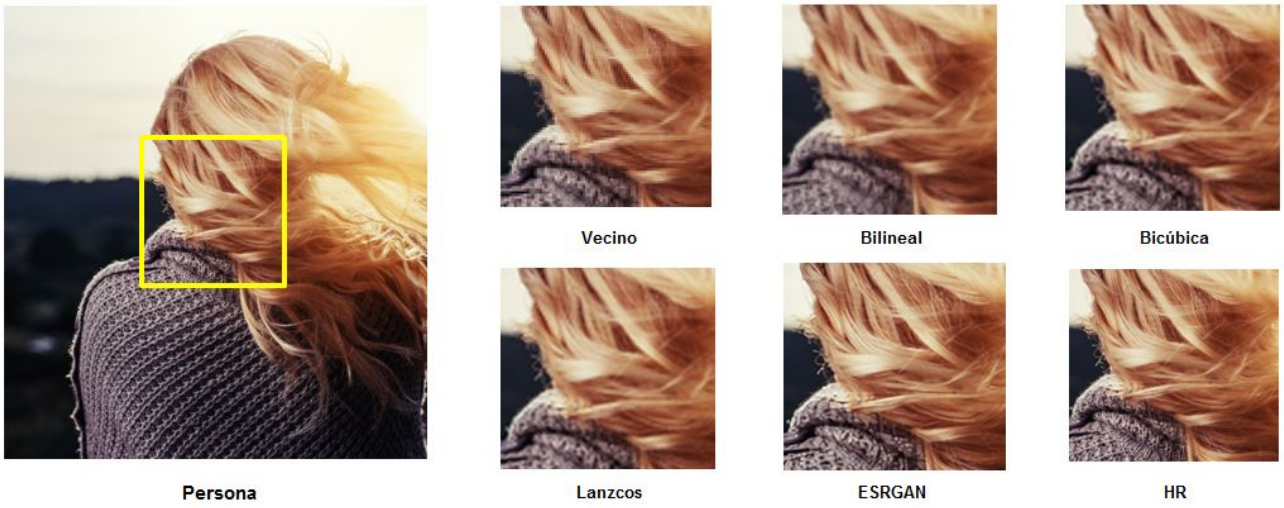


Figura 5.11: Comparación métodos en persona.png

	MSE	PSNR	SSIM	QILV
Vecino	31,077	33,206	0,812	1,000
Bilineal	30,267	33,321	0,820	1,000
Bicúbica	28,462	33,488	0,844	1,000
Lanczos	28,119	33,641	0,849	1,000
ESRGAN	32,854	32,965	0,864	1,000

Tabla 5.10: Comparación métricas persona

En el caso de la imagen Persona, los resultados visualmente se asemejan bastante a la imagen con alta resolución modificando el aparente desenfoque visual aunque la métrica que cuantifica dicha propiedad indica lo contrario. Las métricas tienen unos valores muy próximos entre sí.

Capítulo 6

Conclusiones

La realización del trabajo de fin de grado me ha supuesto una pequeña introducción al mundo de la inteligencia artificial para ampliar los conocimientos adquiridos en la asignatura Fundamentos de Inteligencia Artificial impartida por Dª. Mª Aránzazu Simón Hurtado como son: qué es una red neuronal artificial, tipos de redes que existen.

Desarrollando este proyecto se han ido obteniendo gran número de conclusiones y una serie de aprendizajes. En cuanto al desarrollo de las redes neuronales, se ratifica que el tiempo de análisis y la compresión del problema para generar un proceso de desarrollo es muy importante ya que cualquier componente mínimo puede afectar al resultado.

Se ha comprobado analíticamente con la propia experimentación que las imágenes generadas por la inteligencia artificial tienen mejor calidad empíricamente y visualmente que las imágenes generadas a través de los métodos clásicos vistos en el presente trabajo. Es debido a la utilización de las distintas imágenes en los entrenamientos que modifica el peso de los perceptrones de las distintas capas, en función de las propias texturas y patrones encontrados, proporcionando unos mejores resultados que el aplicar las fórmulas matemáticas de los distintos métodos (vecino más cercano, bilineal, bicúbica, Lanczos). Asimismo, la mejoría de los resultados generados en los algoritmos de inteligencia artificial sobre los "métodos clásicos" radica en la importancia del *dataset* de entrenamiento, cuanto mayor sea la heterogeneidad de las imágenes de alta resolución y contrastes, (paisajes, animales, objetos, personas..), se asemejarán más los resultados obtenidos a la realidad.

6.1. Líneas de trabajo futuras

Indicar que este trabajo de fin de grado, Superresolution, es un proyecto iniciado, con muchas alternativas de ampliación en un futuro a corto y medio plazo, máxime que la inteligencia artificial avanza diariamente. Entre otras líneas de trabajo futuro, podemos destacar:

- Ampliación de este trabajo con el estudio de otros algoritmos que empleen inteligencia artificial como son **SRCNN**: *Super-Resolution Convolutional Neural Network* (Red neuronal convolucional de superresolución), **EDSR**: *Enhanced Deep Residual Networks for Single Image Super-Resolution* (Red mejorada de residuos profundos para una sola imagen de superresolución), **EnhanceNet**. Comparando entre ellos diversos valores como puede ser la rapidez del entrenamiento, de la generación de los resultados, cual de ellos genera la imagen más parecida a la de alta definición.

- La posible construcción del proyecto ESRGAN usando otras librerías y/o leguajes de programación y comprarlo con el original. Se podría analizar si interfiere algo en la obtención de los resultados como en el tiempo de obtención de ellos y el entrenamiento.

- La creación de una métrica o un *benchmark* que cumpla las propiedades para ser una buena métrica: linealidad, fiabilidad, repetitividad, facilidad de medición, consistencia e independencia. Permitiendo la comparación de los distintos algoritmos.

- La experimentación con los **dataset** de entrenamiento y validación para la obtención de resultados que se asemejen lo más posibles a las imágenes de alta definición. Dichas conclusiones podrían ser cuántas imágenes se

necesitan en el entrenamiento para mejorar la calidad de los resultados teniendo un número fijo de validación, cómo afecta en la calidad de los resultados el número de imágenes de validación. También cómo afecta el número de imágenes a los tiempos de computación ya sea a nivel del entrenamiento como al de generar los resultados.

Bibliografía

[1] PetaPixel, 'NVIDIA's Al-Powered 'Content-Aware Fill' is Mind-Blowing', 2018. [Online]. Disponible: https://petapixel.com/2018/04/23/nvidias-ai-powered-content-aware-fill-is-mind-blowing/ [Último acceso Junio. 10, 2019].

[2] Deepsense.ai, "Using deep learning for Single Image Super Resolution". [Online]. Disponible: https://deepsense.ai/using-deep-learning-for-single-image-super-resolution/ [Último acceso Junio. 10, 2019].

[3] DeepAI, "Super Resolution API". [Online]. Disponible: https://deepai.org/machine-learning-model/torch-srgan [Último acceso Junio. 10, 2019].

[4] Let's Enhance.io, "Image enhancement power by IA". [Online]. Disponible: https://letsenhance.io/ [Último acceso Junio. 10, 2019].

[5] TensorFLow. [Online]. Disponible: https://www.tensorflow.org/ [Último acceso Junio. 10, 2019].

[6] The Coding Train, "Session 4 - Neural Networks - Intelligence and Learning". [Online]. Disponible: https://www.youtube.com/playlist?list=PLRqwX-V7Uu6Y7MdSCaIfsxc561QI0U0Tb [Último acceso Junio. 10, 2019].

[7] Idiot Developer, "Multilayer Neural Network in Tensorflow Python". [Online]. Disponible: https://www.youtube.com/watch?v=FbJw8J0rTyQ [Último acceso Junio. 10, 2019].

[8] Apsl, "TENSOR FLOW PARA PRINCIPIANTES (I)". [Online]. Disponible: https://www.apsl.net/blog/2017/12/05/tensor-flow-para-principiantes-i/ [Último acceso Junio. 10, 2019].

[9] La mirada del Golem, "TensorFlow, machine learning I: Perceptrón multicapa". [Online]. Disponible: http://miradadelgolem.blogspot.com/2018/05/tensorflow-machine-learning-i.html [Último acceso Junio. 10, 2019].

[10] Cognitive Class, "Multilayer Perceptron with TensorFlow - Deep Learning with Tensorflow". [Online]. Disponible: https://www.youtube.com/watch?v=kDHR7MjZyTQ [Último acceso Junio. 10, 2019].

[11] Arijit Mukherjee, "Writing a Perceptron in Tensorflow - part 2". [Online]. Disponible: https://www.youtube.com/watch?v=xp6kHUwK51Q [Último acceso Junio. 10, 2019].

[12] Colah's blog, "Understanding LSTM Networks". [Online]. Disponible: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Último acceso Junio. 10, 2019].

[13] The data science blog, "An Intuitive Explanation of Convolutional Neural Networks". [Online]. Disponible: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ [Último acceso Junio. 10, 2019].

[14] FreeCodeCamp, "An intuitive guide to Convolutional Neural Networks". [Online]. Disponible: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/ [Último acceso Junio. 10, 2019].

[15] Skymind, "A Beginner's Guide to Convolutional Neural Networks (CNNs)". [Online]. Disponible: https://skymind.ai/wiki/convolutional-network [Último acceso Junio. 10, 2019].

[16] "Convolutional Neural Networks (CNNs / ConvNets)". [Online]. Disponible: http://cs231n.github.io/convolutional-networks/ [Último acceso Junio. 10, 2019].

[17] Twards Data Science, "Convolutional Neural Networks from the ground up". [Online]. Disponible: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1 [Último acceso Junio. 10, 2019].

[18] Jaakko Lehtinen, "Noise2Noise: Learning Image Restoration without Clean Data". [Online]. Disponible: https://arxiv.org/pdf/1803.04189.pdf [Último acceso Junio. 10, 2019].

[19] Jin Yamanaka, "Fast and Accurate Image Super Resolution by Deep CNN with Skip Connection and Network in Network". [Online]. Disponible: https://arxiv.org/ftp/arxiv/papers/1707/1707.05425.pdf [Último acceso Junio. 10, 2019].

[20] Wenming Yang, "Deep Learning for Single Image Super-Resolution:A Brief Review". [Online]. Disponible: https://arxiv.org/pdf/1808.03344.pdf [Último acceso Junio. 10, 2019].

[21] Interactive Geometry Lab, "A Fully Progressive Approach to Single-Image Super-Resolution". [Online]. Disponible: https://igl.ethz.ch/projects/prosr/ [Último acceso Junio. 10, 2019].

[22] Bee Lim, "Enhanced Deep Residual Networks for Single Image Super-Resolution". [Online]. Disponible: https://arxiv.org/pdf/1707.02921.pdf [Último acceso Junio. 10, 2019].

[23] Yaniv Romano, "RAISR: Rapid and Accurate Image Super Resolution". [Online]. Disponible: https://arxiv.org/pdf/1606.01299.pdf [Último acceso Junio. 10, 2019].

[24] Brandon Amos, "Image Completion with Deep Learning in TensorFlow". [Online]. Disponible: http://bamos.github.io/2016/08/09/deep-completion/ [Último acceso Junio. 10, 2019].

[25] Microsoft Azure, "Precios de máquinas virtuales Linux". [Online]. Disponible: https://azure.microsoft.com/es-es/pricing/details/virtual-machines/linux/ [Último acceso Abril. 3, 2020].

[26] Wikipedia, "Comparison gallery of image scaling algorithms". [Online]. Disponible: https://en.wikipedia.org/wiki/Comparison_gallery_of_image_scaling_algorithms [Último acceso Abril. 15, 2020].

[27] "Escalar la imagen". [Online]. Disponible: https://docs.gimp.org/2.10/es/gimp-image-scale.html [Último acceso Junio. 10, 2019].

[28] Emezeta.com, "Interpolación de imágenes". [Online]. Disponible: https://www.emezeta.com/articulos/interpolacion-de-imagenes [Último acceso Junio. 10, 2019].

[29] Javier Vidal Valenzuela, "Interpolación de Formas en Imágenes Usando Morfología Matemática", M.S. thesis, Universidad Nacional de Educación a Distancia, España, 2008. [Online]. Disponible: http://oa.upm.es/1282/1/JAVIER_VIDAL_VALENZUELA.pdf [Último acceso Junio. 10, 2019].

[30] Javier Colomé Abril, "Escalado de imagen mediante la interpolación por mínimos cuadrados y comparación con los métodos clásicos". Ph.D, Universidad Politécnica de Madrid, Madrid, España, 2015. [Online]. Disponible: http://www.issi.uned.es/Master_ISSI/WebMISSI/RepositorioTFM/2015/15S_MemoriaTFdM_ISI_TipoB_Javier_ColomeAbril.pdf [Último acceso Junio. 10, 2019].

[31] Wikipedia, "Escalado (geometría)". [Online]. Disponible: https://es.wikipedia.org/wiki/Escalado_(geometr%C3%ADa) [Último acceso Junio. 10, 2019].

[32] Wikipedia, "Interpolación (fotografía)". [Online]. Disponible: https://es.wikipedia.org/wiki/Escalado_(geometr%C3%ADa) [Último acceso Junio. 10, 2019].

[33] Univerity of Tartu, "Digital Image Processing". [Online]. Disponible: https://sisu.ut.ee/imageprocessing/book/3 [Último acceso Junio. 10, 2019].

[34] Faisal A. Khan, "Image Interpolation Techniques in Digital Image Processing". [Online]. Disponible: https://pdfs.semanticscholar.org/d6bd/5f7fb72a04c06f1ff0bb275ec19d964e93bd.pdf [Último acceso Junio. 10, 2019].

[35] ResearchGate, "Image Interpolation Techniques in Digital Image Processing: An Overview". [Online]. Disponible: https://www.researchgate.net/publication/301889708_Image_Interpolation_Techniques_in_Digital_Image_Processing_An_Overview [Último acceso Junio. 10, 2019].

[36] TheAILearner, "Image Processing – Nearest Neighbour Interpolation". [Online]. Disponible: https://theailearner.com/2018/12/29/image-processing-nearest-neighbour-interpolation/ [Último acceso Junio. 10, 2019].

[37] ScienceDirect, "Nearest Neighbor Interpolation". [Online]. Disponible: https://www.sciencedirect.com/topics/engineering/nearest-neighbor-interpolation [Último acceso Junio. 10, 2019].

[38] Giassa.net, "I – Nearest Neighbour Interpolation". [Online]. Disponible: https://www.giassa.net/?page_id=207 [Último acceso Junio. 10, 2019].

[39] Cambridge in colour, "DIGITAL IMAGE INTERPOLATION". [Online]. Disponible: https://www.cambridgeincolour.com/tutorials/image-interpolation.htm [Último acceso Junio. 10, 2019].

[40] Wikipedia, 'Nearest-neighbor interpolation'. [Online]. Disponible: http://ocw.uniovi.es/pluginfile.php/4994/mod_resource/content/5/T3b_Interpolacion_imagen_2.pdf [Último acceso Marzo. 10, 2020].

[41] Escuela de Ingeniería Informática de Oviedo, 'Interpolación de Imágenes', 2018. [Online]. Disponible: http://ocw.uniovi.es/pluginfile.php/4994/mod_resource/content/5/T3b_Interpolacion_imagen_2.pdf [Último acceso Marzo. 10, 2020].

[42] TheAILearner, "Image Processing – Bilinear Interpolation". [Online]. Disponible: https://theailearner.com/2018/12/29/image-processing-bilinear-interpolation/ [Último acceso Junio. 10, 2019].

[43] Wikipedia, "Bilinear interpolation". [Online]. Disponible: https://en.wikipedia.org/wiki/Bilinear_interpolation [Último acceso Marzo. 15, 2020].

[44] Wikipedia, "Bicubic interpolation". [Online]. Disponible: https://en.wikipedia.org/wiki/Bicubic_interpolation [Último acceso Marzo. 15, 2020].

[45] Wikipedia, "Lanczos resampling". [Online]. Disponible: https://en.wikipedia.org/wiki/Lanczos_resampling [Último acceso Junio. 10, 2019].

[46] IJARCCE, "A Comparative Analysis of Image Interpolation Algorithms". [Online]. Disponible: https://ijarcce.com/wp-content/uploads/2016/02/IJARCCE-7.pdf [Último acceso Junio. 10, 2019].

[47] Intel, "Developer Reference for Intel Integrated Performance Primitives 2019", 2019. [Online]. Disponible: https://software.intel.com/en-us/ipp-dev-reference-lanczos-interpolation [Último acceso Junio. 10, 2019].

[48] The blog at the bottom of the sea, "Resizing Images With Bicubic Interpolation". [Online]. Disponible: https://blog.demofox.org/2015/08/15/resizing-images-with-bicubic-interpolation/ [Último acceso Junio. 10, 2019].

[49] Ubuntu forums, "windows.h". [Online]. Disponible: https://ubuntuforums.org/showthread.php?t=533304 [Último acceso Junio. 10, 2019].

[50] TheAILearner, "Image Interpolation using OpenCV-Python". [Online]. Disponible: https://theailearner.com/2018/11/15/image-interpolation-using-opencv-python/ [Último acceso Junio. 10, 2019].

[51] Life2Coding, "HOW TO RESIZE IMAGES IN OPENCV PYTHON USING DIFFERENT INTERPOLATION METHODS". [Online]. Disponible: https://www.life2coding.com/how-to-resize-images-in-opencv-python-using-different-interpolation-methods/ [Último acceso Junio. 10, 2019].

[52] OpenCV, "Geometric Image Transformations". [Online]. Disponible: https://docs.opencv.org/3.0-beta/modules/imgproc/doc/geometric_transformations.html#resize [Último acceso Junio. 10, 2019].

[53] OpenCV, "Geometric Image Transformations". [Online]. Disponible: https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#resize [Último acceso Junio. 10, 2019].

[54] Culturacion, "¿Qué son redes neuronales?". [Online]. Disponible: https://culturacion.com/que-son-redes-neuronales/ [Último acceso Mayo. 31, 2020].

[55] Wikipedia, "Neurona". [Online]. Disponible: https://es.wikipedia.org/wiki/Neurona [Último acceso Marzo. 15, 2020].

[56] freepik, "Diagrama de la célula de vástago en el fondo blanco vector gratuito". [Online]. Disponible: https://www.freepik.es/vector-gratis/diagrama-celula-vastago-fondo-blanco_2480958.htm#page=1&query=neurona&position=0 [Último acceso Marzo. 15, 2020].

[57] Xataka, "Las redes neuronales: qué son y por qué están volviendo". [Online]. Disponible: https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo [Último acceso Marzo. 15, 2020].

[58] Wikipedia, "Perceptrón". [Online]. Disponible: https://es.wikipedia.org/wiki/Perceptr%C3%B3n [Último acceso Marzo. 15, 2020].

[59] ResearchGate, "Capas de una Red Neuronal". [Online]. Disponible: https://www.researchgate.net/figure/Figura-III4-Capas-de-una-Red-Neuronal-Capa-de-entrada-neuronas-que-reciben-datos-o_fig3_315762548 [Último acceso Marzo. 15, 2020].

[60] R. Prieto, "El modelo neuronal de McCulloch y Pitts". [Online]. Disponible: http://medicinaycomplejidad.org/pdf/reciente/r31459.pdf [Último acceso Junio. 1, 2020].

[61] Wikipedia, "Red neuronal recurrente". [Online]. Disponible: https://es.wikipedia.org/wiki/Red_neuronal_recurrente [Último acceso Marzo. 15, 2020].

[62] Marcos Gestal Pose, "Introducción a las Redes de NEuronas Artificiales", Universidad da Coruña, Coruña, España, 2015. [Online]. Disponible: http://sabia.tic.udc.es/mgestal/cv/RNATutorial/TutorialRNA.pdf [Último acceso Marzo. 16, 2020].

[63] Wikipedia, "Perceptrón". [Online]. Disponible: https://es.wikipedia.org/wiki/Perceptr%C3%B3n [Último acceso Marzo. 15, 2020].

[64] The Python Package Index, "SSIM-PIL 1.0.10". [Online]. Disponible: https://pypi.org/project/SSIM-PIL/ [Último acceso Junio. 10, 2020].

[65] Pirm2018, "The pirm challenge on perceptual super resolution". [Online]. Disponible: https://www.pirm2018.org/PIRM-SR.html [Último acceso Junio. 10, 2019].

[66] Pirm2018, "Workshop and Challenge on Perceptual Image Restoration and Manipulation". [Online]. Disponible: https://pirm2018.org/ [Último acceso Junio. 10, 2019].

[67] Xinntao, "ECCV18 Workshops - Enhanced SRGAN. Champion PIRM Challenge on Perceptual Super-Resolution (Third Region)". [Online]. Disponible: https://github.com/xinntao/ESRGAN [Último acceso Junio. 10, 2019].

[68] Anaconda, "Anaconda Distribution". [Online]. Disponible: https://www.anaconda.com/distribution/#linux [Último acceso Junio. 10, 2019].

[69] Pytorch. [Online]. Disponible: https://pytorch.org/ [Último acceso Junio. 10, 2019].

[70] Numpy. [Online]. Disponible: https://www.numpy.org/ [Último acceso Junio. 10, 2019].

[71] OpenCV. [Online]. Disponible: https://opencv.org/ [Último acceso Junio. 10, 2019].

[72] OpenCV, "Geometric Image Transformations". [Online]. Disponible: https://docs.opencv.org/master/da/d54/group__imgproc__transform.html [Último acceso Marzo. 10, 2020].

[73] OpenCV, "Image file reading and writing". [Online]. Disponible: https://docs.opencv.org/master/d4/da8/group__imgcodecs.html [Último acceso Marzo. 10, 2020].

[74] Cornell University, "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks". [Online]. Disponible: https://arxiv.org/abs/1809.00219 [Último acceso Marzo. 10, 2020].

[75] "The PIRM dataset". [Online]. Disponible: https://pirm.github.io/ [Último acceso Junio. 10, 2019].

[76] Keras Documentation, "Keras: THe Python Deep Learning library". [Online]. Disponible: https://keras.io/ [Último acceso Junio. 10, 2019].

[77] Tejalal Choudhary, "Print PyTorch model summary".[Online]. Disponible: https://tejalal.wordpress.com/2018/12/11/print-pytorch-model-summary/ [Último acceso Junio. 10, 2019].

[78] Xinntao, "Basic Super-Resolution codes for development. Includes ESRGAN, SFT-GAN for training and testing.". [Online]. Disponible: https://github.com/xinntao/BasicSR [Último acceso Junio. 10, 2019].

[79] PyTorch, "Torch.nn". [Online]. Disponible: https://pytorch.org/docs/stable/nn.html [Último acceso Junio. 10, 2019].

[80] PyTorch, "Torch.tensor". [Online]. Disponible: https://pytorch.org/docs/stable/tensors.html [Último acceso Junio. 10, 2019].

[81] Python, "10.7. glob — Unix style pathname pattern expansion". [Online]. Disponible: https://docs.python.org/2/library/glob.html [Último acceso Junio. 10, 2019].

[82] "DIV2K dataset". [Online]. Disponible: https://data.vision.ee.ethz.ch/cvl/DIV2K/ [Último acceso Junio. 16, 2019]

[83] Google Drive, "2K_resolution". [Online]. Disponible: https://drive.google.com/drive/folders/1B-uaxvV9qeuQ-t7MFiN1oEdA6dKnj2vW [Último acceso Junio. 16, 2019]

[84] "Generative Models". [Online]. Disponible: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/generative_models.html [Último acceso Marzo. 15, 2020]

Wikipedia, "Peak signal-to-noise ratio". [Online]. Disponible: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio [Último acceso Junio. 10, 2019].

[85] MathWorks, "PSNR". [Online]. Disponible: https://www.mathworks.com/help/vision/ref/psnr.html [Último acceso Junio. 10, 2019].

[86] MathWorks, "SSIM". [Online]. Disponible: https://es.mathworks.com/help/images/ref/ssim.html [Último acceso Junio. 10, 2019].

[87] ResearchGate, "Image quality metrics: PSNR vs. SSIM". [Online]. Disponible: https://www.researchgate.net/publication/220931731_Image_quality_metrics_PSNR_vs_SSIM [Último acceso Junio. 10, 2019].

[88] Javier Silvestre-Blanes, "Técnicas de Evaluación de la Calidad de la Imagen. Tendencias y métricas basadas en Bordes". [Online]. Disponible: https://upcommons.upc.edu/bitstream/handle/2117/9805/Image.pdf [Último acceso Junio. 10, 2019].

[89] Wikipedia, "Error cuadrático medio". [Online]. Disponible: https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio [Último acceso Junio. 10, 2019].

[90] Zhou Wang, "Multi-scale structural similarity for image quality assesment". [Online]. Disponible: https://ece.uwaterloo.ca/~z70wang/publications/msssim.pdf [Último acceso Junio. 16, 2019].

[91] "Capítulo 6 Simulaciones y resultados". [Online]. Disponible: http://catarina.udlap.mx/u_dl_a/tales/documentos/meie/rosas_o_mc/capitulo6.pdf [Último acceso Marzo. 16, 2020].

[92] FREMAP, "Recomendaciones básicas sobre iluminación". [Online]. Disponible: https://www.icv.csic.es/prevencion/Documentos/breves/FREMAP/iluminacion.pdf [Último acceso Marzo. 16, 2020].

[93] Wikipedia, "Contraste". [Online]. Disponible: https://es.m.wikipedia.org/wiki/Contraste [Último acceso Marzo. 16, 2020].

[94] Santiago Aja-Fernández, "Full Reference Image Quality Assessment based on Local Statistics". [Online]. Disponible: https://www.lpi.tel.uva.es/~santi/personal/docus/qilv_techrep.pdf [Último acceso Junio. 16, 2019].

[95] V. K. Bhola, "Image Quality Assessment Techniques". [Online]. Disponible: https://pdfs.semanticscholar.org/1d10/8ad4dcf1b9752b285970e6d2b0825f9a6b47.pdf [Último acceso Junio. 16, 2019].

[96] Wikipedia, "Luma (video)". [Online]. Disponible: https://en.wikipedia.org/wiki/Luma_(video) [Último acceso Marzo. 16, 2020].

[97] Gabriel Prieto, "Structural similarity index for image quality assessment in radiological images". [Online]. Disponible: https://www.ucm.es/data/cont/media/www/pag-88221/JMI-17059R_online.pdf [Último acceso Marzo. 16, 2020].

[98] M. Tech, "Comparison of image quality metrics". [Online]. Disponible: https://www.ijert.org/research/comparison-of-image-quality-metrics-IJERTV1IS4105.pdf [Último acceso Marzo. 16, 2020].

[99] Santiago Aja-Fernández, "Image Quality Assessment based on Local Variance". [Online]. Disponible: http://poseidon.tel.uva.es/~santi/personal/embc06.pdf [Último acceso Marzo. 16, 2020].

[100] LMDB, "Lightning Memory-Mapped Database Manager (LMDB)". [Online]. Disponible: http://www.lmdb.tech/doc/ [Último acceso Marzo. 16, 2020].

[101] Wikipedia, "Lightning Memory-Mapped Database". [Online]. Disponible: https://en.wikipedia.org/wiki/Lightning_Memory-Mapped_Database [Último acceso Marzo. 16, 2020].

[102] Sysmas, "LIGHTNING MEMORY-MAPPED DATABASE An extraordinarily fast, memory-efficient database we developed for the Symas OpenLDAP Project.". [Online]. Disponible: https://symas.com/lmdb/ [Último acceso Marzo. 16, 2020].

[103] Yuri Torres, Rendimiento y Evaluación de Computadoras: Tema 2.1: Métricas de Rendimiento", 2017.

[104] "Cómo escribir un Trabajo Fin de Grado". [Online]. Disponible: http://personales.upv.es/fjabad/pfc/comoEscribir.pdf [Último acceso Junio. 10, 2019].

[105] Yochai Blau,"The Perception-Distortion Tradeoff". [Online]. Disponible: http://openaccess.thecvf.com/content_cvpr_2018/papers/Blau_The_Perception-Distortion_Tradeoff_CVPR_2018_paper.pdf [Último acceso Junio. 10, 2019].

[106] Christian Ledig, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network". [Online]. Disponible: https://arxiv.org/pdf/1609.04802.pdf [Último acceso Junio. 10, 2019].

[107] DedeLinux, "Cómo manipular imágenes desde el terminal". [Online]. Disponible: https://blog.desdelinux.net/como-manipular-imagenes-desde-el-terminal/ [Último acceso Enero. 15, 2020].

[108] Sharon McCutcheon, Unsplash. [Online]. Disponible: https://arxiv.org/pdf/1609.04802.pd [Último acceso Enero. 15, 2020].

[109] Gratisography. [Online]. Disponible: https://arxiv.org/pdf/1609.04802 [Último acceso Enero. 15, 2020].

[110] Takeshi Hayakawa, lifeofpix, "Small japanese arch". [Online]. Disponible: https://www.lifeofpix.com/photo/small-japanese-arch/ [Último acceso Enero. 15, 2020].

[111] Nextvoyage, Pexels. [Online]. Disponible: https://www.pexels.com/photo/aerial-photography-of-waterfalls-2618714/ [Último acceso Enero. 15, 2020].

[112] Magdeleine.co. [Online]. Disponible: https://cdn.magdeleine.co/wp-content/uploads/2016/04/26145429985_1cbb5d9440_o.jpg [Último acceso Enero. 15, 2020].

[113] ComputerHoy, "Linux: Ver especificaciones completas de un PC". [Online]. Disponible: https://computerhoy.com/noticias/software/linux-ver-especificaciones-completas-pc-78235 [Último acceso Junio. 10, 2019].

[114] DesdeLINUX, "Comandos para conocer el sistema (identificar hardware y algunas configuraciones de software)", 2018. [Online]. Disponible: https://blog.desdelinux.net/comandos-para-conocer-el-sistema-identificar-hardware-y-algunas-configuraciones-de-software/ [Último acceso Junio. 10, 2019].

[115] Paraiso Linux, 'Ver detalles tecnicos del hardware de tu PC en Linux", 2010. [Online]. Disponible: https://paraisolinux.com/ver-detalles-tecnicos-del-hardware-de-tu-pc-en-linux/ [Último acceso Junio. 10, 2019].

[116] KDe Blog, "Cómo saber que tarjeta gráfica tengo en mi linux", 2018. [Online]. Disponible: https://www.kdeblog.com/como-saber-que-tarjeta-grafica-tengo-en-mi-linux.html [Último acceso Junio. 10, 2019].

[117] Wikipedia, "Artificial neuron". [Online]. Disponible: https://en.wikipedia.org/wiki/Artificial_neuron [Último acceso Marzo. 30, 2020].