Aplicación Android de fotografía de superresolución

Investigación para resolver el problema de superresolución en dispositivos móviles

Universidad de Oviedo

Grado en Ingeniería Informática del Software

Curso 2021-2022

Trabajo Fin de Grado

Guillermo Fernández-Campoamor Fernández

Tutor: Pablo Revuelta Sanz

Oviedo, Julio 2022

Resumen

En la actualidad el campo de la inteligencia artificial avanza cada día de forma más rápida, descubriéndose nuevas aplicaciones y metodologías para mejorar la eficacia de esta. Hay muchos tipos de problemas que hace años se resolvían mediante algoritmos complejos de forma poco eficiente, pero gracias a la aparición de las redes neuronales se han resuelto de formas novedosas y con resultados que hace años hubiesen sido inverosímiles. Este proyecto nace con la idea de aplicar este concepto al campo de la superresolución, en el que hasta hace años existían únicamente algoritmos simples para realizar el reescalado. Hoy en día es posible obtener resultados “milagrosos” gracias a la aplicación de la inteligencia artificial en este campo.

Este documento proporciona una explicación generalizada del problema de superresolución, explicando desde la base de los métodos clásicos de reescalado a una introducción sobre redes neuronales y los diferentes modelos neuronales existentes. Como resultado de este proceso de investigación se han escogido las redes generativas adversarias como modelo de red neuronal, para resolver el problema de superresolución. Se ha programado un modelo neuronal de este tipo junto a su integración en una aplicación móvil Android.

Abstract

Currently, the field of artificial intelligence is advancing faster every day, discovering new applications and methodologies to improve its efficiency. There are many types of problems that years ago were solved inefficiently by complex algorithms, thanks to the emergence of neural networks these have been solved in new ways and with results that years ago would have been unlikely. This project was born with the idea of applying this concept to the field of super-resolution, in which until a few years ago only simple algorithms existed. Today it is possible to obtain "miraculous" results thanks to the application of artificial intelligence in this field.

This paper provides a generalized explanation of the super-resolution problem, explaining from the basis of classical rescaling methods to an introduction on neural networks and the different existing neural models. As a result of this research process, the generative adversarial networks have been chosen to solve the super resolution problem. Such a neural model has been programmed together with its integration in an Android mobile application.

Contenido

Índice de ilustraciones

Índice de tablas

Glosario

Existe una variedad de términos técnicos que van a ser usados en la documentación. En este apartado se hace una agrupación de esos términos junto a su definición.

- Activity: clase Java que se corresponde a una vista de la aplicación.
- Alucinar o alucinación: término técnico que implica generar patrones aprendidos por una red durante su entrenamiento.
- Artefactos: distorsiones o errores visibles en una imagen.
- Bordes de sierra: deformaciones en los bordes de los objetos presentes en las imágenes, siendo visibles artefactos o deformaciones del borde.
- Capa Dense: es un tipo de capa que recibe el output de todas las neuronas de la capa que le precede, esta capa realiza una multiplicación de matrices para obtener una nueva salida.
- Época: medida de duración del entrenamiento que representa el proceso en el que un conjunto de datos completo se introduce hacia delante y hacia atrás de la red neuronal una vez.
- Filtros de la capa convolucional: es un valor entero el número de filtros que tiene la capa convolucional como salida. Estos filtros ayudan a extraer características específicas de los valores de entrada.
- Firmware: conocido como soporte lógico inalterable, es un programa compuesto por las instrucciones básicas capaces de controlar los circuitos electrónicos de un dispositivo.
- GAN: acrónimo inglés para el término redes generativas adversarias, “generative adversarial network”.
- Imagen rasterizada: imagen compuesta por un mapa de bits o píxeles.
- Iteración: generalmente, es el número de lotes necesario para completar una época.
- Keras: biblioteca de código abierto para la implementación de redes neuronales.
- Kernel: el kernel o matriz de convolución es una matriz usada para diferentes ajustes de la imagen como desenfoque, enfoque, detección de bordes y más. Para obtener estos ajustes se realiza una convolución entre la imagen de entrada y el kernel.
- Lote: subconjunto de un set de datos. También llamado “batch”.
- Pixel art: es el diseño artístico digital donde las imágenes son editadas a nivel de pixel.
- Tensorflow Lite: biblioteca que dispone de un conjunto de herramientas para exportar modelos neuronales a dispositivos móviles.
- Tensorflow: biblioteca de código abierto para aprendizaje automático.

- tflite: formato de archivos de las redes neuronales exportadas a Tensorflow Lite.

Introducción

Interés del proyecto

El escalado de la resolución de imágenes lleva siendo analizado e investigado de forma notoria desde la última década, abarcado principalmente por técnicas basadas únicamente en la utilización de la información local de la imagen a nivel de píxel. Una de las técnicas más conocidas en este caso es el apilamiento de múltiples imágenes; el origen de esta técnica se atribuye a las cámaras de la marca HasselBlad [1], en concreto al modelo H4D-200MS. Este modelo realizaba hasta 6 tomas de una imagen combinándolas para obtener una imagen resultado de hasta cuatro veces la resolución original.

No ha sido hasta estos últimos años en los que se ha incrementado su interés debido a la creciente presencia de las redes neuronales en el panorama computacional. Las técnicas de Machine Learning y Deep Learning han logrado mejorar los resultados de la superresolución, lo que ha permitido su aplicación a más campos. Sin embargo, todavía existen dudas sobre su eficacia y fiabilidad.

Uno de los campos en los que se está empezando a estudiar cómo implementar la superresolución haciendo uso de redes neuronales es en el de la medicina [36]. Uno de los principales usos que se le está dando a esta tecnología dentro de este campo es el escalado y mejora de la imagen de radiografías, ecografías y TAC craneales [23]. Estas aplicaciones abren el debate de cuánto deberíamos fiarnos de los resultados que nos proporcionan estas redes ya que podríamos tener una imagen de entrada ambigua y no podemos estar totalmente seguros de si algún detalle de la imagen estaba originalmente ahí o es un artefacto de la red neuronal.

El interés principal de este proyecto es investigar y agrupar los diferentes algoritmos y mecanismos que existen actualmente para aplicar la superresolución, así como diseñar y probar una nueva configuración. Por último, desarrollar una aplicación Android que implemente de forma local una red neuronal capaz de escalar una imagen a 4 veces su resolución original usando métodos y funciones de la librería OpenCV.

Panorama actual

El panorama actual de superresolución se centra en la investigación de nuevos modelos de redes neuronales capaces de mejorar los resultados de superresolución actuales. Los dos principales modelos de redes que se están aplicando son redes neuronales convolucionales y redes generativas adversarias, ambas teniendo características relacionadas.

Existe un tipo de redes convolucionales que fueron diseñadas específicamente para resolver el problema de la superresolución en diagnósticos médicos, conocido como redes U-Net [37]. El mejor ejemplo de esta aplicación de este tipo de red es el proyecto Decrappify [24].

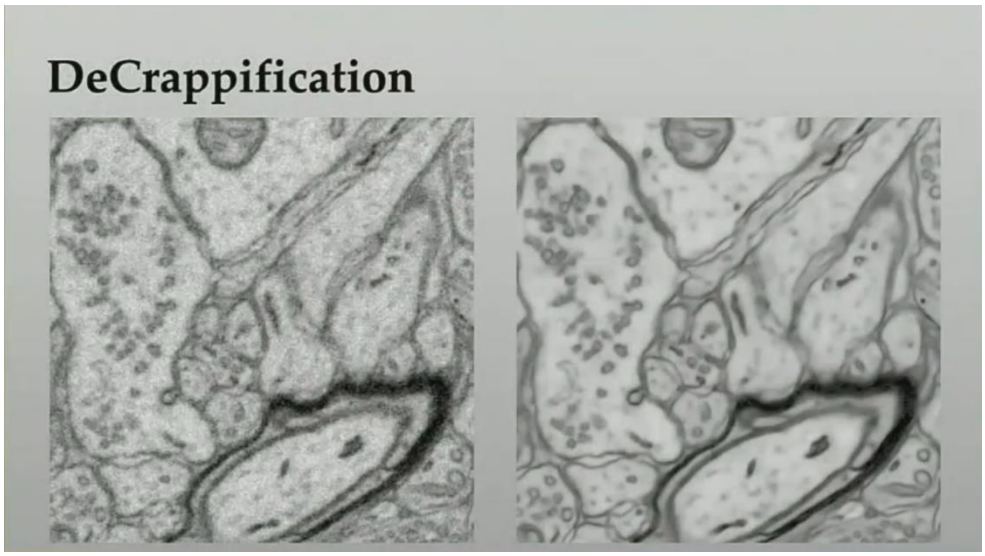


Ilustración 1 Demostración de Decrappify obtenida de la conferencia de Meta "New Approaches to Image and Video Reconstruction Using Deep Learning"

El principal problema de las redes neuronales es la cantidad de potencia computacional y tiempo que se necesita para realizar un entrenamiento adecuado. El entrenamiento de las redes se puede realizar sobre una unidad de procesamiento gráfico (GPU) o una unidad de procesamiento de datos (DPU) para poder obtener resultados en un tiempo de ejecución bastante menor.

En el campo de la superresolución ya existen varios proyectos conocidos, de los cuales la mayoría de ellos hacen uso de redes generativas adversarias. Algunos de estos proyectos son:

- TecoGAN, proyecto para restaurar y reescalar películas antiguas [2,21]
- Real-ESRGAN [3], proyecto para resolver el problema de superresolución en imágenes.

Marco Teórico

En este apartado de la memoria se van a abarcar diferentes conceptos relacionados con el problema de la superresolución, escalando poco a poco hacia conceptos relacionados con la solución escogida para generar el modelo neuronal: una red generativa adversaria.

Técnicas clásicas de reescalado

Existen varios tipos de técnicas utilizadas para escalar de resolución imágenes rasterizadas. Antes de entender estas técnicas deberíamos entrar más en detalle de cómo funciona la información de una imagen, concretamente en definir qué es un píxel. Los píxeles son la unidad mínima de la que está compuesta una imagen rasterizada, son unidades de color definidas numéricamente donde cada valor indica cuál es la intensidad dada a cada canal de color.

El grupo de técnicas clásicas que vamos a analizar trabajan usando la información local de la imagen a nivel de píxel, pero realmente no llegan a entender el contenido de la imagen a la hora de hacer el reescalado. La explicación de estas técnicas está enfocada en el reescalado de imágenes en dos dimensiones.

Vecino más cercano

El método de vecino más cercano o también conocido como método de proximidad es la estrategia más sencilla para reescalar una imagen. El primer paso a realizar es crear un espacio nuevo que tenga las dimensiones que se esperan de la imagen resultado, generando píxeles vacíos entre los píxeles ya conocidos. Los pixeles que ya conocíamos de la imagen original se colocan en las coordenadas que les corresponden de la nueva imagen, dejando sin valor los píxeles nuevos resultado de aplicar este método de reescalado. Los píxeles de los que desconozcamos su valor se les asigna el valor del píxel de la imagen original más cercano.

De esta forma estaremos haciendo un reescalado siguiendo de forma estricta el patrón de píxeles de la imagen original por lo que podríamos llegar a notar texturas extrañas en la imagen o bordes de sierra.

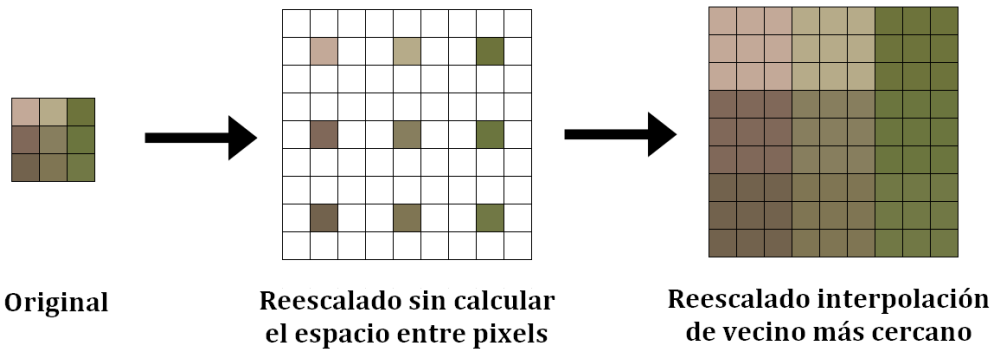


Ilustración 2 Funcionamiento del método de reescalado por proximidad

En la ilustración superior observamos la imagen original siendo reescalada un 900% de su tamaño original usando este método, donde antes había un único píxel ahora hay nueve.

Esta técnica es ideal para casos muy concretos como podría ser el reescalado de imágenes pixel art, pero en el resto de los casos no tendría sentido por la posibilidad de generar patrones cuadriculados en la imagen resultado.

Interpolación bilineal

Al igual que en el método de proximidad, creamos un espacio nuevo con las dimensiones de la imagen deseada y alojamos los pixeles de la imagen original en su coordenada correspondiente en el nuevo plano.

Para obtener el valor de los píxeles desconocidos se traza un gradiente, que es una interpolación matemática con la que se combinan proporcionalmente las intensidades de todos los pixeles cercanos que conocemos.

Con esta técnica acabaríamos rellenando todos los píxeles desconocidos de nuestra imagen, este proceso se conoce como reescalado bilineal por hacer interpolaciones bilineales en dos dimensiones. El resultado puede presentar pérdidas de detalles y ciertos desenfoques, pero el resultado general es notablemente mejor que el método de proximidad.

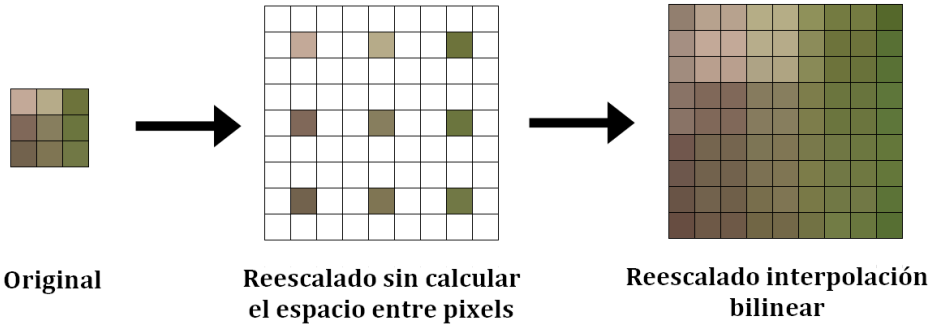


Ilustración 3 Funcionamiento del método de reescalado por interpolación bilineal

Apilamiento múltiple de imágenes

Otra técnica utilizada es el apilamiento múltiple de imágenes, este proceso es usado principalmente en el campo de la fotografía y se suele encontrar en el firmware propio de las cámaras o en el software de edición de fotografía como Photoshop.

En este método se realizan varias fotografías del mismo plano para obtener una imagen final de mayor resolución a la que puede capturar el dispositivo.

Las imágenes se toman a mano sin un trípode, para que el propio movimiento de la mano sirva como desplazamiento del sensor y permite a los píxeles tomar diferentes partes de la escena.

Con las tomas realizadas ya hechas, las apilamos, se realinean para que coincidan en cierta medida y se les aplica un filtro estadístico. El filtro utilizado comúnmente es el filtrado de mediana, que se encarga de reducir la variación de intensidad entre los píxeles vecinos. Tras la aplicación del filtrado se realizado un reescalado con métodos clásicos como la interpolación bilineal.

Este proceso da como resultado una imagen de mayor resolución, con mejor detalle y menor nivel de ruido del que tendría cualquiera de las tomas originales.

Aun así, este proceso incluye inconvenientes como la dificultad de aplicar este proceso a imágenes en movimiento debido ya que el movimiento de estos objetos se hará visible en la imagen resultado. Por otra parte, los aumentos de la resolución solo serán evidentes en las zonas de la imagen con más detalles. En consecuencia, esta técnica solo muestra rendimientos tangibles en escenas muy detalladas.

Comparativa del método bilineal y de proximidad

En la ilustración inferior se muestra una comparativa visual de la aplicación del método de proximidad e interpolación bilineal sobre una imagen, en la que se puede observar la diferencia perceptual entre ambas a primera vista. Donde el método de proximidad tiene una apariencia similar con la imagen de baja resolución, la única diferencia es la resolución de ambas. Mientras que el método de interpolación bilineal ofrece una mayor definición de los detalles acercándose a la imagen original, pero con desenfoques en ciertas partes de la imagen.

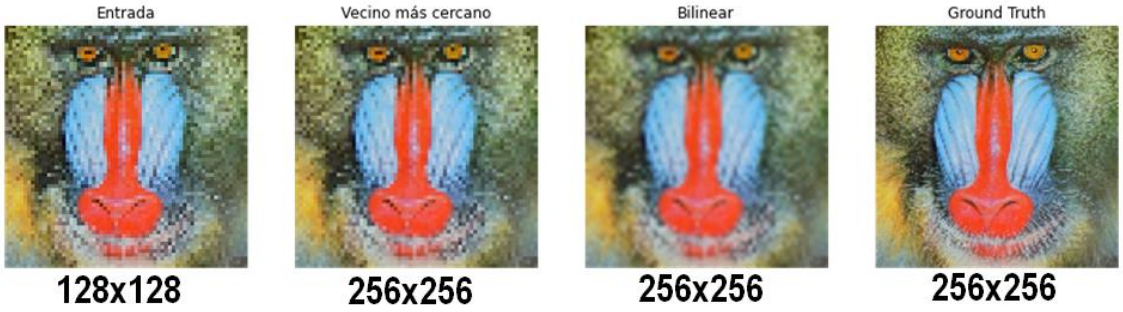


Ilustración 4 Comparativa visual entre métodos. Imagen propia.

Introducción a las redes neuronales

Machine Learning y Deep Learning, inteligencia artificial

¿Qué es el Machine Learning?

Históricamente el Machine Learning aparece en los años 80. Esta tecnología usaría algoritmos matemáticos para permitir a las máquinas aprender de forma analítica como resolver los distintos problemas mediante procesos de identificación, clasificación o predicción.

El Machine Learning o aprendizaje automático es una rama de la inteligencia artificial que permite a las máquinas aprender imitando la forma en la que aprenden los humanos, sin ser expresamente programadas para realizar esa función, usando algoritmos de regresión o árboles de decisión. El Machine Learning no solo es un conjunto de algoritmos, sino que también es un enfoque para abordar distintos problemas; es una vía para conseguir inteligencia artificial. Esta tecnología es indispensable para crear sistemas capaces de identificar patrones entre conjuntos de datos y realizar predicciones sobre los mismos.

Como resultado de hacer uso del aprendizaje automático se permite a las máquinas realizar tareas específicas de forma autónoma, siendo parte fundamental de algunos problemas como el Big Data.

¿Qué es el Deep Learning?

En el año 2011 aparecería una nueva rama del Machine Learning, el Deep Learning o aprendizaje profundo, que en concepto es muy similar al Machine Learning, pero usa algoritmos distintos.

Mientras que el Machine Learning trabaja con algoritmos de regresión o con árboles de decisión, el Deep Learning hace uso de redes neuronales.

El Deep Learning pertenece al aprendizaje automático y se considera como una evolución del Machine Learning. Está formado por un algoritmo automático que trata de imitar la percepción humana y las distintas conexiones entre neuronas de nuestro cerebro. Esto logra que el Deep Learning sea la técnica que más se acerca a la forma en la que aprendemos y pensamos los humanos.

Los métodos de Deep Learning en su mayoría hacen uso de redes neuronales por lo que un nombre común que se le da a esta tecnología es “redes neuronales profundas”.

¿Cuál es la diferencia entre ambos?

Tanto para el Deep Learning como el Machine Learning es esencial disponer de datos fiables de calidad para asegurar la eficacia de los resultados de estos dos tipos de inteligencia artificial,

Ambas tecnologías, Machine Learning y Deep Learning, pueden aprender de forma supervisada o no supervisada y tratan de imitar la forma en la que aprende el cerebro humano, pero tienen una principal diferencia entre ellas y es los algoritmos que usan. Por un lado, el coste computacional de los algoritmos de Deep Learning es superior a los de Machine Learning, mientras que el aprendizaje del Deep Learning suele requerir menos intervenciones.

Mientras que el Deep Learning tiene más similitudes con el aprendizaje humano por su funcionamiento con redes neuronales, el Machine Learning hace uso mayoritariamente de árboles de decisión. Resultando que el Deep Learning está más evolucionado por la tecnología que usa.

Tipos de aprendizaje

Existen diferentes tipos de aprendizajes usados en el Machine Learning y Deep Learning. A continuación, se explican brevemente algunos de ellos.

Aprendizaje supervisado

Este proceso de aprendizaje se realiza en un entrenamiento controlado en el que se utiliza un conjunto de datos etiquetados para entrenar el algoritmo. Al usar un conjunto etiquetado, se conocen los valores objetivo que debe alcanzar el algoritmo.

El aprendizaje supervisado se puede dividir en:

- Aprendizaje por corrección de error: Se ajustan los pesos de las conexiones de la red en función del error calculado, siendo este error la diferencia entre los valores esperados y la predicción obtenida. El algoritmo más común que usa este tipo de aprendizaje es la regla de aprendizaje del perceptrón simple, y se usa en las redes monocapa o de perceptrón simple.
- Aprendizaje estocástico: Este aprendizaje se caracteriza por la introducción de cambios aleatorios sobre los pesos de la red, evaluando su efecto a partir de la salida que deseas y de una distribución de probabilidad.

Aprendizaje no supervisado o auto supervisado

Este tipo de aprendizaje se caracteriza porque no utiliza influencia externa para ajustar los pesos de la red. El set de datos del que dispone no es etiquetado por lo que no sabe los valores de salida deseados a priori. El aprendizaje no supervisado trata los objetos de entrada como un conjunto de variables aleatorias.

Durante el aprendizaje, el algoritmo no tiene conocimientos para etiquetar los datos de entrada, pero sí puede agrupar estos datos según las características y regularidades que presenten los mismos. Este proceso lleva al algoritmo a encontrar patrones y estructuras en los datos no etiquetados, obteniendo como resultado del aprendizaje un análisis de los datos de entrada.

Qué es una red neuronal

Una red neuronal es un modelo computacional evolucionado a través de distintos avances científicos registrados a lo largo de la historia de la computación.

Este modelo trata de emular de forma simplificada la manera en la que el cerebro humano procesa la distinta información de entrada recibida. Se compone de un gran número de unidades de procesamiento que funcionan de forma simultánea, similares a una abstracción de las neuronas cerebrales. Las

abstracciones de las neuronas se denominan neuronas artificiales y se organizan en capas que componen la red neuronal.

Las redes neuronales son capaces de realizar observaciones generales sobre un conjunto de datos no estructurados y tomar decisiones inteligentes basadas en los resultados de las observaciones.

Estructura de la red neuronal

Por regla general una red neuronal se compone de tres partes principales:

- Una capa de entrada: las neuronas de esta capa representan los campos de entrada del modelo.
- Una o varias capas ocultas: este tipo de capa no tiene una conexión directa con el entorno, puede estar conectada a otras capas ocultas y a las capas de entrada o salida. El nombre de este tipo de capa se debe a que no es observable por lo que desconocemos tanto sus valores de salida como de entrada.
- Una capa de salida: formada por una o varias neuronas que representan el campo o campos de destino de la red.

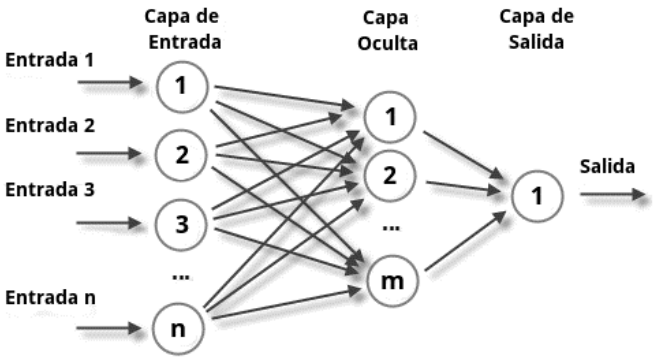
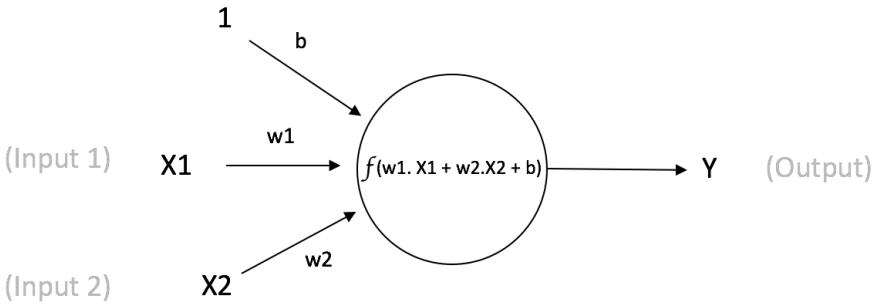


Ilustración 5 Estructura básica de una red de tipo perceptrón simple que muestra las capas de entrada, ocultas y de salida. Imagen obtenida de “Perceptrón Multicapa”, Wikipedia

Los datos de entrada del modelo se recogen en la primera capa, la capa de entrada, y se comienzan a propagar desde cada neurona hasta cada una de la siguiente capa, pasando por las capas ocultas si las hubiese, hasta alcanzar la capa de salida. Una vez la información ha alcanzado la capa de salida se envía el resultado del modelo al entorno.

Cada conexión de neuronas del modelo contiene un valor numérico modificable llamado peso, cuyo peso inicial normalmente es aleatorio, con el cual se modifica el valor de la entrada recibida. Los pesos con valor positivo representarían una conexión excitatoria mientras que los negativos serían una conexión inhibidora.



Output of neuron = Y= f(w1. X1 + w2.X2 + b)

Ilustración 6 Diagrama de una neurona, donde se muestran las entradas X1 y X2 con los pesos w1 y w2 respectivamente, junto a un sesgo b. La imagen ha sido obtenida del blog TechnologiesRunning

Gradiente

El gradiente es la generalización del conjunto de todas las derivadas parciales de una función. Generalmente en redes neuronales el gradiente de interés se obtiene de la función de coste o función de pérdida.

Gracias al gradiente podemos saber cómo se han de ajustar los distintos parámetros de la red para disminuir su desviación a la salida.

Existe un problema llamado desvanecimiento del gradiente. En este problema el gradiente se va desvaneciendo a valores cada vez más pequeños, impidiendo que cambie su valor. En el peor de los casos este problema conlleva que la red no pueda continuar su entrenamiento.

Función de activación

Las capas de la red poseen una función de activación que devuelve un valor de salida generado por la neurona a partir de un valor de entrada o un conjunto de valores de entrada. Podría ser definida como la manera de transmitir la información a través de las conexiones de salida de la neurona.

Las redes neuronales buscan cada vez resolver problemas más complejos por lo que generalmente las funciones de activación nos ayudarán convertir a los modelos neuronales en no lineales.

Existen diversos dos tipos de funciones de activación [27]: lineales y no lineales. Generalmente se hace uso de funciones no lineales porque permiten al modelo adaptarse para trabajar con una mayor cantidad de datos.

Entre las funciones de activación no lineales las más usadas son: la función de Heaviside, la función sigmoide y la función rectificadora.

Función escalón de Heaviside

Es una función discontinua cuyo valor es 1 para cualquier argumento positivo incluido 0, y 0 cuando el argumento es negativo.

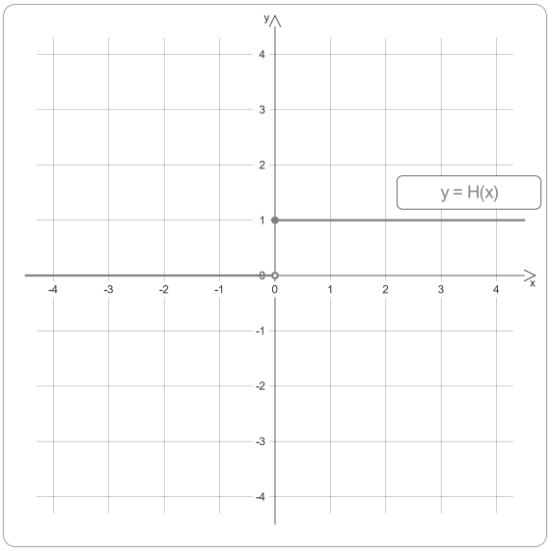


Ilustración 7 Gráfica de la función de Heaviside. Obtenida de “Función escalón Heaviside”, Wikipedia

Función de activación sigmoide

La función sigmoide fue la base de las redes neuronales durante muchos años, pero recientemente han ido perdiendo popularidad. Esta función se define como:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (1)$$

La constante “e” es el número de Euler y la constante “x” el valor de entrada. Definimos un ejemplo donde el valor de entrada de función es la suma ponderada de los pesos de las neuronas de la capa. w_iy x_i representan los pesos y las entradas numéricas respectivamente. La variable "n” representa el número de entradas y “b” el valor de sesgo.

$$z = b + \sum_{i=0}^{n} w_i x_i \quad (2)$$

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad (3)$$

La siguiente gráfica es una representación de la función sigmoide en la que podemos apreciar que actúa como una especie de normalizador, ya que comprime la salida de los datos entre los valores 0 y 1.

A valores positivos muy grandes de "z", el componente e^{-z} va a reducir el valor hacia 0, por lo que $\sigma(z)$ se aproxima a 1. Para los valores negativos de "z" muy grandes ocurre lo contrario, e^{-z} va a crecer exponencialmente y $\sigma(z)$ se aproximará a 0.

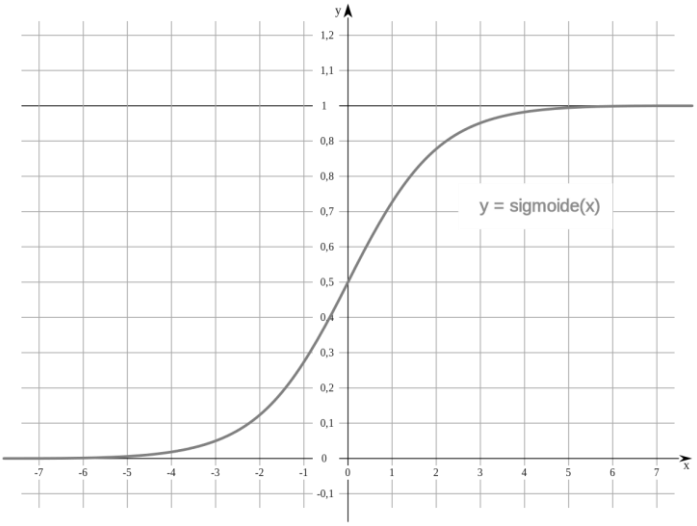


Ilustración 8 Gráfica de la función sigmoide. Obtenida de "Función Sigmoide", Wikipedia

Función rectificadora ReLU

Actualmente es el tipo de función de activación más utilizada, sustituyendo a las funciones sigmoides debido a que obtienen mejores resultados en aplicaciones generales. Esto se debe a que este nuevo tipo de funciones soluciona de mejor manera el problema de desvanecimiento de gradiente.

Se define como:

$$R(z) = max(0, z) \tag{4}$$

El funcionamiento es muy sencillo, las funciones ReLU permiten cualquier valor positivo como salida y los valores negativos se establecen a 0.

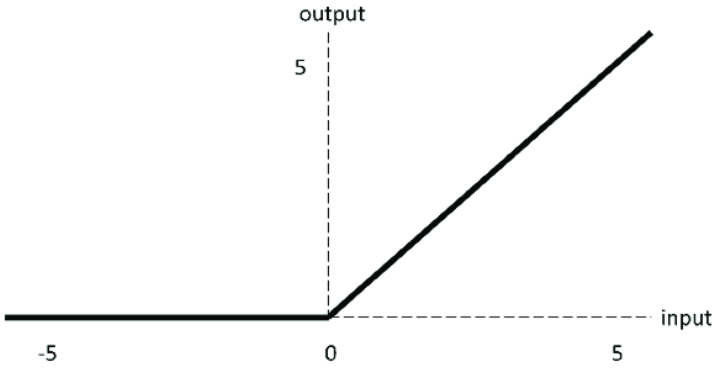


Ilustración 9 Gráfica de la función de activación ReLU. Obtenida del artículo “Multi-Classification of Brain Tumor Images Using Deep Neural Network” del portal *ResearchGate*

Este tipo de función tiene un problema que ocurre cuando muchas de las neuronas con esta función presentan valores de salida de 0. Esta situación lleva a que la mayoría de las entradas de estas neuronas estén en el rango negativo y pueda causar la muerte de la red, haciendo que las funciones rectificadoras actúen como funciones constantes. A este problema se le conoce como ”Dying ReLU” [41].

Existe una variación de este tipo de función de activación que soluciona el problema de las “Dying ReLU”, llamada Leaky ReLU [45]. Este tipo de función actúa de forma similar a la función ReLU estándar, pero incluye una pequeña pendiente negativa para los valores de entrada negativos. Se podría definir de la siguiente forma, siendo la constante “a” la pendiente por la que se multiplican los valores negativos:

$$f(x) = \begin{cases} ax & si \quad x < 0 \\ max(0,x) & si \quad x \geq 0 \end{cases} \tag{5}$$

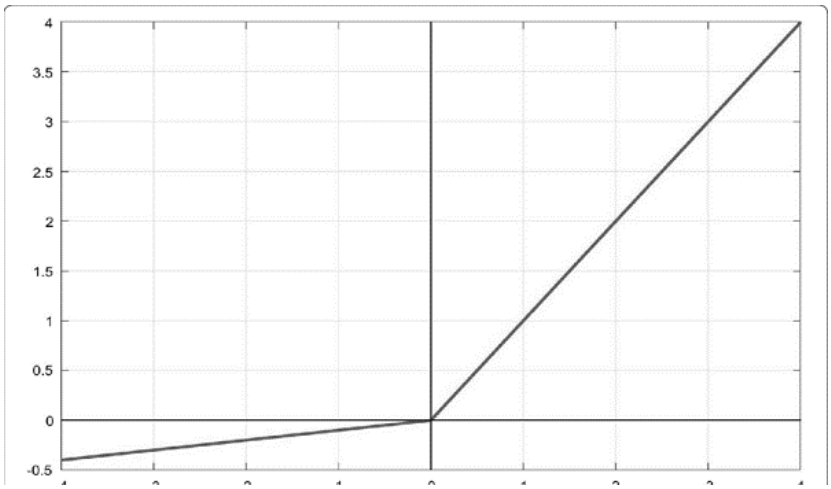


Ilustración 10 Gráfica de la función de activación Leaky ReLU. Obtenida del artículo “Leaky ReLU Activation Function” del portal *VidyAsheela*

Tipos de redes neuronales

Existen varios tipos de redes neuronales en el campo de la inteligencia artificial, abarcando desde las estructuras más simples hasta arquitecturas complejas de miles de capas de neuronas conectadas entre sí.

Monocapa o perceptrón simple

Este tipo de red está compuesta por una sola neurona de salida con una serie de diferentes entradas, es la red neuronal más simple. La única capa de esta red realiza diferentes cálculos basados en los valores de las entradas, obteniendo un resultado como salida.

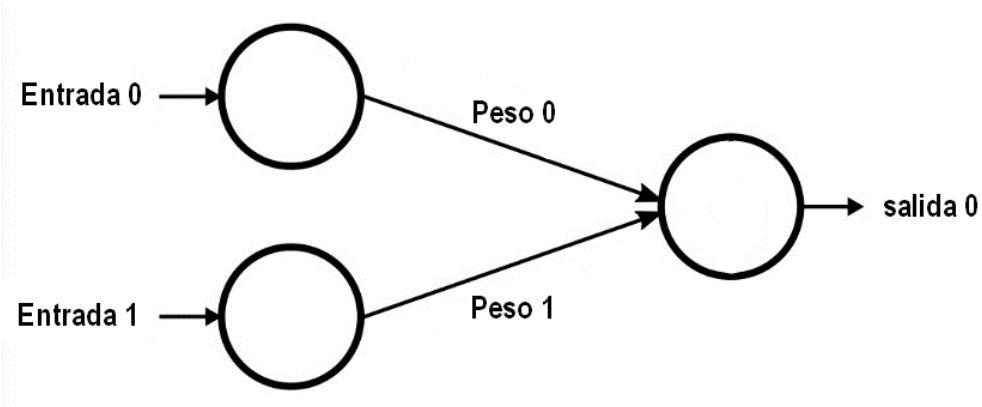


Ilustración 11 Diagrama de una red mono capa. Obtenida del artículo “Single Layer Network for Classification” en Kaggle.

Este tipo de redes están limitadas a problemas de clasificación lineal en la que los valores observados pueden ser divididos por una recta. El principal objetivo de este tipo de redes es la clasificación estadística para reconocer a qué clase de salida pertenece el conjunto de entrada, el clasificador que usan generalmente es la regresión logística simple. Algunos ejemplos de problemas linealmente separables en los que se podría aplicar una regresión simple son las funciones AND y OR.

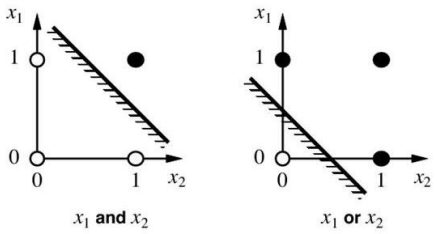


Ilustración 12 Representación de las funciones AND y OR linealmente separables. Obtenida del artículo “Radial Basis Functions Neural Networks” del portal TowardsDataScience

Multicapa o perceptrón multicapa

Este tipo de red neuronal es una generalización del tipo de red de perceptrón simple. Este tipo de redes se componen de una capa de salida, una capa de entrada y entre ambas un conjunto de capas ocultas, o capas intermedias. El número de conexiones que presente la red puede hacer que esta esté total o parcialmente conectada. Con las redes monocapa solo éramos capaces de representar problemas lineales, pero con este tipo de redes podemos abarcar problemas no lineales como el problema de resolver la función XOR.

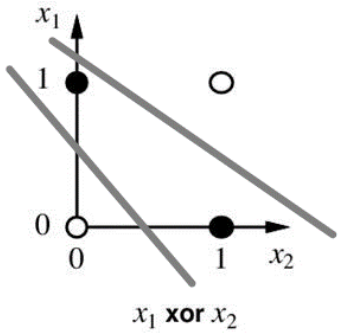


Ilustración 13 Representación de la función XOR, separable no linealmente. Obtenida del artículo “Radial Basis Functions Neural Networks” del portal TowardsDataScience.

El algoritmo de aprendizaje que se suele usar es el algoritmo de propagación hacia atrás o back propagation, con el que los errores obtenidos en las salidas del perceptrón se van propagando hacia capas anteriores. Este algoritmo tiene como objetivo modificar los pesos de la red de forma que el valor estimado de la red se asemeja cada vez más al real.

El perceptrón multicapa puede ser usado tanto en problemas de clasificación y regresión, es lo suficientemente flexible como para poder adaptarse a diferentes tipos de datos. Esta flexibilidad le permite resolver distintos problemas que impliquen valores de imágenes, palabras o numéricos.

Redes neuronales recurrentes RNN

Las redes neuronales recurrentes no tienen una estructura de capas, sino que permiten conexiones arbitrarias entre neuronas e incluso crean bucles. La creación de estos bucles crea temporalidad y le otorgan memoria a la red.

Este tipo de red toma como entrada un conjunto de datos que es dividido en una secuencia de datos; el primer dato de la secuencia es introducido a la red como entrada y se obtiene una salida correspondiente. Para la segunda salida se va a introducir como entrada el segundo dato de la secuencia más la salida de la anterior, haciendo que a partir de la primera secuencia generada todas

las entradas de la red sean una concatenación de la nueva entrada correspondiente junto a la salida de su secuencia anterior.

El problema principal de este tipo de redes es que cuanto mayor sea la secuencia de datos, menos valor tienen los primeros pesos de la secuencia introducida a la red durante el entrenamiento. Este problema es similar a la "falta de memoria" humana.

Redes Transformers

Tienen su origen reciente en 2017, bajo el estudio del articulo "Attention Is All You Need" [4]. Desbancan a las redes recurrentes en procesamiento de secuencias ya que solucionan el problema de "falta de memoria" gracias a que poseen una memoria más duradera a largo plazo [31]. Este problema se soluciona introduciendo un nuevo componente llamado "mecanismos de atención", que logra procesar la secuencia de datos en paralelo y no de forma secuencial como ocurre en las redes recurrentes.

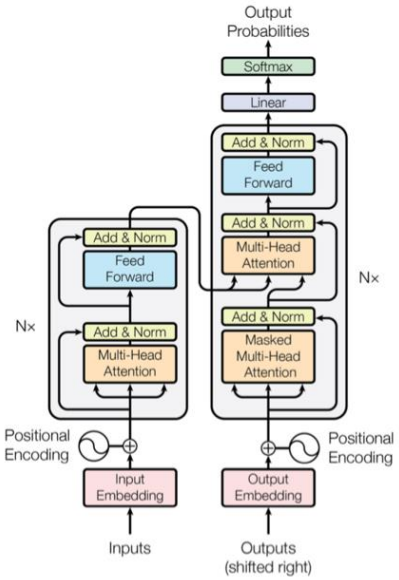


Ilustración 14 Arquitectura de una red transformer. Obtenida del artículo "Attention Is All You Need" [4].

Existen ya grandes proyectos que hacen uso de este tipo de redes, algunos de los proyectos más influyentes son:

- GPT-3 [5], para la generación de texto realista.
- AutoPilot de Tesla [6], funcionando como piloto automático de conducción de los automóviles de la compañía Tesla.
- AlphaFold 2 [7], para realizar predicciones de la estructura de secuencias de proteínas.

Redes convolucionales

Las redes neuronales convolucionales son un tipo de red neuronal que se caracteriza por aplicar un tipo de capa en la que se realiza una operación matemática como convolución. Las neuronas artificiales de esta red son una abstracción de las neuronas de la corteza visual del cerebro humano, cuya principal tarea será el reconocimiento de patrones en imágenes de la misma forma que lo haría el cerebro humano. Esta red es una variación del perceptrón multicapa, pero es más efectiva para tareas de visión artificial como la segmentación y clasificación de imágenes. La razón por la que son más efectivas en tareas de visión artificial es debido a que su aplicación se realiza sobre matrices bidimensionales.

La aplicación de estar redes abarca principalmente el campo de extracción de patrones en imágenes y vídeos de forma similar a como lo haría la mente humana. Pudiendo realizar tareas de reconocimiento de imágenes como la detección de rostros en una imagen o el análisis predictivo de una imagen de un diagnóstico médico. Las redes neuronales convolucionales son muy potentes en el campo del análisis de imágenes, debido a que son capaces de detectar características simples como la detección de bordes o líneas, y componerlas en características más complejas hasta detectar lo que se busca.

La principal diferencia de la red neuronal convolucional con el perceptrón multicapa es que cada neurona no se une con todas y cada una de las neuronas de las capas siguientes, sino que solo se une con un subgrupo de ellas, se especializa. Con esta especialización, la red es capaz de reducir el número de neuronas necesarias y la complejidad computacional necesaria para su ejecución.

La principal ventaja de este tipo de redes es que cada parte de la red se entrena para realizar una tarea concreta, cada capa convolucional se encarga de obtener distintos patrones y características de la imagen. Esto disminuye el número de capas ocultas presentes y reduce significativamente el tiempo de entrenamiento.

En las capas de convolución de la red se realiza una operación de producto y suma entre la capa inicial y los “**n**” filtros, también llamados kernel, que como resultado genera el mapa de características de la imagen. Las características extraídas corresponden a todas las posiciones posibles del filtro sobre la imagen original. Este proceso es conocido como convolución.

Los filtros de la capa consisten en una matriz que se encarga de realizar un barrido sobre la imagen, realizando el proceso de convolución. Este barrido generalmente se realiza desde la esquina superior izquierda de la imagen

hasta la esquina inferior derecha. Según cómo estén configurados los filtros se podrán detectar distintos patrones en la imagen y nos permiten extraer las características de esta. Cada uno de estos filtros no necesita de una configuración manual, durante el entrenamiento son ajustados de forma automática por la red para detectar cada vez patrones y características más complejas de la imagen.

En la siguiente ilustración se puede observar el proceso de convolución en su primera iteración. A la izquierda se presenta la imagen original con la ubicación del filtro marcada en rojo, y a su derecha el valor resultante de la operación con el filtro. La imagen resultante del proceso de convolución tendrá un tamaño de $(m-u+1)x(n-v+1)$donde "m" y "u" son el número de filas de la imagen de entrada y el filtro respectivamente, mientras que "u" y "v" son el número de columnas de la entrada y el filtro. Aplicando esta fórmula al ejemplo de la ilustración inferior podemos comprobar que se cumple ya que $(6-3+1)x(6-3+1)=4x4$.

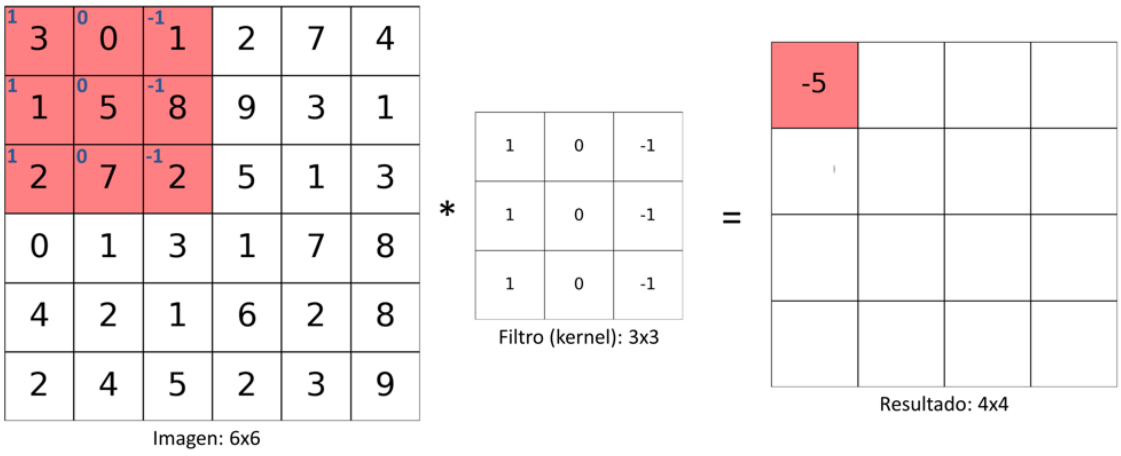


Ilustración 15 Representación de la primera iteración de una convolución. Obtenida del artículo "La Convolución en las Redes Convolucionales" del portal CodificandoBits

Existen dos parámetros importantes en las capas convolucionales que influencian en el tamaño de la imagen resultante de la convolución, llamados "strides" o saltos y "padding".

Como se ha mencionado previamente, las convoluciones generan una imagen de menor tamaño que la de entrada. Con el uso del padding podemos hacer que la imagen generada sea del mismo tamaño que la original, este proceso se realiza añadiendo píxeles de valor 0 en los bordes de la imagen original. Cuando se realice el barrido de la imagen durante el proceso de convolución se tendrán en cuenta estos píxeles añadidos, teniendo como resultado de la convolución una imagen del mismo tamaño que la original.

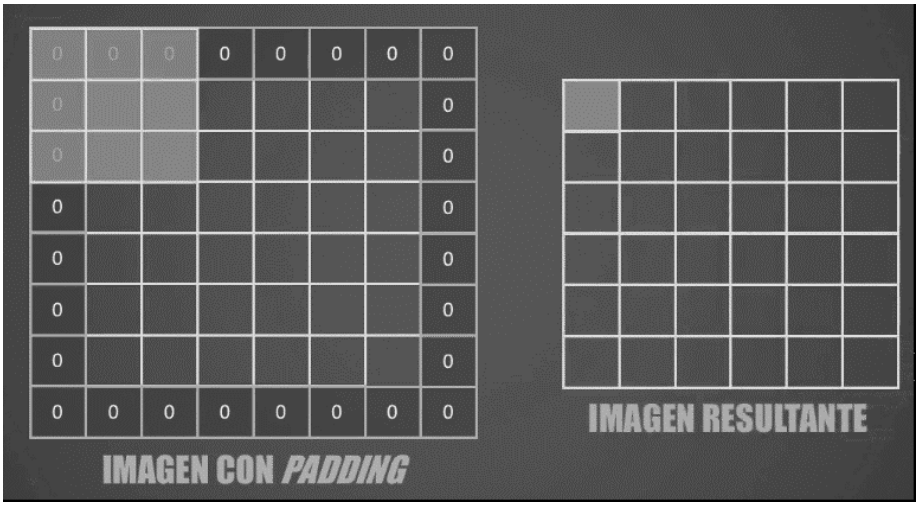


Ilustración 16 Simulación de la convolución de una imagen con padding añadido. Obtenida del artículo “Padding, strides, max-pooling y stacking en las Redes Convolucionales” del portal CodificandoBits.

El otro parámetro mencionado, los strides, influye en el barrido que realiza el kernel durante la convolución. Según el valor “x” que le asignemos, el kernel se desplazará “x” pixeles hacia la derecha o hacia abajo en cada iteración. Si usamos un valor mayor que 1 con este parámetro, conseguimos reducir el tamaño de la imagen resultante respecto a la convolución original.

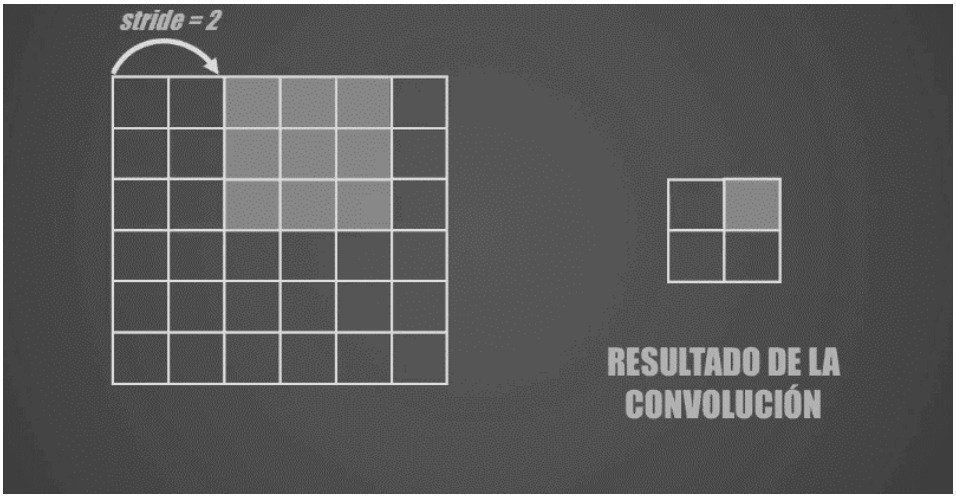


Ilustración 17 Simulación de la convolución de una imagen con saltos de 2 píxeles añadidos. Obtenida del artículo “Padding, strides, max-pooling y stacking en las Redes Convolucionales” del portal CodificandoBits.

En las redes convolucionales se aplica una técnica de regularización conocida como “Dropout” o dilución, con el objetivo de reducir el sobreajuste del modelo. Este método desactiva una cantidad aleatoria de diferentes neuronas en cada iteración, al desactivar estas neuronas no formarán parte de los procesos de propagación hacia adelante ni propagación hacia atrás.

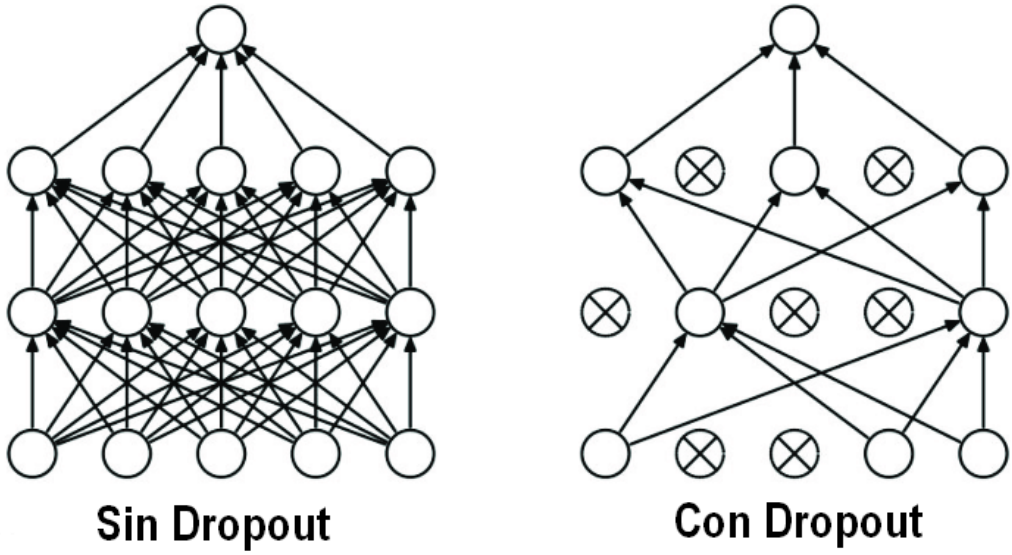


Ilustración 18 Representación de la desactivación de neuronas con Dropout. Obtenida del artículo “Ranking to Learn and Learning to Rank” del portal ResearchGate

Como resultado conseguimos que las neuronas cercanas a las desactivadas no dependan tanto de ellas, las neuronas aprenderán a trabajar mejor de forma solitaria y con menor dependencia de las neuronas cercanas.

La función de activación recomendada para las capas convolucionales es la conocida como ReLU, también se hace uso de la función Leaky ReLU.

Existe una variación de este tipo de redes que encarga de realizar el mismo proceso, pero de forma transpuesta. Estas redes son llamadas redes neuronales deconvolucionales y presentan capas de convolución transpuesta. Este tipo de capas se encargan del proceso de deconvolución, siendo la restauración de datos a partir de los datos de salida de las capas convoluciones.

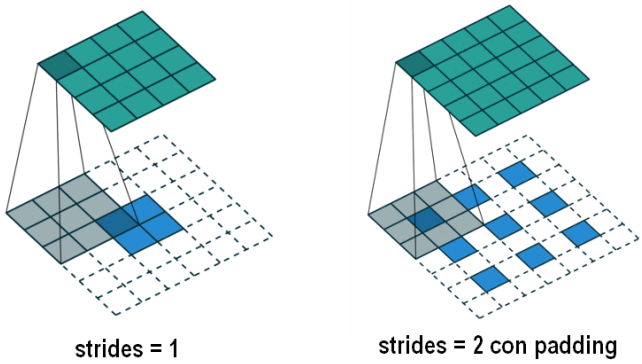


Ilustración 19 Representación de cómo actúan las capas de convolución transpuestas. Obtenida del artículo "A guide to convolution arithmetic for Deep Learning”

A diferencia de las capas convolucionales usuales, el parámetro strides sirve para aumentar el tamaño de la imagen resultante. Durante el entrenamiento. el modelo neuronal aprenderá a interpretar y rellenar los huecos creados por este tipo de capas en la imagen.

Este tipo de redes son clave en la solución de algunos problemas como el problema de superresolución, permitiendo la detección de patrones en la imagen y posteriormente aumentar su resolución.

Redes generativas adversarias

Los modelos generativos adversarios están formados comúnmente por dos redes neuronales convolucionales denominadas como generador y discriminador. Este tipo de redes originalmente fueron planteadas de forma que el generador y el discriminador eran redes multicapa [42], a día de hoy no se hace uso de esta arquitectura. La explicación de la red se realiza con el enfoque del uso de redes convolucionales.

Uno de los primeros proyectos en los que aparecieron las redes generativas adversativas convolucionales fue en la generación de dormitorios no reales [28].

Ilustración 20 Compilación de dormitorios no reales generados con redes GAN. Obtenida del artículo "Unsupervised representation Learning with deep convolutional generative adversarial networks" [28].

Normalmente los generadores son redes neuronales deconvolucionales y los discriminadores convolucionales. El modelo se denomina como adversario porque las dos redes que lo componen “compiten” entre ellas a lo largo del entrenamiento.

Vamos a profundizar en este concepto de competición a través de un ejemplo aplicado a este trabajo de investigación, la generación de fotos de superresolución.

El generador se encarga de producir imágenes a superresolución que parezcan auténticas mientras que el discriminador tiene la tarea de predecir si esta

imagen reescalada es auténtica o no. A partir de la predicción del discriminador se calcularán los valores de pérdida del generador y discriminador.

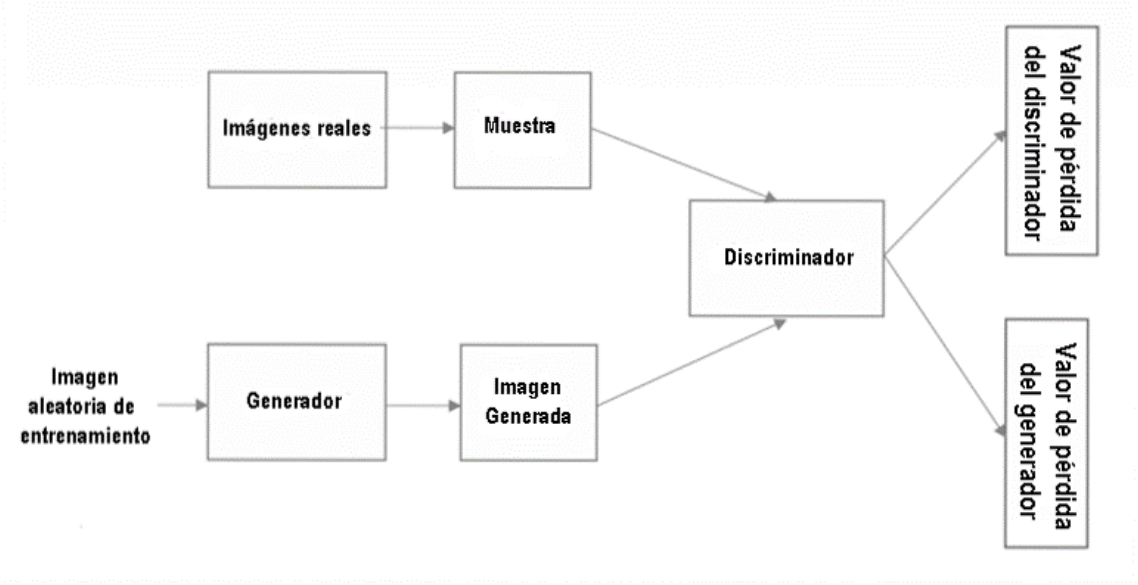


Ilustración 21 Arquitectura básica de una red generativa adversaria.

Para poder entrenar esta arquitectura de dos redes neuronales necesitamos tener un set de datos etiquetado que va a alimentar ambos modelos.

En ambas redes se usa el mismo set de datos etiquetado, en el que los valores de objetivo son imágenes reales y los datos de entrada son esas mismas imágenes con menor resolución. Al generador le vamos a administrar como entrada las imágenes del set de datos real con una resolución menor a la original. Por el otro lado se introducen como entradas del discriminador las imágenes reales del set de datos y las imágenes alucinadas por el generador.

La clave de las redes generativas adversarias es el entrenamiento indirecto del generador realizado a través del discriminador. Esta red es capaz de analizar cuán real es la entrada que recibe de la otra red de la arquitectura, el generador. El discriminador, teniendo como entradas la imagen alucinada por el generador y la imagen real, tendrá que decidir si la imagen alucinada es real o no. Resultado de este proceso genera como salida los datos necesarios para poder calcular el valor de pérdida del generador y discriminador, necesarios para el cálculo de los gradientes.

Esta curiosa combinación de redes convolucionales va a generar una competición entre el generador y el discriminador, en la que ambas compiten para ver cuál consigue hacer mejor su trabajo. El objetivo principal es que el generador genere imágenes que sean catalogadas como reales por el discriminador.

El generador de la red generativa adversaria suele presentar en su estructura conexiones de salto entre los bloques residuales que componen su estructura [47].

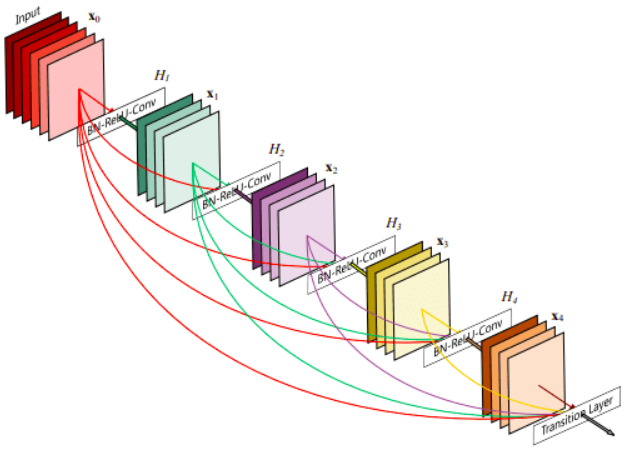


Ilustración 22 Representación de conexiones de salto que conectan bloques de capas convolucionales de la red. Obtenida del artículo “Intuitive Explanation of Skip Connections” del portal AI Summer.

Las conexiones de salto existentes entre los bloques de capas se utilizan en la salida de cada bloque para permitir ignorar capas posteriores e introducir los datos de entrada en capas posteriores de la red. El uso de estas conexiones es uno de los factores más importantes en los bloques residuales, con su uso se permiten solventar problemas de degradación de la red. Los problemas de degradación son ocasionados por el sobreajuste y la profundidad del modelo.

En un inicio es muy sencillo para el discriminador acertar qué fotos son reales y cuáles son las alucinadas por el generador. Sin embargo, a medida que el generador mejore durante el entrenamiento, la tarea del discriminador será más complicada haciendo que tenga que realizar ajustes en sus pesos. Por lo tanto, ambas redes evolucionan durante el entrenamiento.

Durante el entrenamiento, el discriminador ayuda al generador a crear cada vez mejores imágenes, ambas redes van mejorando simultáneamente hasta que llega un punto teórico en el que las imágenes creadas por el generador son percibidas por los humanos como si fuesen imágenes indistinguibles de las reales.

Existen proyectos que hacen uso de este tipo de red como StyleGAN [29] o NVIDIA Maxine [30].

- StyleGAN es un modelo utilizado para la generación de rostros no reales. La red de este modelo se ha encargado de aprender los distintos patrones de “estilos” que existen las imágenes de los rostros observados,

siendo los estilos definidos como aquellos detalles que son el pelo, la forma de la cara, expresiones fáciles y otros rasgos de un rostro. Al generar un nuevo rostro la red se apoya en estos patrones, tomando dos fotografías reales y combinando los estilos de ambas para generar un rostro no real.

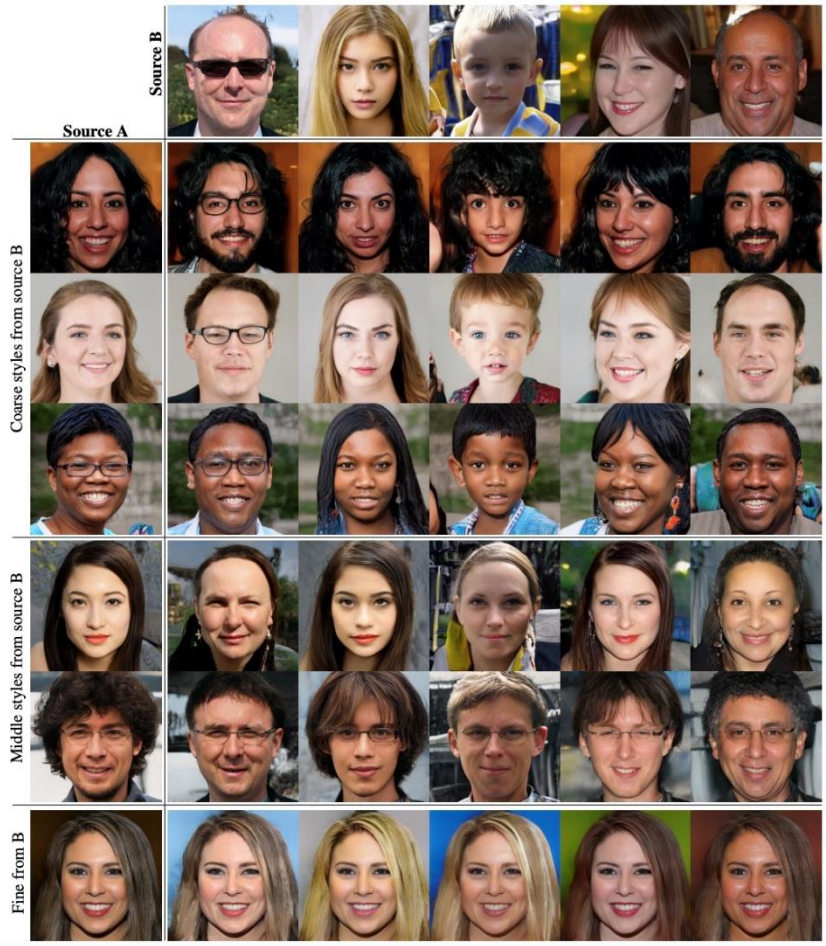


Ilustración 23 Ejemplo de combinación de los estilos de dos imágenes. Obtenida del artículo original de StyleGAN [29].

Un ejemplo en vivo de esta tecnología se puede ver en la página web https://thispersondoesnotexist.com.

- NVIDIA Maxine es una tecnología que funciona como asistente de video llamadas y promete reducir drásticamente el consumo de bando de ancha usando las redes generativas adversarias. Las imágenes de la videollamada del emisor se capturan a una baja resolución y se envían al receptor, una vez llegado al ordenador receptor se procesa esta imagen con una red generativa adversaria para aumentar la resolución de la

imagen obtenida. Este proceso permite que se realicen videollamadas a una buena resolución sin necesidad de tener un buen bando de ancha.

Las redes generativas adversarias tienen un gran potencial en el campo de procesado de imágenes por lo que en los últimos años han surgido varios modelos con el propósito de resolver el problema de superresolución. Estos modelos son conocidos con el nombre de ESRGAN [8] y SRGAN [9].

SRGAN

El modelo de SRGAN es una red generativa adversaria que tiene el propósito de aplicar superresolución a una imagen de entrada.

Este modelo se compone de dos bloques principales, el generador y el discriminador.

El generador se encarga de alucinar una imagen de alta resolución basándose en la imagen de baja resolución introducida como entrada. Por otro lado, el discriminador se encargará de distinguir las imágenes reales de las obtenidas de la salida del generador.

Utiliza para el generador una función de pérdida compuesta por un valor de pérdida adversario y un valor de pérdida de contenido, este último valor se apoya en el uso de un modelo VGG.

Normalmente se hace uso de un valor de pérdida de contenido basado en el error cuadrático medio del propio generador, pero la novedad que introdujo este modelo fue el calcular el valor de pérdida de contenido usando el mapa de características obtenido al alimentar el modelo VGG. Se añaden como entradas del modelo VGG la imagen de referencia y la imagen generada, generando dos salidas. Se realiza el cálculo del error cuadrático medio de las dos salidas obtenidas para conseguir el valor de pérdida mencionado.

La red hace uso de un discriminador entrenado para diferenciar entre imágenes alucinadas por el generador e imágenes reales.

El valor de pérdida total del generador se calcula como:

$$l^{SR} = l_{VGG}^{SR} + 10^{-3} l_{Gen}^{SR} \quad (6)$$

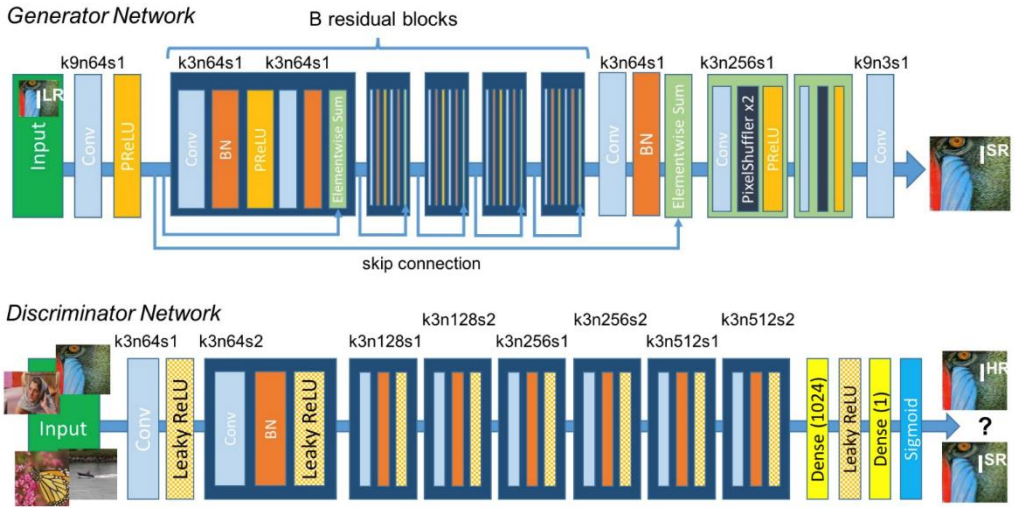


Ilustración 24 Arquitectura del generador y discriminador del modelo SRGAN. Obtenida del artículo original de SRGAN[9].

En el caso del modelo de SRGAN existe también un componente importante del generador basado en combinación de múltiples bloques residuales, que poseen capas de normalización por lotes. Este tipo de capas son usadas de forma general en las redes neuronales para la normalización de los valores de la red y obtener un entrenamiento más rápido [33].

Ilustración 25 Bloque residual con capas de normalización de lotes, original del artículo del modelo SRGAN [9].

ESRGAN

El modelo ESRGAN sigue las mismas bases que el modelo SRGAN para resolver el problema de superresolución, pero incluye algunas mejoras respecto a este.

Una de las principales diferencias es la arquitectura del generador, que está compuesto por bloques RRDB (residual in residual dense blocks) en vez de simples bloques residuales. Este nuevo tipo de bloques encadena múltiples bloques residuales junto con una conexión de salto global. Además, los bloques residuales de los que se componen los RRDB carecen de capas de normalización por lotes debido a diferentes problemas que ocasionan durante el entrenamiento [34].

Existen diferentes problemas causados por el uso de las capas de normalización por lotes.

- El aumento en gran medida del coste computacional de la red.
- La existencia de diferencias notables entre los datos generados durante la inferencia y el entrenamiento. Existen discrepancias en los resultados de la inferencia respecto a los del entrenamiento debido a que al usar capas de normalización por lotes estamos introduciendo un nuevo hiperparámetro que es necesario afinar, el tamaño de los lotes.
- En la investigación del proyecto del modelo ESRGAN se realizó un estudio para demostrar que además de estos problemas, este tipo de capas generaban distintos artefactos de forma aleatorias en la imagen durante el entrenamiento. Este problema da lugar a que no se tenga un rendimiento estable durante el entrenamiento del modelo.

Otra de las mejoras presentadas por este modelo ESRGAN respecto a SRGAN, es el uso de un discriminador relativista.

El discriminador relativista posee una función de pérdida que calcula la diferencia relativa entre las imágenes. El discriminador se encargará de predecir la probabilidad de que la imagen real sea más realista que la imagen alucinada por el generador.

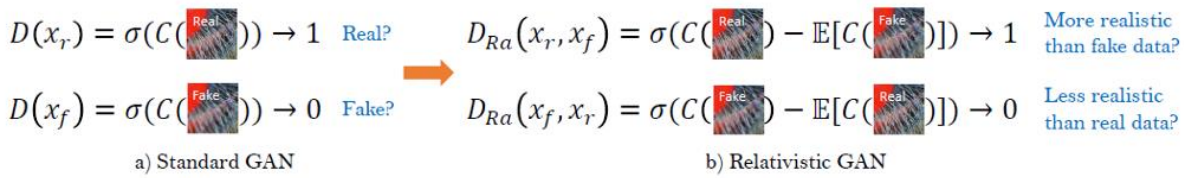


Ilustración 26 Comparativa entre la función de pérdida del discriminador estándar contra el discriminador relativista. Obtenida del artículo “An overview and Implementation of the methods of ESRGAN” del portal ResearchGate.

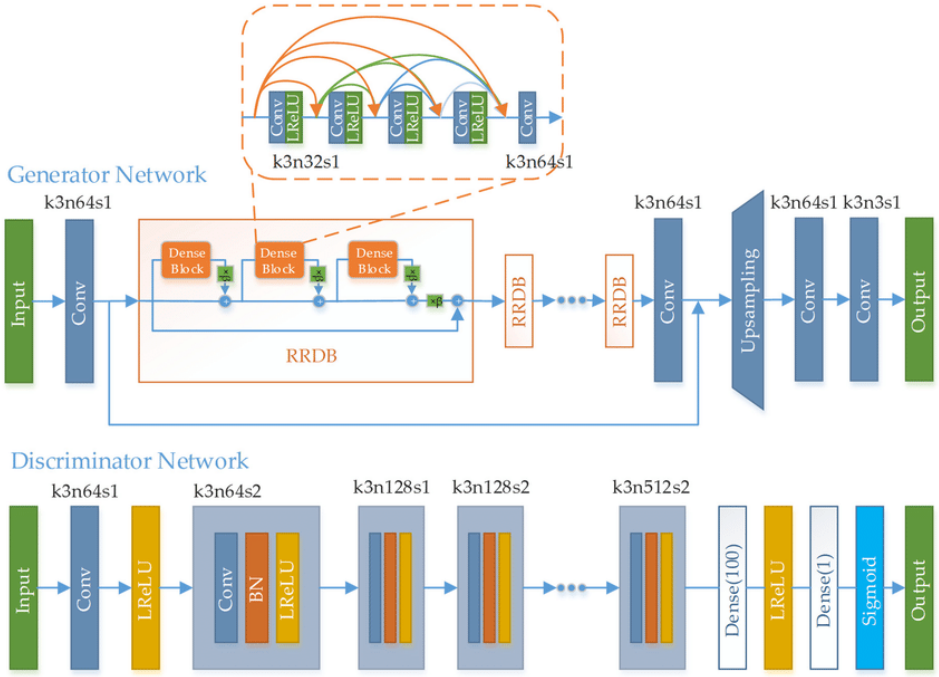


Ilustración 27 Arquitectura del generador y discriminador del modelo ESRGAN. Obtenida del artículo "Generative Adversarial Networks Capabilities for Super-Resolution Reconstruction of Weather Radar Echo Images" del portal ResearchGate.

Entrenamiento de la red neuronal

El entrenamiento de una red neuronal se basa en ajustar los pesos de cada una de las entradas de las neuronas que componen la red. Realizando este ajuste conseguimos adecuar las respuestas de la capa de salida para obtener los valores deseados de salida.

Una red puede ser entrenada para distintos propósitos, como podría ser la clasificación de imágenes o la generación automática de texto, pero todos los entrenamientos tienen la misma premisa: conseguir que la red sea capaz de generalizar. Si una red neuronal no es capaz de generalizar significa que no es capaz de inferir características y patrones a partir de los datos del entrenamiento, ni reconocer nuevos datos que no han sido observados previamente por la red. En estas condiciones la red simplemente está memorizando los datos del entrenamiento y sufre problemas de sobreajuste.

Sobreajuste y desajuste

El sobreajuste se da en las situaciones que la red neuronal se limita a memorizar únicamente los datos observados durante el entrenamiento, pero no es capaz de inferir las características y patrones generales de estos datos. La

red neuronal no es capaz de clasificar e identificar adecuadamente los datos observados como consecuencia del sobreajuste.

Por otro lado, el desajuste ocurre en casos que la red no ha sido entrenada durante el suficiente tiempo o cuando, al igual que el sobreajuste, el modelo neuronal ha memorizado el conjunto de datos de entrenamiento y no es capaz de generalizar con datos no observados en el entrenamiento.

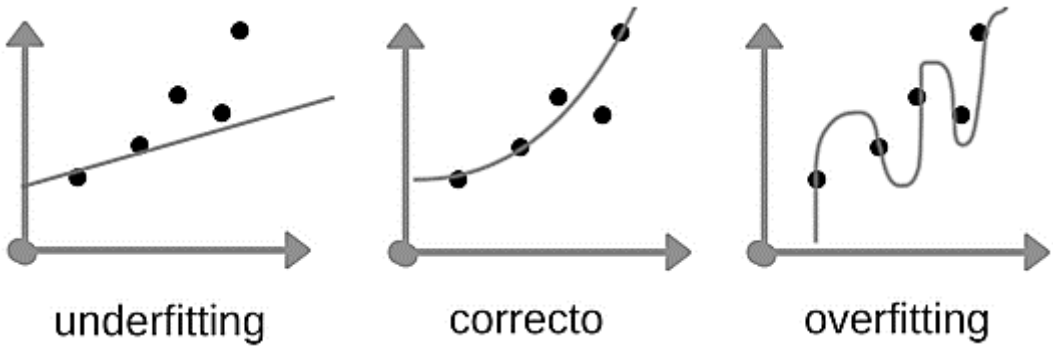


Ilustración 28 Gráficas de un set de valores que representan los problemas de sobreajuste y desajuste. Obtenida del artículo “Qué es overfitting y underfitting” del portal AprendeMachineLearning.

El principal problema de la falta de generalización es la incapacidad para detectar las características, pequeños matices, patrones y complejidad de los datos introducidos al modelo. Para prevenir el sobreajuste y desajuste podemos hacer uso de la validación cruzada como medida preventiva.

Validación cruzada

La validación cruzada es un método de evaluación utilizado en modelos neuronales para comprobar si las predicciones de la red son independientes de los datos de entrenamiento. Esta técnica nos permite detectar si existe sobreajuste en el modelo.

Función de perdida

Uno de los componentes importantes del proceso de entrenamiento de las redes neuronales son las funciones de pérdida [10,11]. Estas funciones evalúan la desviación entre los datos reales observados durante el entrenamiento y las predicciones de la red neuronal. El objetivo del entrenamiento es reducir el resultado de estas funciones ajustando los pesos de la red neuronal; cuanto menor sea el resultado obtenido de la función, más eficiente es la red neuronal.

En las redes neuronales se generan predicciones y se comparan con los valores reales, el resultado de esta comparativa es conocido como valor de pérdida. Cuanto más alto es el valor, peor el rendimiento de la red. Con las funciones de pérdida se calcula el valor de pérdida, que será usado en el cálculo de los gradientes necesarios para actualizar los pesos de la red neuronal.

Las funciones de pérdida se pueden dividir en dos tipos principalmente [32]: funciones de regresión y de clasificación.

Función de regresión

Las funciones de regresión tratan de predecir un valor cuantitativo basado en el valor de una o varias variables predictoras. La función de regresión más sencilla es la regresión lineal, aunque sea útil en muchas ocasiones no permite modelar relaciones complejas ni capturar relaciones no lineales sin transformar los valores de entrada. A su vez, la regresión lineal puede producir valores atípicos.

Existen dos valores de pérdida que se usan generalmente en las funciones de regresión:

- Error lineal: es el error local de la red y se calcula obteniendo la diferencia entre el valor de la predicción y del valor esperado.
- Error cuadrático medio: es el error global del aprendizaje. Este error representa cuán cerca está la recta de regresión de un conjunto de puntos. Esta función nos permite conocer con una comparación global el porcentaje de error de nuestras neuronas.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (7)$$

Función de clasificación

Las funciones de clasificación tratan de predecir si las muestras están sujetas a una clase o categoría de un grupo múltiple de estas.

Estos tipos de funciones se podrían dividir en dos grupos, funciones de clasificación binaria y funciones de clasificación multiclase.

- Las **funciones de clasificación binarias** son usadas en problemas que poseen solo dos clases para clasificar las salidas de la red. La entropía cruzada binaria es la función por defecto usada en los problemas de clasificación binaria.
  Los valores objetivos, categorías, de esta función de clasificación están en el conjunto {0, 1}. Este tipo de entropía calcula una puntuación que

representa la diferencia media entre las distribuciones de probabilidad real y predicha para predecir la clase 1. Si la puntuación se minimiza se obtiene un valor perfecto de entropía cruzada, 0.

- Las **funciones de clasificación multiclase** son aquellas usadas en problemas predictivos de más de dos clases. Cada clase del modelo predictor es asignada a un valor numérico dentro del rango , las funciones se encargarán de predecir la probabilidad de que una muestra pertenezca a cada una de las clases conocidas.

Optimizador

Los optimizadores se encargan de reducir el error cometido por la red modificando los valores de sus hiperparámetros. El proceso para lograr esta tarea es llamado back propagation. Existen múltiples optimizadores, pero el método más común usado en redes neuronales y en el Machine Learning es el descenso del gradiente.

Descenso del gradiente

Es un algoritmo de optimización que actúa como un proceso iterativo, con el objetivo de encontrar el valor de los parámetros de una función que minimice la función de pérdida. Es el optimizador más común para el entrenamiento de redes neuronales.

El algoritmo de descenso del gradiente realiza los siguientes pasos para cumplir su propósito:

1. Se introduce un lote de datos del set de datos de entrenamiento como entrada de la red.
2. Una vez que se realizan los cálculos de cada capa de la red, se obtienen como resultado las predicciones del modelo. Este paso es llamado forward propagation.
3. Se evalúa la función de pérdida, también llamada función de coste, para el set de datos que fue introducido en la red. El resultado de esta función es el que se trata de minimizar usando el descenso del gradiente.
4. Se calcula el valor del gradiente como la derivada parcial de la función de pérdida con respecto a los hiperparámetros de la red. La cantidad de estos parámetros es de millones por lo que la tarea de calcular el gradiente se nos complica ya que es difícil saber la influencia de la variación de un valor de las primeras capas en el coste de capas siguientes. Aquí entra en juego el uso del algoritmo de back-propagation.

5. El optimizador de la red aplica el algoritmo de back propagation calculando las derivadas parciales de las funciones de pérdida para obtener el vector gradiente, empezando por los hiperparámetros de la última capa y haciendo uso de la regla de la cadena. Una vez calculadas las derivadas, pasaremos a la capa anterior calculando de nuevo las derivadas parciales de la función de pérdida usando los hiperparámetros de esta capa. Este proceso iterativo se repetirá hasta alcanzar el inicio de la red.
6. Se actualizan los hiperparámetros de la red restando a su valor actual el valor del gradiente correspondiente y multiplicando la tasa de aprendizaje para ajustar la longitud de los pasos que damos con estas optimizaciones. Cuanto más cerca estemos de un mínimo global, la longitud de los pasos dados será menor debido a que la pendiente de la función de pérdida será menor.

El proceso de descenso del gradiente se aplicará de forma iterativa a lo largo del entrenamiento con el objetivo de alcanzar el mínimo global de las funciones de pérdida, intentando no caer un mínimo local no deseado del que no se pueda salir. Puede suceder que también se quede atascado en un mínimo local en una dimensión y en un máximo local en otra, a este concepto se le conoce como ”punto de silla”.

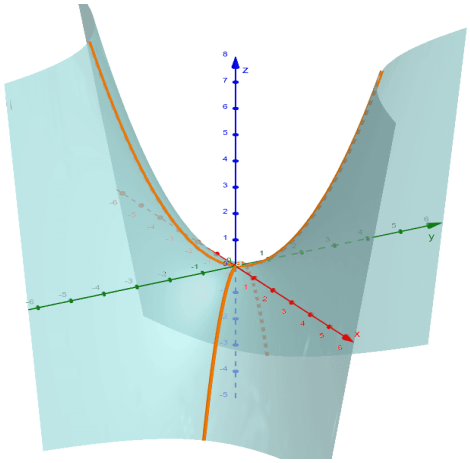


Ilustración 29 Ejemplo de un punto de silla. Obtenido de un proyecto “Ejemplo punto silla”, de la web GeoGebra.

Según el valor de la pendiente de la función podremos saber cuán largos han de ser los pasos que damos mientras descendemos. Si la pendiente del gradiente es elevada daremos un paso largo para descender lo máximo posible, mientras que por lo contrario si la pendiente es poco elevada daremos pasos más pequeños.

Este algoritmo cuenta con distintas variaciones dependiendo del número de lotes que introduzcamos en cada iteración del entrenamiento, guardando relación con los tres modos del tamaño de lotes [43]:

- Descenso del gradiente en lotes: se introducen todos los datos disponibles, que supondrá un problema de estancamiento debido a que el cálculo del gradiente se realizará siempre en todas las muestras. Esto ocasionará que en algún momento del entrenamiento las variaciones sean mínimas.
- Descenso del gradiente estocástico: se introduce una sola muestra por cada iteración, el gradiente se calcula para esa muestra y se obtiene el factor de aleatoriedad ausente en el descenso del gradiente en lotes. Con la aleatoriedad favorecemos que no ocurra el estancamiento, pero no lo eliminamos completamente y añadimos una mayor lentitud al entrenamiento de la red. Otra forma que favorece la eliminación del estancamiento en el descenso estocástico es la reducción progresiva de la tasa de aprendizaje según el gradiente se aproxime al mínimo local.
- Descenso del gradiente en mini-lotes: se introduce un lote cuyo tamaño es menor que el set de datos completo pero mayor que 1. Este enfoque permite conservar las ventajas del descenso del gradiente estocástico junto a una mayor velocidad al entrenar la red. Usando este método de descenso del gradiente lo ideal será escoger un tamaño de lotes que de un buen balance entre el tiempo de entrenamiento y la aleatoriedad de los datos en cada iteración.

Aprendizaje por lotes

El aprendizaje por lotes es utilizado cuando es necesario usar el set de datos completo en cada época del entrenamiento. Si se usa el set completo sin dividirse por lotes, la red neuronal deberá de almacenar todos los errores para todas las muestras en memoria, pudiendo dar lugar a velocidades de entrenamiento muy pobres. Dividiendo el set en lotes conseguimos entrenar la red con subconjuntos de datos, que en cada iteración otorgan una reducción del coste de computación necesario y mejoramos la velocidad del entrenamiento. Al dividir el set de datos en “x” subconjuntos, tardará “x” iteraciones en completarse 1 época del entrenamiento. El modelo aplicará la corrección de los pesos y la actualización de los hiperparámetros por cada lote procesado, en vez de cada vez que itere sobre todo el set de datos.

Si los datos de entrenamiento cambian, será necesario reentrenar completamente el modelo desde cero.

Una de las principales ventajas de este tipo de aprendizaje son sus condiciones de convergencia, que guardan relación con el descenso del gradiente [12]:

- Si el tamaño de los lotes es igual que el set de datos completo, el cálculo del gradiente nos acercará a un mínimo local. Si existen múltiples

mínimos en la función, se puede quedar atascado en el mínimo local más cercano rebotando alrededor de este sin alcanzar el óptimo global.

- Si el tamaño de lote es de solo una muestra no se garantiza su convergencia al óptimo global, sino que rebotará alrededor de este.
- Usando un tamaño de lotes pequeño pero mayor que una muestra, evitamos la tendencia de estancarse en un mínimo local.

Tasa de aprendizaje

Un componente relacionado con las funciones de pérdida durante el entrenamiento de la red es la tasa de aprendizaje, la cual funciona como un parámetro de afinación afectando a la velocidad en la que el algoritmo converge en el valor óptimo, mínimo, de la función de pérdida. Los valores habituales de la tasa de aprendizaje oscilan en el rango entre 0.1 y 0.01. Una tasa de aprendizaje mayor puede compensar tamaños grandes de lotes durante el entrenamiento.

El tamaño de la tasa de aprendiza influye en la longitud de los pasos dados en el descenso del gradiente.

- Una tasa de aprendizaje muy pequeña nos llevará a dar pasos muy pequeños, pero acabaremos convergiendo en el mínimo deseado.
- Una tasa de aprendizaje demasiado alta hará que demos pasos muy largos y no seamos capaz de descender a la zona del mínimo, saltando por encima de ella y provocando una oscilación en torno al mismo que nunca termina de converger.

Existe una buena práctica en el entrenamiento de redes conocida como decaimiento de la tasa de aprendizaje [44], en la cual la tasa de aprendizaje no es un hiperparámetro fijo, sino que se actualiza dependiendo de distintas condiciones. Ha sido demostrado que este método proporciona mejoras en la optimización y generalización conseguida durante el entrenamiento de la red.

Existen varias maneras de decaer la tasa de aprendizaje, pero las más comunes son:

- Decrementar su valor cada cierta iteración, hasta llegar a un valor próximo a cero, durante todo el entrenamiento.
- Decrementar su valor durante un número de iteraciones concreto y una vez se ha alcanzado, usar la tasa de aprendizaje obtenida de forma fija.

Set de datos

Previo al entrenamiento de la red se selecciona un conjunto de datos reales, cuyos valores servirán como entrada de la red durante su entrenamiento. Existen tres tipos de sets de datos:

- De entrenamiento: compuesto por las muestras utilizadas para ajustar el modelo durante su entrenamiento.
- De validación: compuesto por las muestras utilizadas para proporcionar una evaluación insesgada del modelo mientras que se ajustan los hiperparámetros del modelo
- De pruebas: al igual que el set de validación proporciona una evaluación insesgada, pero se realiza sobre el modelo final para comprobar su rendimiento.

No existe un porcentaje general para dividir el set de datos, pero si hay factores a tener en cuenta para decidir este porcentaje.

- Cuantos menos datos de entrenamiento haya, mayor varianza tendrán los hiperparámetros del modelo.
- Cuanto menor sea el número de datos de validación o de prueba, mayor varianza tendrá el rendimiento del modelo.

Teniendo en cuenta estos factores se suele elegir como proporción inicial un 80% de datos para entrenamiento y un 20% para validación, siguiendo el principio de Pareto [13]. Si queremos dividir el set de datos para entrenamiento, validación y pruebas, la forma ideal de hacerlo es con la validación cruzada k-Fold.

Los sets de datos pueden ser etiquetados o no etiquetados, usándose el primer tipo en aprendizaje supervisado y el segundo en no supervisado. Cuando un set de datos es etiquetado quiere decir que los datos de entrenamiento se componen de pares de objetos, un componente es el dato de entrada y el otro los valores objetivo que se desean alcanzar.

Redes neuronales y el concepto de superresolución

Existen dos conceptos denominados percepción y generación, los cuales respectivamente significan entender que hay en la imagen y sintetizar los pequeños detalles de la imagen. Estos dos conceptos son las dos claves principales que siguen los métodos de reescalado basados en aprendizaje, siendo los casos de éxito de estos algoritmos cuando nuestra imagen de entrada nos facilita mucha de la estructura real de la imagen como brillos, bordes de objetos o relieves.

Los algoritmos basados en aprendizaje se van a encargar de generar con gran definición los pequeños detalles de la imagen aplicando texturas y patrones que se han aprendido durante su entrenamiento al observar otras imágenes, este último concepto es conocido como alucinación de las redes neuronales.

El término de alucinar se refiere a la generación de los patrones que la red ha ido aprendiendo internamente durante su entrenamiento. Aplicado al problema de superresolución, nos interesa que la red alucine patrones de bajo nivel como bordes, esquinas, texturas de superficies o pequeños detalles.

En este proceso de superresolución nuestro modelo está alucinando detalles de la imagen que ha aprendido durante su entrenamiento, por lo que los datos de entrada escogidos pueden llegar a condicionar los resultados obtenidos. Este factor es importante ya que dependiendo del uso que le queramos dar a esta tecnología de superresolución deberemos usar unos datos de entrada u otros. Existen diferentes tipos de modelos para solucionar el problema de superresolución en diferentes ámbitos, como podrían ser la restauración de imágenes antiguas o la aplicación en diagnósticos médicos.

Una mala selección de estos datos podría resultar en un sobreajuste del modelo ocasionando que este no sepa analizar y procesar correctamente imágenes fuera del set de entrenamiento causando una falta de generalización del modelo y que no sea fiable aplicarlo en escenarios externos o de mayor relevancia al usado durante el entramiento.

Algunos casos famosos de la aplicación de redes para solucionar el problema de superresolución es la remasterización de videojuegos [14,26].

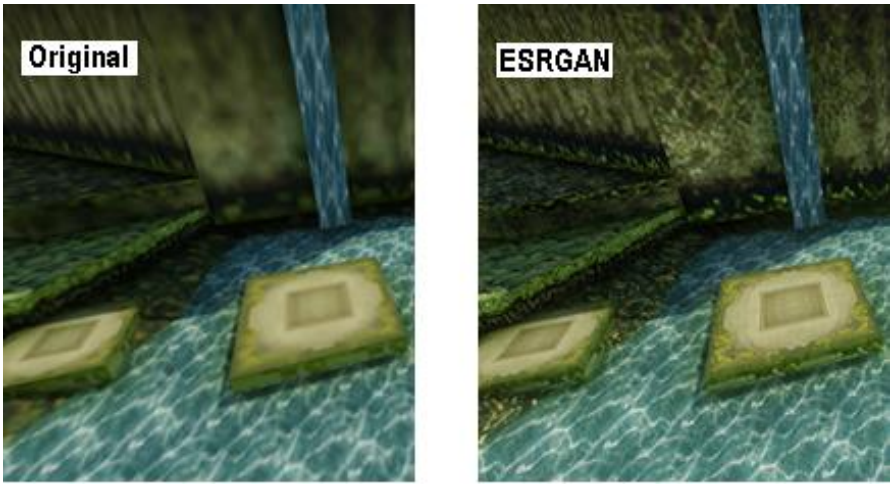


Ilustración 30 Comparativa modelado 3D en baja resolución con su versión reescalada usando el modelo ESRGAN. Obtenida del hilos “ESRGAN-La I.A que se usa para reescalar imagen y vídeo en alta definición” del foro El Otro Lado.

Objetivos del proyecto

Se proponen una serie de objetivos generales a cumplir durante el desarrollo del proyecto.

Objetivos de la investigación

- OI1. Analizar las diferentes soluciones posibles y escoger que modelo neuronal. Este modelo se implementará desde cero en base a otros proyectos para solucionar el problema de superresolución.
- OI2. El modelo desarrollado deberá de ser capaz de aumentar 4 veces la resolución de la imagen de entrada, 2 veces en cada dimensión.

Objetivos de la red neuronal

Se espera cumplir los siguientes objetivos durante el desarrollo del modelo neuronal:

- OR1. El error cuadrático obtenido de la imagen generada comparada con la imagen de referencia tendrá que oscilar entre el 5% y 10%.
- OR.2 Obtener una estructura del generador del modelo que sea ligera, con pocos nodos respecto a otros modelos, que no suponga una gran computación para los dispositivos móviles.
- OR3. Conseguir exportar el modelo de la red a un formato de archivo ".tflite" para su uso en dispositivos móviles.

Requisitos funcionales del sistema Android

- RG1. El sistema aplicará el reescalado de superresolución a la imagen seleccionada por el usuario.
    - RG1.1. La imagen podrá ser escogida de la galería de imágenes del dispositivo del usuario.
    - RG1.2. La imagen podrá ser tomada con la cámara del dispositivo del usuario.
- RG2. El usuario podrá guardar la imagen resultado del reescalado en la galería de su dispositivo móvil.
- RG3. El usuario podrá escoger sobre el hardware que se ejecute el método de reescalado.
    - RGI3.1 CPU.
    - RGI3.2 GPU.
- RG4. El usuario podrá escoger si se limita el tamaño de los paquetes en los que se procesa la imagen escogida.

Planificación

Metodología de trabajo

Para el desarrollo de las distintas fases del proyecto se aplicará un modelo en cascada como metodología de trabajo. Esta elección se basa principalmente en el concepto de que muchas tareas del proyecto dependerán de la realización previa de otras.

Fases principales del proyecto

El proyecto será dividido en cuatro etapas principales: desarrollo del modelo neuronal, entrenamiento del modelo, desarrollo de la aplicación Android y escritura de la memoria.

Desarrollo del modelo

El punto de partida del proyecto será la investigación de las diferentes técnicas ya existentes de superresolución, que dará lugar a mi descubrimiento del uso de redes neuronales en este campo.

Una vez haya sido recabada la información necesaria sobre el tipo de redes neuronales que se usan para solucionar el problema de superresolución, se optará por el uso de las redes generativas adversarias. La gran mayoría de proyectos que resuelven este problema con resultados notorios usan este tipo de red.

Una vez se haya escogido el tipo de red a implementar, se realizará la investigación sobre varios artículos científicos e implementación de una red GAN para solucionar el problema de superresolución. La implementación de la red se apoyará en el uso de las librerías de código abierto Tensorflow y Keras. La implementación de la red se basa en conceptos clave de los modelos SRGAN [9] y ESRGAN [8].

Se usará para el entrenamiento de la red el set de datos es General-100 [15]. Este set fue creado para el problema de superresolución y está compuesto por 100 imágenes escogidas de forma que la red pueda entender de forma general cualquier input de entrada que tenga sobre una imagen del mundo real. Este set nace como una extensión del set T91 [46], el cuál fue creado con el mismo propósito.

Durante esta fase se realizarán varias modificaciones de la arquitectura inicial del modelo del generador y el ajuste de varios parámetros del entrenamiento.

Entrenamiento del modelo

Durante esta etapa se realizará un ajuste inicial de los parámetros de entrenamiento junto a la realización de múltiples entrenamientos con variaciones en estos valores de los parámetros de entrenamiento.

Desarrollo aplicación Android

Tras obtener un modelo funcional de la red se desarrollará la aplicación Android capaz de correr el modelo y realizar predicciones a partir de una imagen elegida por el usuario.

La aplicación Android se desarrollará bajo la premisa de que sea simple de utilizar, por lo que en su primera ejecución se mostrará un pequeño tutorial de cómo utilizar las distintas funciones que ofrece.

Escritura de la memoria

Como última fase se recopilará toda la información obtenida durante todo el proceso de investigación y desarrollo. El contenido será añadido a esta memoria junto a las conclusiones del trabajo realizado.

Diagrama Gantt

El siguiente diagrama muestra las diferentes tareas del proyecto desglosadas y ubicadas en el tiempo que ha durado el mismo. Junto al proyecto, se adjunta un documento sencillo de planificación en el que se muestran estas tareas.

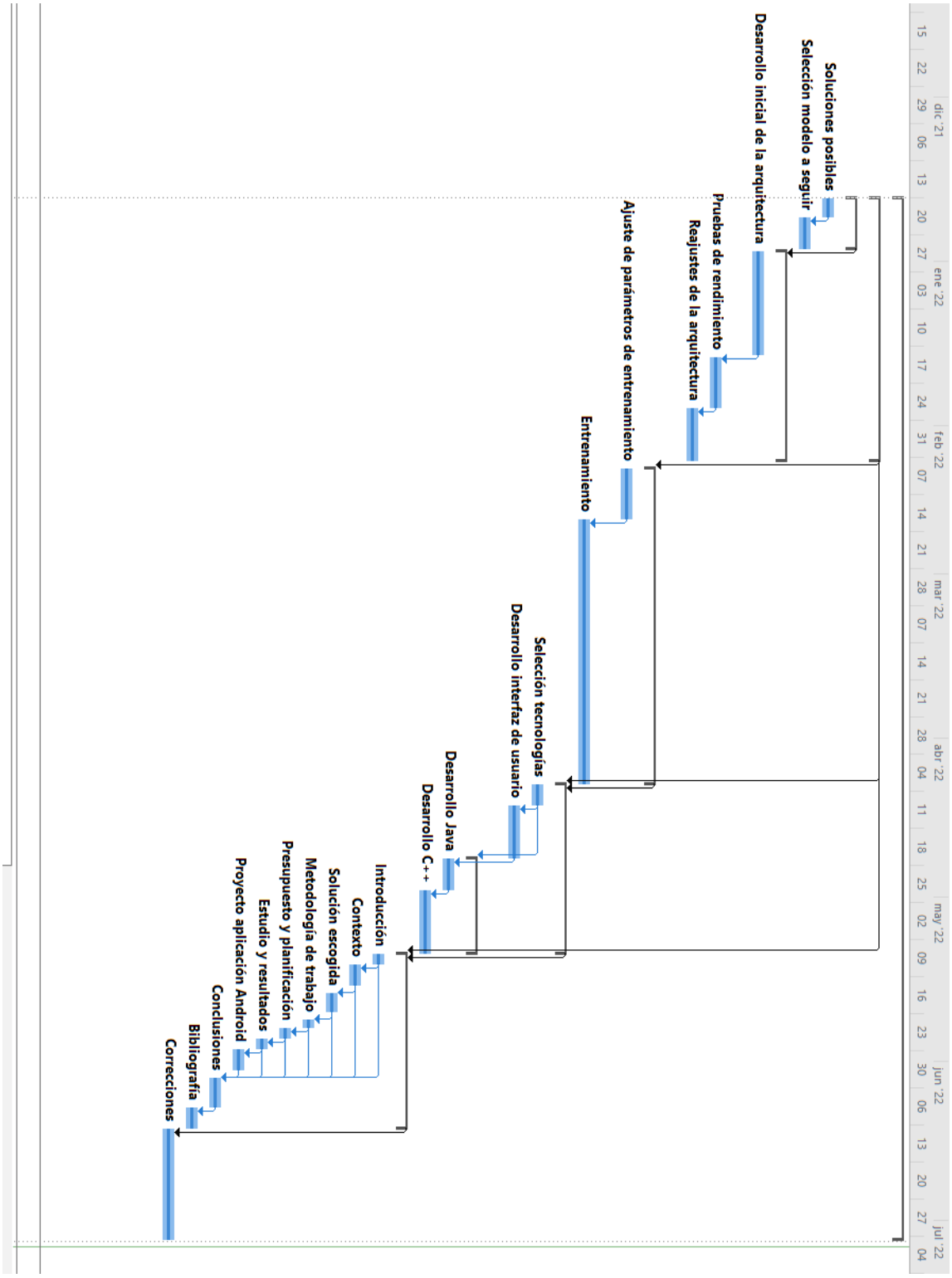


Ilustración 31 Diagrama Gantt de la planificación del proyecto.

Trabajo desarrollado

Desarrollo del modelo neuronal

El proyecto ha sido desarrollado en el lenguaje Python 3.8, junto a la herramienta Jupyter Notebooks. Las librerías de código abierto usadas para el soporte del modelo han sido Tensorflow 2.4.1, Tensorflow-GPU y Keras. Para la exportación del modelo a la aplicación Android se ha usado la librería Tensorflow Lite.

Para la creación del modelo de superresolución se ha escogido una red generativa adversaria relativista [16], formada por un generador y un discriminador relativista. El discriminador relativista se va a encargar de estimar probabilísticamente si los datos reales son o no más realistas que los datos alucinados por el generador

La arquitectura del modelo se ha desarrollado basándose en los modelos ESRGAN [8] y SRGAN [9], con el objetivo de recrear una versión más ligera capaz de ejecutarse en móviles sin inconvenientes.

Modelo VGG

El entrenamiento del generador se ha apoyado con el uso de un modelo previamente entrenado, el modelo VGG19. Este modelo es un tipo de red neuronal convolucional entrenado con el set de datos ImageNet, originalmente desarrollado para resolver el problema del reconocimiento de objetos en la competición de **Image Net Large Scale Visual Recognition Challenge (ILSVRC).** Este modelo ha logrado una precisión del 97% en su tarea sobre el set de datos de ImageNet [17], y contiene un total 1.281.167 imágenes. En el modelo SRGAN [9] se implementó el uso de este modelo con el propósito de sustituir el método del cálculo el error absoluto entre todos los píxeles de la imagen, conocido como "**pixel-wise**".

Se introduce como entrada la imagen alucinada por el generador en este modelo para obtener un valor de pérdida perceptual basado en los mapas de activación de la imagen. Este valor de pérdida lo usaremos junto al valor de pérdida del generador para obtener el valor total de pérdida de la red generadora.

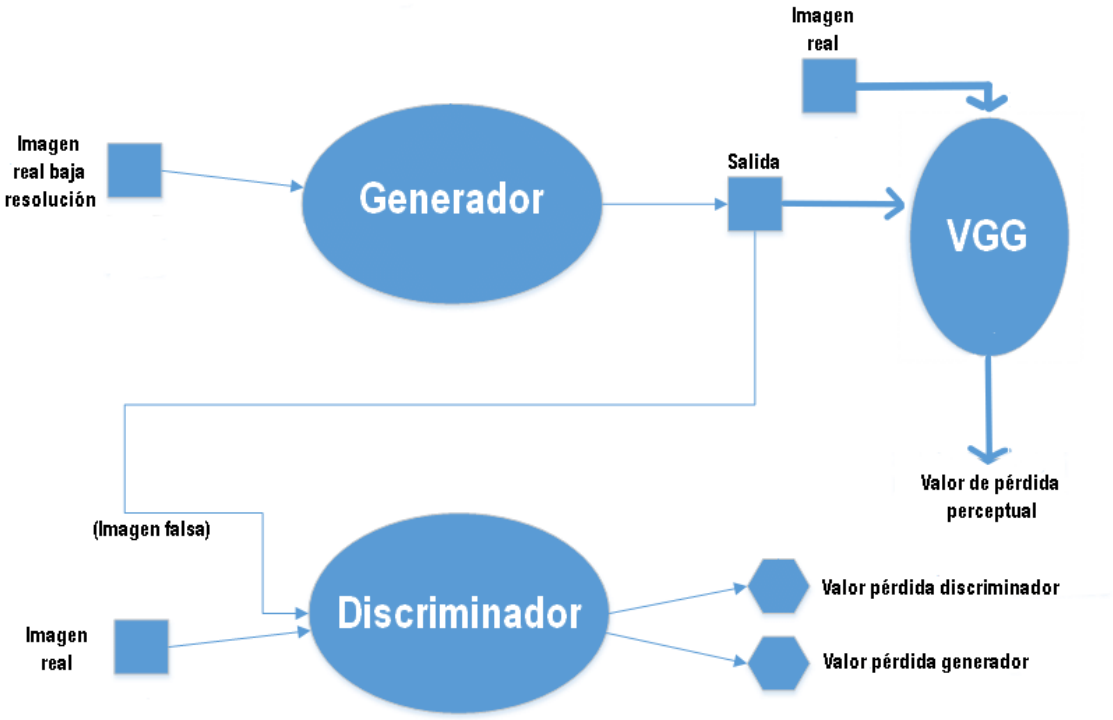


Ilustración 32 Arquitectura de la red con el uso del modelo VGG. Obtenida y editada del artículo "Generator From Edges: Reconstruction of Facial Images" del portal Research Gate.

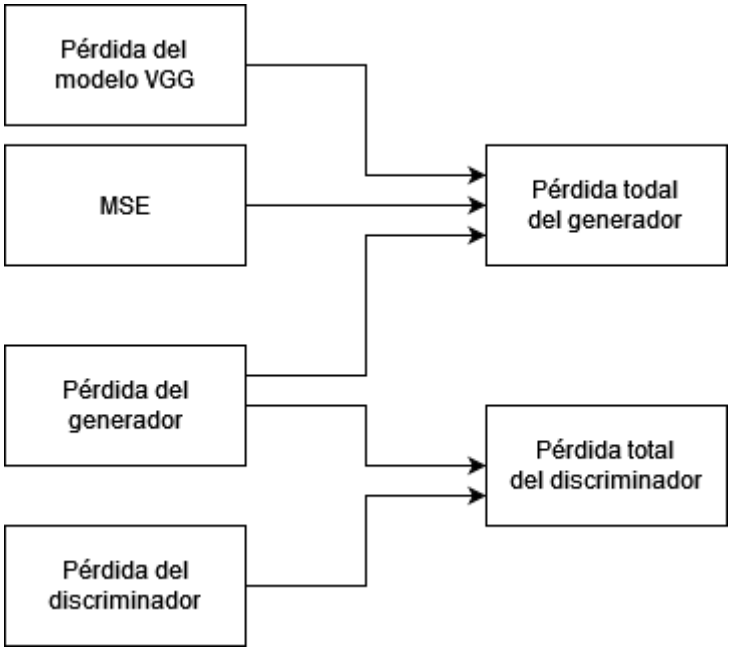


Ilustración 33 Diagrama del uso de los valores de pérdida para calcular el total del generador y discriminador.

Con esta metodología buscamos aumentar la calidad percibida de la imagen sobre la calidad objetiva. De esta manera se obtiene de forma teórica una imagen de superresolución con mayor calidad visual percibida por el ojo humano.

Discriminador relativista de la red

En este modelo en vez de usar un discriminador estándar, se hace uso de un discriminador relativista encargado de predecir la probabilidad de que una imagen real sea más realista que una falsa generada. Esta arquitectura está basada en la usada para el discriminador del modelo ESRGAN.

Arquitectura

La arquitectura del discriminador es la misma utilizada en el modelo SRGAN, compuesta por distintos bloques.

La primera capa de entrada del discriminador es una capa convolucional Conv2D con un tamaño de filtros de capa de 128, un tamaño del kernel de 4x4 con una función de activación tipo Leaky ReLU.

El resto de las capas del discriminador están compuestas por una estructura de bloques con el siguiente formato:

- Una capa convolucional con una cantidad variable de filtros y también un kernel variable en tamaño.
- Una capa de normalización de lotes.
- Una capa de función de activación Leaky ReLU.

El siguiente diagrama muestra la composición completa de la red del discriminador. Los valores “n” representan el número de filtros de la capa convolucional, los valores “k” el tamaño del kernel que al ser de dimensiones simétricas solo se indica un valor, y los valores “s” los strides.

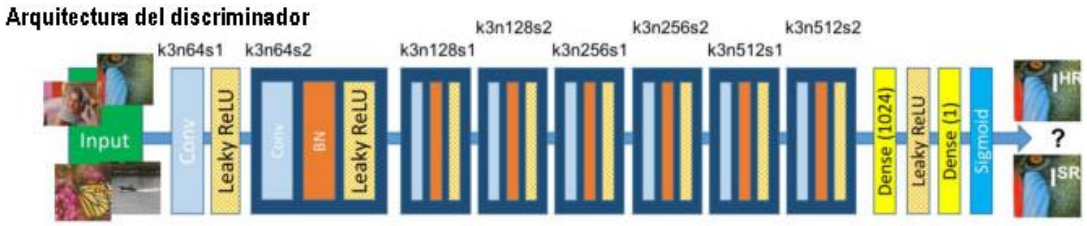


Ilustración 34 Arquitectura del discriminador usado en la solución escogida. Obtenida del artículo científico de SRGAN [9].

Finalmente se incluye una capa del tipo Dense, la salida de esta capa es de un solo valor. Así la red dirá si la imagen procesada es real o no.

Función de pérdida

La función de pérdida adversaria del discriminador relativista se encarga de calcular la entropía cruzada de cada uno de los pixeles de las imágenes que hemos obtenido, acotándolos al dominio de los valores {0,1}. Teniendo todos los valores normalizados se asume que si todos los valores son igual a 1, el resultado del discriminador está indicando que la imagen de entrada es lo más real posible bajo su criterio.

Existen dos valores idóneos para el discriminador dependiendo de su entrada. Si la entrada del discriminador es una imagen real, el valor idóneo obtenido de este sería un tensor con todos los valores iguales al valor máximo del dominio normalizado, uno. En cambio, si la entrada fuese la imagen alucinada por el generador, el valor idóneo de respuesta del discriminador sería un tensor con todos sus valores al mínimo del dominio normalizado, es decir cero.

El concepto de los valores idóneos de la respuesta del discriminador frente a una entrada es utilizado para calcular el valor de pérdida total de este. El valor total de pérdida del discriminador lo vamos a obtener con la suma de los dos siguientes valores:

- El valor de pérdida del discriminador es el valor resultado de la entropía cruzada binaria entre el valor idóneo al observar una imagen real y, la resta del valor del discriminador al observar una imagen real menos el valor de observar la imagen alucinada por el generador.

$$l_D^{Ra} = \left[\sigma(C(x_{or}) - \mathbb{E}(\Delta(x_r, x_f))\right] \quad (8)$$

- El valor de pérdida generado es el valor resultado de la entropía cruzada binaria entre el valor idóneo del discriminador al observar una imagen falsa y, la resta del valor del discriminador al observar la imagen alucinada por el generador menos el valor de observar la imagen real.

$$l_D^{Ra} = \left[\sigma(C(x_{of}) - \mathbb{E}(\Delta(x_f, x_r))\right] \quad (9)$$

Esta función de pérdida está basada en la desarrollada para el discriminador del modelo ESRGAN.

Generador de la red

Los generadores tienen dos objetivos principales. Su primer objetivo es alucinar una imagen que sea identificada como realista por el discriminador, siendo comparada con su referencia real del set de datos. Por otro lado, las redes generativas adversarias se cimientan en el concepto de la competición entre el generador y discriminador y, por lo tanto, el otro objetivo del generador es maximizar el error del discriminador.

El modelo desarrollado para el generador de la red generativa adversaria se ha basado en los generadores de los modelos de superresolución SRGAN y ESRGAN, desarrollando un nuevo generador ligero capaz de escalar imágenes al doble de su resolución original en cada dimensión.

Una las principales características de este generador desarrollado es la baja cantidad de nodos que posee, unos 700.000 nodos comparados con los 18 millones que tiene el modelo ESRGAN. Esta cantidad permite aligerar el proceso computacional necesario, favoreciendo el uso del modelo en aplicaciones para dispositivos móviles.

Arquitectura

La estructura principal del modelo generador se compone de una arquitectura residual formada por una combinación múltiple de bloques residuales con conexiones de salto entre ellos. Cada uno de estos bloques está formado por dos capas:

- Capa convolucional 2D: esta capa aplica filtros convolucionales a su entrada generando un tensor de salidas. La función de activación usada en esta capa es una función ReLU.
- Capa de suma: esta capa se encarga de tomar como entrada una lista de tensores del mismo tamaño y devolver un solo tensor, también del mismo tamaño

Las conexiones de salto usadas en los bloques residuales de este modelo son del tipo aditivas. Cuando la salida de la capa predecesora es añadida a la entrada de la capa siguiente no se necesitan parámetros de configuración extra. En los modelos residuales estas conexiones ayudan en la tarea de preservar el valor gradiente.

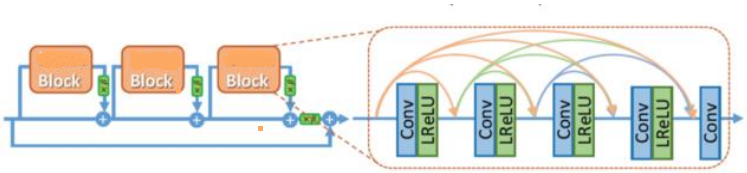


Ilustración 35 Estructura de los conjuntos de bloques residuales usados en la red generadora. Imagen editada del artículo original de ESRGAN [8].

La arquitectura del generador se cierra después de la combinación los múltiples bloques residuales con otra pequeña combinación de capas que es la encargada de reescalar la imagen final, formando el bloque de reescalado:

- Capa convolucional
- Capa SubpixelConv2D, factor de reescalado ajustado a x2 en cada eje.
- Capa de función de activación Leaky ReLU.

La capa SubpixelConv2D está basada en la implementación de [40].

Funciones de pérdida

El error adversario del generador es la diferencia entre el resultado del discriminador al evaluar la imagen generada y el valor idóneo en el que la imagen generada fuese interpretada como real por el discriminador.

La función de pérdida total del generador es la suma del valor de pérdida del modelo VGG, el error adversario del generador, y el error cuadrático medio del modelo. Obtenemos un valor total de pérdida del generador de la red cuya fórmula es:

$$L_G = \lambda L_{VGG} + \eta {L_G}^{Ra} + MSE \quad (10)$$

El valor del modelo VGG está ponderado por $\lambda = 10^{-6}$, y el error adversario está ponderador por $\eta = 10^{-3}$. El error cuadrático medio destaca sobre el resto de los valores de pérdida del generador y como resultado se obtendrá una imagen más suave. Las ponderaciones se han ajustado a las usadas en el modelo SRGAN.

Con los diferentes componentes que se han explicado en esta sección de “Desarrollo del modelo neuronal”, se va a construir la red generativa adversativa usada para solucionar el problema de superresolución.

Desarrollo de la Aplicación para Android

El sistema se ha desarrollado para el sistema operativo Android teniendo en cuenta las limitaciones que este puede ofrecer a la hora de realizar tareas que requieran un cálculo computacional mayor al habitual de las aplicaciones del sistema.

La aplicación corre bajo el SDK de Android y se ha usado tanto el lenguaje Java para la lógica de las vistas, como C++ para procesar y obtener la inferencia del modelo de la red neuronal sobre la imagen que escojamos.

Se ha escogido ejecutar el código de inferencia de la red sobre el lenguaje C++ para evitar la limitación de la cola de memoria de Java, debido a que de esta forma el código puede ser ejecutado sobre la memoria nativa del dispositivo. El tamaño de la memoria para que Java que puede asignar en cada aplicación del dispositivo viene asignado por el fabricante del dispositivo y puede estar en el rango de 16 a 512 MB [18]. Debido a esta incertidumbre en el tamaño que podría tener el dispositivo en el que se ejecute la aplicación se tomó la decisión de ejecutar el código en C++.

Funciones

La aplicación permite escoger el origen de la imagen de entrada se, puede provenir de la galería del dispositivo o de la cámara. El resultado de la inferencia de la red neuronal puede almacenarse en la galería del dispositivo móvil.

Desde la vista de configuración de la aplicación, se puede seleccionar si la inferencia de la red se quiere ejecutar sobre la CPU o la GPU del dispositivo y limitar el tamaño de los segmentos en los que se divide la imagen de entrada.

Limitaciones

Si se ejecuta desde la CPU la ejecución es mucho más lenta pero más segura, ya que la ejecución sobre la GPU puede dar problemas de falta de memoria dependiendo de la memoria dinámica disponible del teléfono. Si la imagen es demasiado grande y se ejecuta en la GPU es seguro que va a dar un fallo de memoria.

Dependiendo de la cantidad de memoria dinámica que pueda usar el dispositivo móvil la probabilidad de que la aplicación lance errores de falta de memoria aumenta. Para solucionar esta limitación, la imagen de entrada se divide en diferentes segmentos de tamaño limitado.

Con las pruebas realizadas se ha limitado el tamaño de los segmentos a 320 píxeles de altura y 320 píxeles de ancho. Aun así, sigue siendo posible que ocurran errores con dispositivos que no se hayan probado por lo que también existe la opción de limitar los segmentos a un tamaño de 64 píxeles de altura y 64 píxeles de ancho.

Las fotos tomadas de la cámara se han limitado una resolución estándar de 720p, ya que los componentes para visualizar imágenes de Android limitan el tamaño de la imagen a 2048x2048 píxeles.

```
E/AndroidRuntime: FATAL EXCEPTION: main
    Process: org.uniovi.guillermo.superresolution, PID: 10617
    java.lang.RuntimeException: Canvas: trying to draw too large(165888000bytes) bitmap.
        at android.graphics.RecordingCanvas.throwIfCannotDraw(RecordingCanvas.java:266)
        at android.graphics.BaseRecordingCanvas.drawBitmap(BaseRecordingCanvas.java:94)
```

Ilustración 36 Error de fallo de memoria de la aplicación Android.

Estructura del proyecto

La estructura del proyecto es muy sencilla ya que no abarca demasiadas funcionalidades. La vista de la aplicación se compone de cuatro pantallas de tipo “Activity”:

- SplashActivity: vista splash de inicio.
- WelcomeActivity: vista tutorial de uso de la aplicación.
- MainActivity: vista principal.
- ConfigurationActivity: vista configuración.

Existe una quinta Activity que sirve como clase padre para aquellas que necesitan actualizar valores globales o presentar pantallas de carga, pero no tiene ningún uso por si sola.

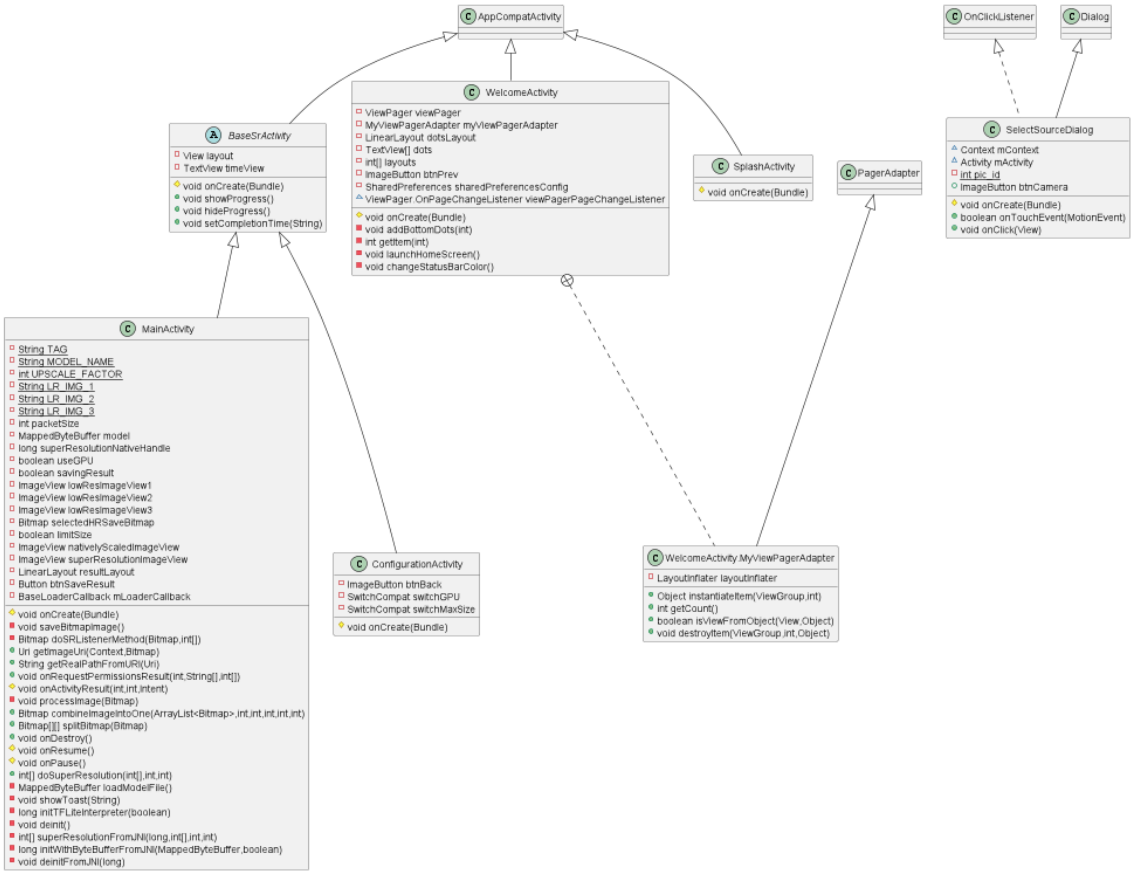


Ilustración 37 Diagrama de las clases de vista de la aplicación.

La parte de la lógica de superresolución se divide en dos bloques. Un bloque sirve como puente entre el código Java y el código C++, mientras que el otro bloque se encargará de realizar las funciones necesarias para la inferencia escrita en C++.

- El bloque de puente usará la interfaz nativa Java JNI, para poder traducir el código en otros lenguajes a código Java.
- El bloque de lógica está compuesto por dos archivos: SuperResolution.cpp y SuperResolution.h. Contienen los métodos para la invocación del intérprete de la red y realizar su inferencia.

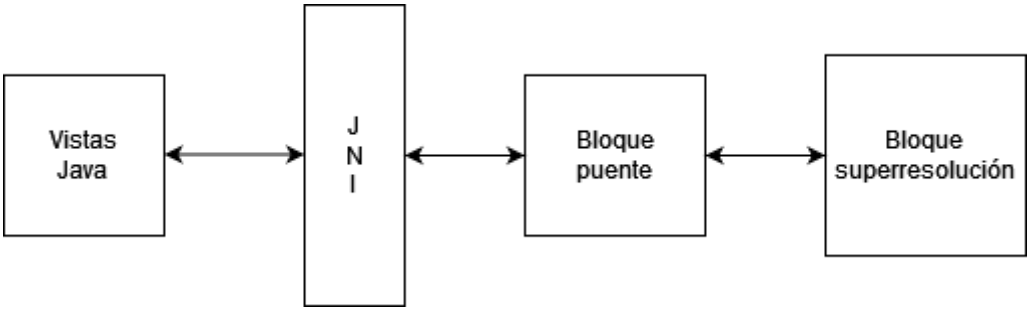


Ilustración 38 Diagrama de la conexión entre los bloques de la app y la interfaz nativa de Java.

El código de C++ para la inferencia del modelo cargado con Tensorflow Lite se ha basado en la implementación base propuesta por el equipo de la misma organización [19].

La aplicación también hace uso de pequeñas clases Java para resolver tareas de utilidad: cargar las imágenes, crear un archivo de guardado o simplemente almacenar constantes que se usan en varias Activity.

Flujo entre pantallas

Al arrancar la aplicación se muestra una pantalla de bienvenida con una pequeña animación del logo de la Escuela de Ingeniería Informática de Oviedo, a esta vista corresponde a la clase SplashActivity. Si es la primera vez que se ejecuta la aplicación, se le mostrará al usuario un tutorial con qué funciones ofrece la aplicación, esta vista corresponde a la clase WelcomeActivity.

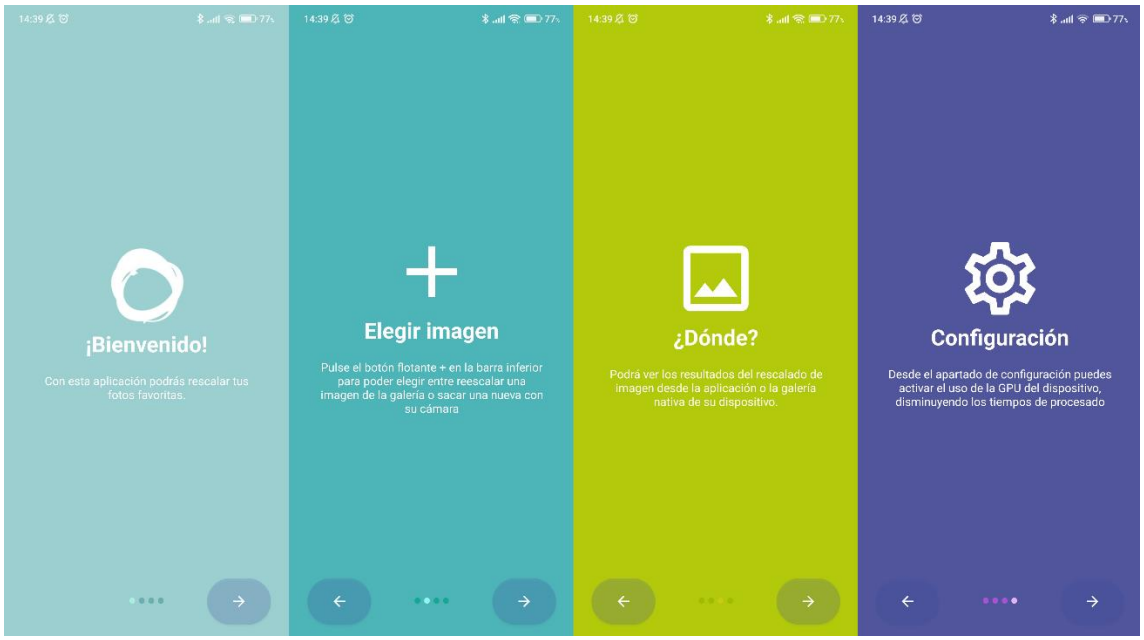


Ilustración 39 Capturas de las cuatro etapas que tiene el tutorial de inicio de la aplicación.

Una vez finalizado el tutorial se nos mostrará la vista principal de la aplicación, asociada a la clase MainActivity. Desde esta pantalla se puede escoger una de las tres imágenes de ejemplo ubicadas en la parte superior o mostrar más opciones pulsando el botón flotante.

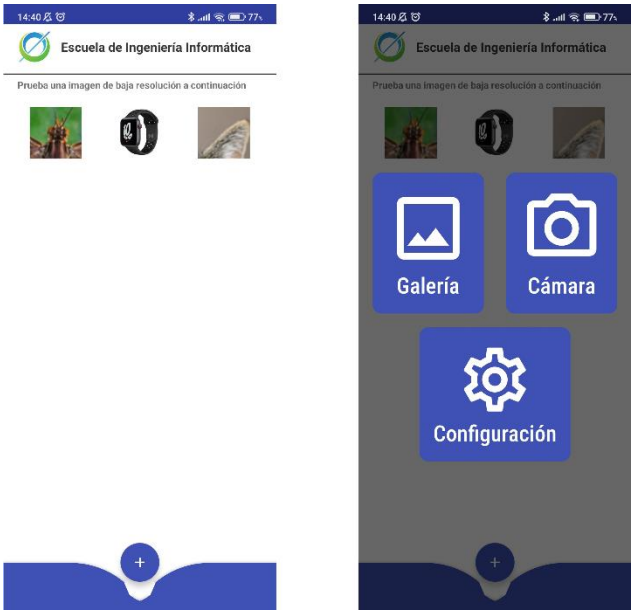


Ilustración 40 Vista de inicio y modal de opciones respectivamente.

Desde el modal de opciones se puede acceder a la galería o a la cámara del dispositivo móvil para elegir la imagen a la que se quiere aplicar el método de superresolución. También podemos acceder a la vista de configuración donde podemos habilitar la GPU del dispositivo para el procesado de la imagen o limitar el tamaño de los paquetes en los que se divide la imagen al procesarse.

Ilustración 41 Vista de configuración de la aplicación.

Si escogemos una de las opciones de procesado de la imagen, se ejecutará el proceso de reescalado. Este paso tardará más o menos tiempo dependiendo del tamaño de la imagen original y el número de segmentos en el que se divida. Después de la inferencia se aplica a la imagen un filtro bilateral usando la librería OpenCV. Con este filtro conseguimos reducir el ruido de la imagen final y preservar los bordes de la imagen, eliminando algunos artefactos que pudiesen ser generados durante la inferencia.

Una vez finalizado podremos observar la imagen original y el resultado en la página de inicio, si lo deseamos podemos tocar encima de cada una de las imágenes para expandir la imagen al tamaño total de la pantalla. En la parte inferior de la pantalla aparece un botón que nos permite guardar la imagen reescalada en la galería nativa del dispositivo móvil.

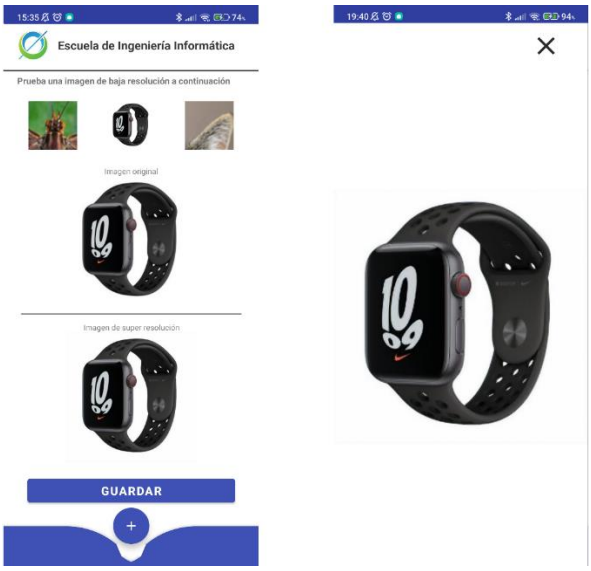


Ilustración 42 Pantalla de inicio de la aplicación con los resultados del proceso de superresolución y zoom de la imagen.

El siguiente diagrama de flujo muestra la navegación completa de la aplicación con todos sus estados.

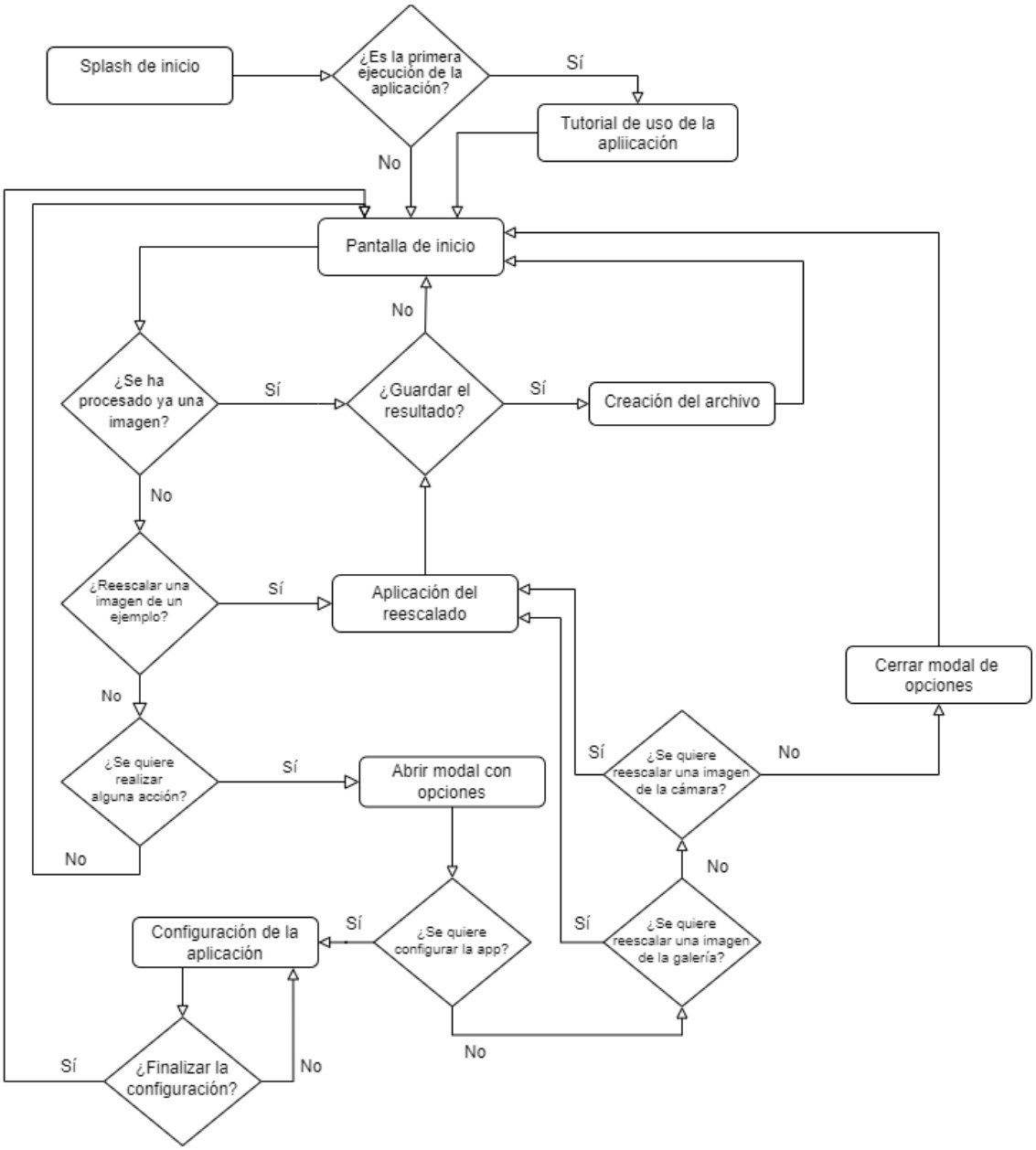


Ilustración 43 Diagrama de flujo, muestra la navegación de la aplicación.

Pruebas y resultados

Para el entrenamiento de la red se ha usado el set de datos General-100 compuestos por 100 imágenes distintas que intenta abarcar un conjunto general de entornos y seres vivos, del cual se han usado 90 imágenes para el entrenamiento y las restantes para la evaluación del modelo.

Antes de introducir en la red las diferentes muestras del set de datos, las imágenes de entrada se reescalan con una interpolación bilineal a un tamaño de 64 pixeles de altura y de ancho, y las imágenes objetivo a 128 x 128 píxeles. Las imágenes se normalizan en el dominio [-1,1], con el objetivo de hacer la computación más eficiente y tener los datos en un rango similar para que los gradientes de la red no se salgan de control cuando sean calculados.

Durante el proyecto se han realizado varias iteraciones sobre el código original de la red generadora del modelo, en pos de refinar su arquitectura y obtener una red ligera y que diese buenos resultados.

El entrenamiento de la red ha sido supervisado porque partimos de un conjunto de datos etiquetado previamente, es decir, conocemos el valor del atributo objetivo para el conjunto de datos del que disponemos. Se ha realizado el ajuste de los parámetros de entrenamiento tras pruebas de 4000 épocas.

Parámetros de entrenamiento

Se ha seleccionado una serie de parámetros generales para el entrenamiento del modelo, que se listan a continuación:

- Tamaño de los lotes: valor de 4. Se ha ajustado a este tamaño debido a que si el tamaño era mayor mi GPU no era capaz de ejecutar el entrenamiento y a un tamaño menor los resultados de la red eran casi idénticos, pero tomaban más tiempo de entrenamiento.
  Cabe destacar que el tamaño del lote es una potencia de 2 debido a la alineación de los procesadores físicos y virtuales de la GPU. Por cada lote se hace uso de un procesador virtual y los procesadores físicos se estructuran en conjuntos de tamaños de potencia de dos, por lo que si usamos un valor que no sea una potencia de 2 en el tamaño del lote obtendremos un bajo rendimiento.
- Tasa de aprendizaje inicial: 0.0001. El mismo valor que se usa en el entrenamiento de los modelos SRGAN y ESRGAN.
- Valor beta1, utilizado para los optimizadores de los gradientes: 0.9. Es el valor por defecto en los optimizadores que proporciona la librería de Keras.

- Tamaño set de entrenamiento: 90, las 10 imágenes restantes del set se han reservado para el set de pruebas del modelo. En una red generativa adversaria no es necesario el uso de un test de validación, los hiperparámetros son ajustados por los optimizadores a partir de los valores de pérdida obtenidos del procesamiento de las muestras.

Comparativa de los modelos

Se han usado tres tipos de métricas para medir la eficacia de nuestro modelo frente a otras soluciones para el problema de superresolución:

- MSE o Error cuadrático medio, representa la media de las desviaciones al cuadrado entre las predicciones y los valores verdaderos.
- SSIM o Índice de similitud estructural, es una métrica perceptual utilizada para cuantificar la degradación de la calidad de la imagen respecto a una imagen original.
- PSNR o Proporción Máxima de Señal a Ruido, representa la relación entre la energía máxima posible de una señal y el ruido que afecta a su fiel representación. Sirve como medida cuantitativa de la percepción humana sobre la calidad de la reconstrucción de la imagen. Si no existiese ruido en la imagen, siendo ambas imágenes idénticas, el valor del PSNR sería infinito.

Se ha realizado la comparativa de nuestro modelo frente al modelo neuronal SRGAN y a los métodos clásicos de interpolación bilineal y método de proximidad o vecino más cercano.

El modelo ESRGAN usado ha sido entrenado por el equipo de desarrollo de Tensorflow [20].

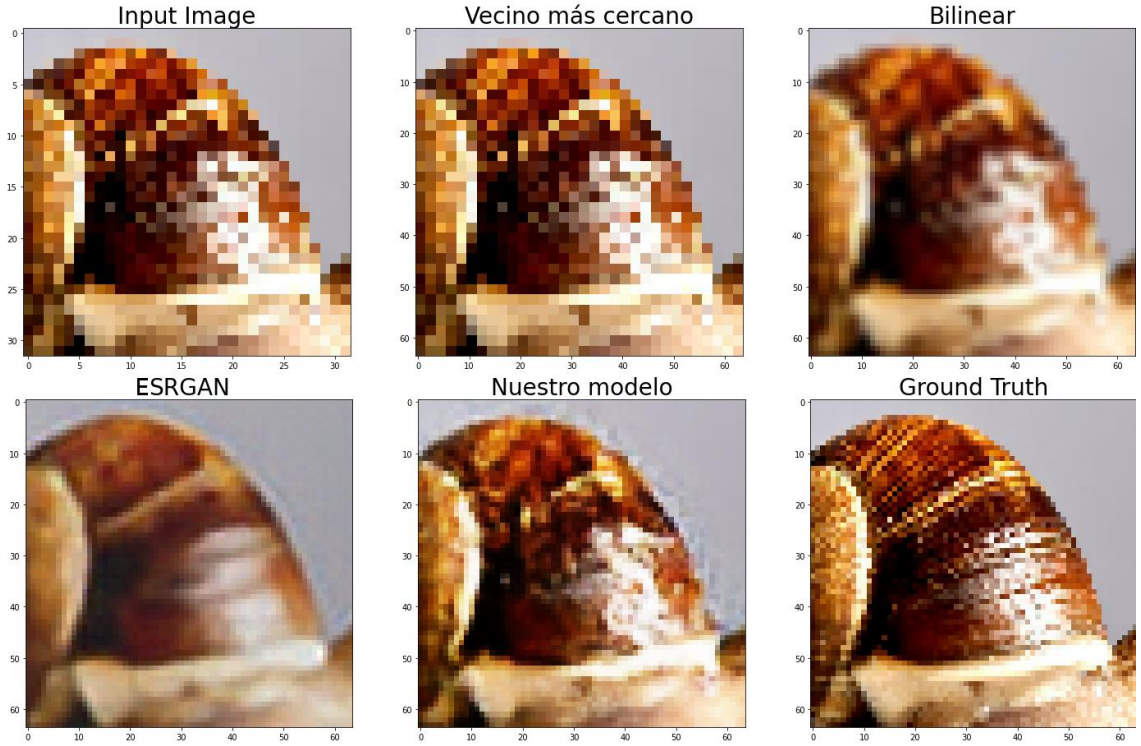


Ilustración 44 Comparativa visual entre las técnicas clásicas de reescalado, el modelo ESRGAN entrenado por el equipo de Tensorflow, y nuestro modelo.

De esta comparativa obtenemos las tres métricas, ya mencionadas, de cada una de las técnicas de superresolución aplicadas.

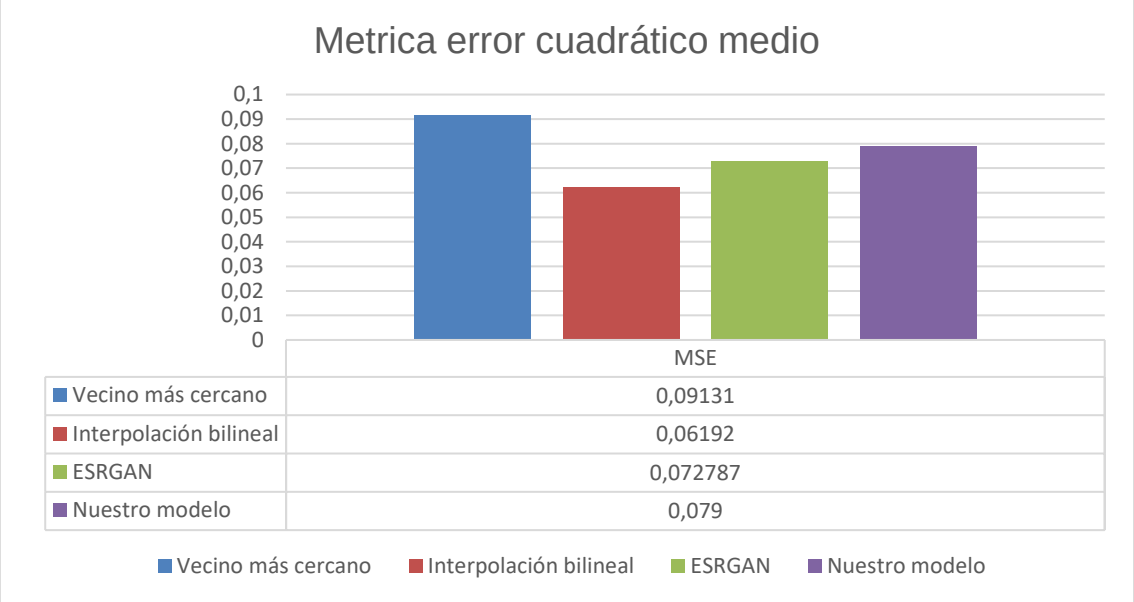


Ilustración 45 Métrica MSE resultados

En términos de la métrica del error cuadrático medio podemos observar que el error de nuestro modelo, 0,079, es ligeramente superior al error obtenido del

modelo de SRGAN, 0,073, entrenado por los desarrolladores de Tensorflow. Podemos observar que en esta situación el método bilineal tiene un menor error que el resto de los métodos, por lo que podemos asumir con el error de los modelos neuronales que han tenido un bajo rendimiento en las predicciones realizadas.

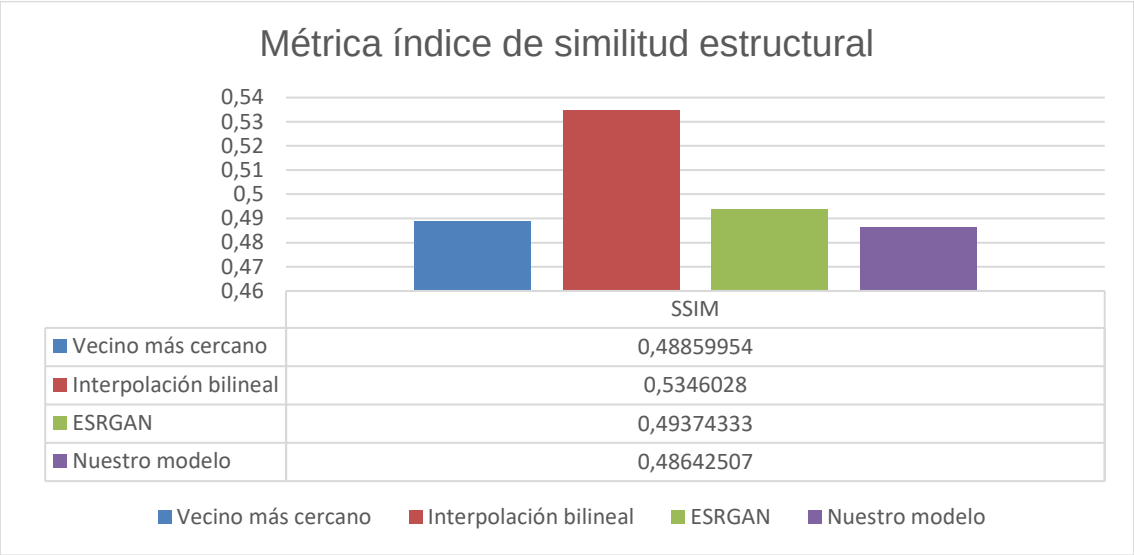


Ilustración 46 Métrica SSIM resultados.

En este caso podemos observar que nuestro modelo tiene un menor índice de similitud estructural. Obtenemos para el modelo de SRGAN un valor de 0,49374 y de 0,48642 para nuestro modelo. Podemos asumir con estos valores que nuestro modelo tiene una peor calidad de imagen respecto al modelo SRGAN según esta métrica.

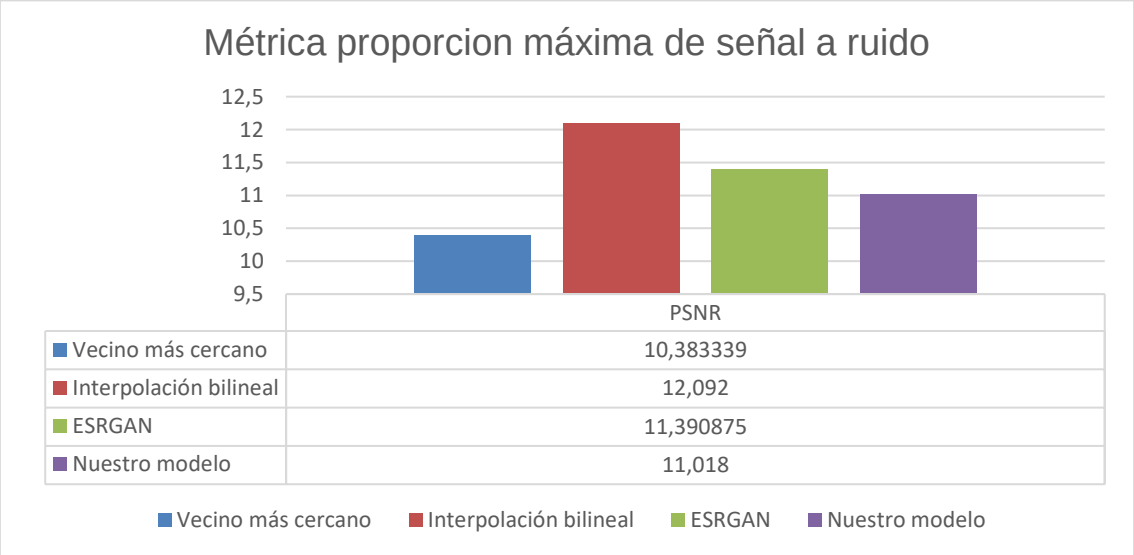


Ilustración 47 Métrica PSNR resultados.

En este caso podemos observar que nuestro modelo tiene una mayor proporción máxima de señal de ruido comparado con el modelo SRGAN. El valor de nuestro modelo 11,9374. aunque el valor parezca mostrar que en esta métrica nuestra red es la que mejores resultados da, debemos tener en cuenta que el valor estándar para una imagen con buena calidad y compresión está en el baremo de los valores 20 y 30.

Presupuesto

Recursos del proyecto

El proyecto tiene recursos de tipo trabajo y tipo material.

Trabajo

Un desarrollador encargado de modelar la red neuronal y desarrollar la aplicación Android.

Material

El ordenador de sobremesa utilizado para el desarrollo del proyecto, con los principales componentes:

- GPU: NVIDIA RTX 2070
- CPU: AMD Ryzen 5600x

Gastos

Los gastos del proyecto se pueden dividir en gastos directos e indirectos. En la partida de los gastos directos se incluyen los gastos de los recursos principales usados en el proyecto, mientras que en la partida de los indirectos se incluirán los gastos derivados de los directos.

Gastos directos

Costes de personal del proyecto

El valor del coste total del del personal del proyecto se ha obtenido en base a los cálculos del documento de MS Project adjuntado en el trabajo.

Costes de personal		
Personal	Precio / hora	Coste total
Desarrollador	30,00 €	11580,00€

Tabla 1 Costes de personal del proyecto.

Costes de los medios de producción

Otro de los gastos del proyecto es el valor de los medios de distribución utilizados. Para ello debemos de tener en cuenta que los medios de producción del proyecto consisten en un ordenador de sobremesa y que es un activo fijo ya que tiene una vida útil estimada.

Se debe tener en cuenta varios factores al calcular este coste:

- El coste inicial del ordenador de sobremesa es de 1200 euros y no se compró para este proyecto, por lo que únicamente se calculará el valor de amortización.
- La vida útil del ordenador de sobremesa está estimada en 10 años.
- El ordenador ya ha consumido 3 años de su vida útil.
- El ordenador se ha utilizado durante 7 meses para este proyecto.

Calculamos el coste de amortización del ordenador usado:

$$Amortización = Precio * (meses\ de\ uso\ \div\ Meses\ de\ vida\ útil) =$$

$$1200 * \left(\frac{7}{120}\right) = 70€ \quad (11)$$

Obtenemos un coste de 70€ de los medios de producción utilizados durante el periodo que ha sido utilizado.

Gastos indirectos

El principal gasto indirecto del proyecto es la electricidad consumida por los medios de producción usados durante el desarrollo del proyecto.

Haciendo uso de la calculadora de consumos de Outervision obtenemos que, para la configuración del ordenador de sobremesa usado, el consumo medio es de 372 vatios.

La media de horas por día se ha establecido como 8 teniendo en cuenta que hay varios factores variables que no siempre se cumplieron en las jornadas diarias al realizar el proyecto, como el número de horas de entrenamiento de cada prueba realizada.

Para facilitar los cálculos en el territorio de España, ajustaremos el precio de la luz a unos 0.163 kWh.

Usando la siguiente formula de consumo podemos obtener fácilmente el gasto del ordenador durante el proyecto:

$$Coste\ € = Vatios_{PC} \bullet Horas_{día} \bullet Precio_{kWh} \quad (12)$$

Aplicando la formula obtenemos un gasto de 0,364 euros por cada jornada de 8 horas.

Costes indirectos		
Servicio	Coste por día	Coste duración del proyecto
Electricidad	0,364€	77,49€

Tabla 2 Costes indirectos del proyecto.

Gasto total

Teniendo en cuenta los gastos directos e indirectos podemos estimar un gasto total de 11.727,49 euros durante el desarrollo del proyecto.

Conclusiones

La solución del problema de superresolución en redes neuronales está siendo uno de los campos más interesantes de los últimos años en los estudios de inteligencia artificial, la mayor prueba de ello es la cantidad de proyectos existentes que aportan soluciones variadas que difieren unas de otras y aportan novedosos resultados.

Todas las métricas usadas para las comparativas nos muestran que el modelo ESRGAN y nuestro modelo comparten valores muy similares en las distintas métricas. En cambio, si visitamos la documentación oficial del proyecto ESRGAN podemos obtener esta imagen con la métrica PSNR comparando su modelo con otros.

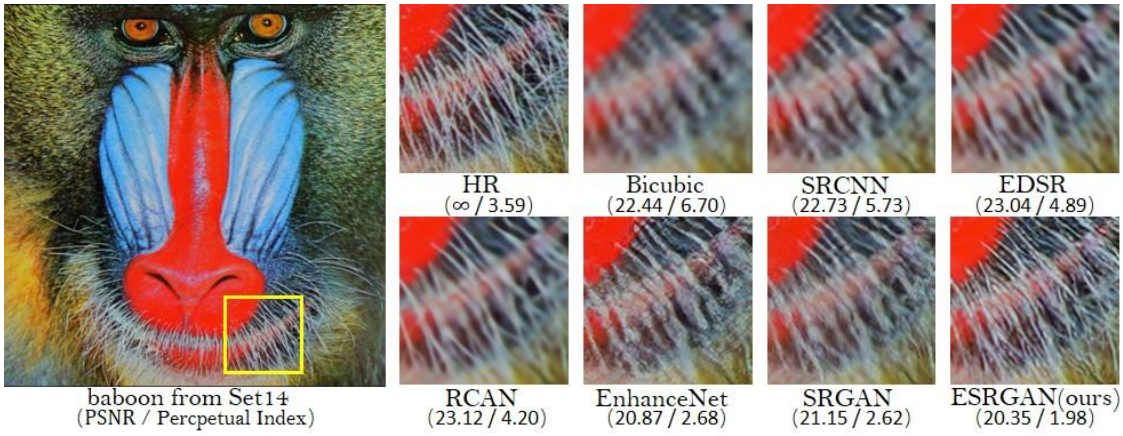


Ilustración 48 Comparativa de resultados de ESRGAN, original del artículo del modelo ESRGAN [8].

Si regresamos a la comparativa realizada en la sección “Comparativa de los modelos”, el modelo ESRGAN conseguía un valor de la métrica PSNR de 11,391. El valor de la métrica PSNR realizada por el equipo desarrollador de ESRGAN dobla el valor obtenido que se ha obtenido para nuestra comparativa, e incluso la imagen a primera vista tiene mucha mayor calidad perceptual.

Esta diferencia puede deberse al número de épocas de entrenamiento e hiperparámetros que han elegido los diferentes equipos que han entrenado el modelo. Por ello podemos concluir que nuestro modelo tiene un rendimiento estadístico similar a los resultados del entrenamiento del modelo SRGAN hecho por el equipo de Tensorflow, pero en cambio difiere y tiene menor calidad que el modelo entrenado por el equipo que desarrolló originalmente el modelo. Sin embargo, es un modelo mucho más ligero, capaz de correr sobre dispositivos móviles u otros con escasos recursos computacionales y de memoria.

Retornando a los objetivos del proyecto, podemos concluir ciertos aspectos comparándolos con el trabajo realizado:

- Se han analizado diferentes modelos junto a aplicaciones reales de estos para poder determinar cuál era la solución más adecuada a aplicar. Dando lugar a la creación de un modelo capaz de reescalar una imagen al doble de su tamaño en cada dimensión. Podemos afirmar que los objetivos OI1 y OI2 de la sección “Objetivos de la investigación” fueron cumplidos satisfactoriamente.
- Se ha obtenido un error cuadrático medio del 7,3% en nuestro modelo, como se puede observar en la comparativa realizada en el capítulo “Pruebas y resultados”, “Comparativa de los modelos”. Este porcentaje de error está en el rango inicial esperado en el objetivo OR1 de la sección “Objetivos de la red neuronal”, se cumple este objetivo.
- La cantidad de nodos de la red encargada de reescalar la imagen es mucho menor que otros modelos ya existentes, facilitando su exportación a dispositivos móviles. El objetivo OR2 del desarrollo de la red neuronal, de la sección “Objetivos de la red neuronal”, se cumple correctamente.
- Se han conseguido implementar todas las funcionalidades de la aplicación Android marcadas en la lista de requisitos de la sección “Requisitos funcionales del sistema Android”. La aplicación es capaz de reescalar la imagen seleccionada por el usuario, de la galería o cámara, y guardar el resultado. También ofrece la posibilidad de limitar el tamaño de los segmentos en los que se procesa la imagen y si se desea ejecutar sobre la GPU.

Existen varios puntos que considero de vital importancia en mi experiencia durante la elaboración de este proyecto de investigación:

- Este proyecto de investigación me ha generado un interés en estas nuevas ramas del aprendizaje profundo de redes y he conseguido ver desde otra perspectiva conceptos básicos sobre inteligencia artificial aprendidos durante la carrera.
- Mis habilidades con lenguajes como Python o C++ se han visto mejoradas, pues me he tenido que enfrentar a librerías de código abierto en entornos que no esperaría hacerlo. Séase un buen ejemplo el uso de una implementación de código C++ en la aplicación de dispositivos móviles.
- Por último, pero no menos importante, he de considerar que he aprendido la importancia de documentar las tareas que se van realizando. Este proyecto ha sido realizado en un rango de 2 años y durante los primeros meses, aunque muchas ideas fuesen desechadas,

no apunté de forma exhaustiva el trabajo realizado. Todo esto conllevó a una mayor dificultad en la redacción de esta memoria.

Con el trabajo realizado de investigación sobre el problema de superresolución aplicado a redes neuronales puedo afirmar que hay un gran potencial en las tecnologías usadas en este campo.

Las redes generativas adversarias presentan un claro potencial pues, como se comentó a lo largo del documento, cada vez existen más y mejores proyectos que solucionan este problema e incluso pueden llegar a tener aplicaciones importantes como en el campo de la medicina. Pese a este gran potencial, estos tipos de modelos adversarios siguen teniendo a día de hoy el problema de que el entrenamiento que necesitan tiene un gran coste computacional y abarca un gran periodo de tiempo.

Aun así, no me cabe la menor duda de que más pronto que tarde acabaremos viendo una evolución de este tipo de redes convolucionales que solucione este problema temporal y de computación para llevar a un nuevo nivel la solución del problema de superresolución con redes generativas adversarias, como ya vimos en la introducción con la evolución de las redes transformer que solucionaban los problemas de las redes neuronales recurrentes.

Anexo I: Ficheros adicionales

La imagen generada por el modelo ESRGAN usada en el sección “Comparativa de los modelos” se obtuvo de un cuaderno de Jupyter Notebooks, alojado en el portal Google Colab. La imagen usada en la comparativa se adjunta en los archivos del proyecto con el nombre “input.jpg”. El siguiente enlace da acceso al código del cuaderno:

- *https://colab.research.google.com/drive/18Ik8LhKGaNYDpak0igF-Pbx0brUUaQ-1#scrollTo=lnyLTyUt0ukN*

Toda la parte del trabajo del desarrollo de la red neuronal se ha desarrollado en el sistema Linux-Ubuntu 18.04.6 LTS. Para el desarrollo del cuaderno del modelo se usó como plataforma la distribución de código abierto Anaconda. Para facilitar la ejecución del cuaderno se adjunta también un archivo “environment.yml” y un script “.sh” para copiar la configuración del entorno en el que se comprobó el correcto funcionamiento del código, el comando ha de ejecutarse desde el terminal de Anaconda. El entorno del archivo “environment.yml” también se puede importar desde la interfaz de usuario.

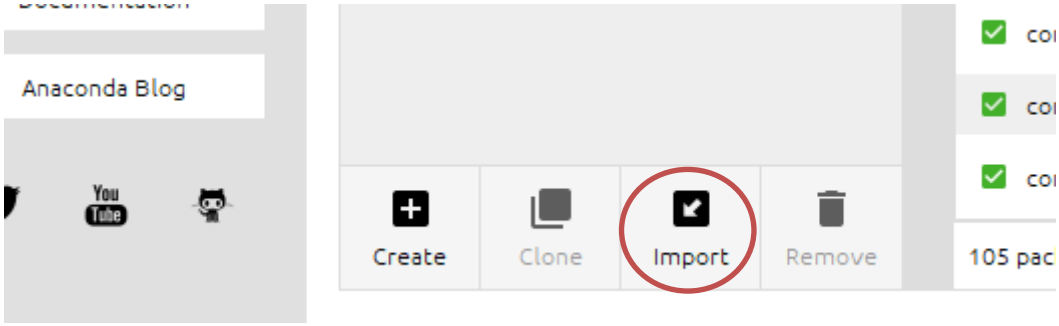


Ilustración 49 Imporatación entorno Anaconda

La aplicación Android está subida al SharePoint de la universidad por el límite de tamaño de la entrega, el enlace es el siguiente:

- https://unioviedo-my.sharepoint.com/:u:/g/personal/uo264207_uniovi_es/EeuwTsRfNN1Mi6-YIXe99PQBBpg8-VwZgHWioYIwDb_BuQ?e=bQ0YyE

La aplicación necesita también un par de configuraciones. Antes de ejecutar la aplicación es necesario instalar en Android Studio la herramienta Cake.

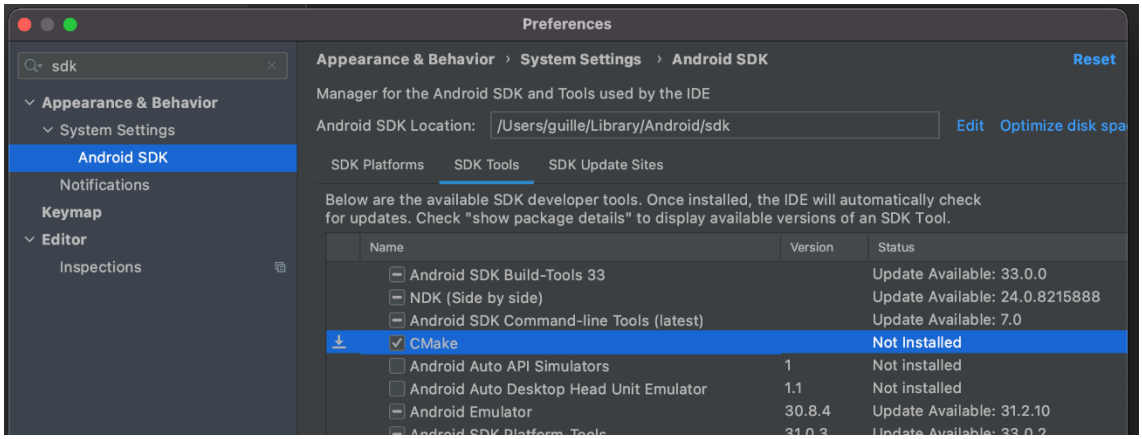


Ilustración 50 Instalación CMake

Después de instalar CMake hay que poner la ruta del SKD, NDK y de CMake en el archivo local.properties. La ruta del NDK puede no ser necesaria si se instala el NDK a través de Android Studio en vez de hacerlo manualmente.

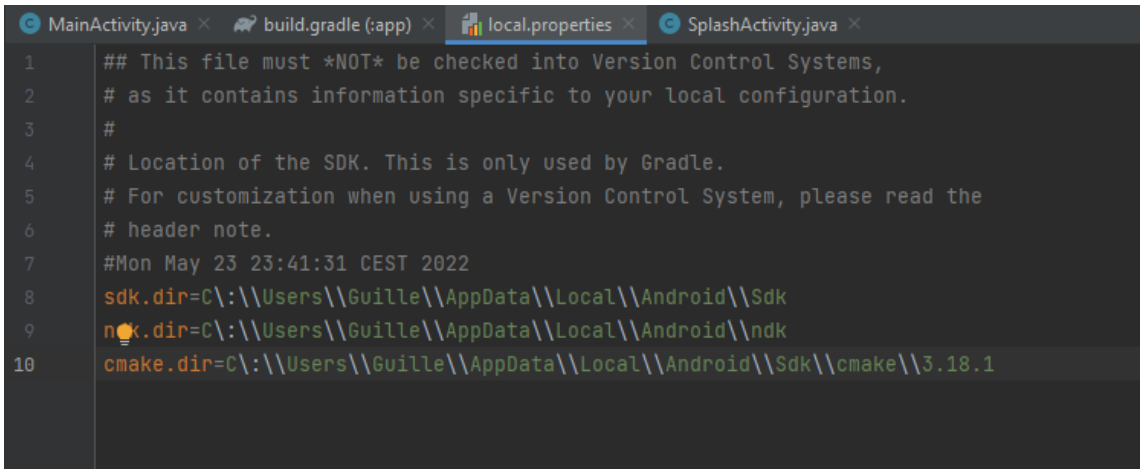


Ilustración 51 Rutas locales

Bibliografía

[1] I. Norman, "Enhance! A Practical Superresolution Tutorial in Adobe Photoshop," 20 2 2015. [Online]. Available: http://photoncollective.com/enhance-practical-superresolution-in-adobe-photoshop.

[2] M. Chu, "Temporally Coherent GANs for Video Super-Resolution (TecoGAN)," Arxiv-vanity.com, 2015. [Online]. Available: https://www.arxiv-vanity.com/papers/1811.09393/.

[3] L. X. C. D. Y. S. Xintao Wang, "Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data," 2021. [Online]. Available: https://arxiv.org/abs/2107.10833. [Accessed 30 6 2022].

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," arXiv.org, 12 06 2017. [Online]. Available: https://arxiv.org/abs/1706.03762v5.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, "Language Models are Few-Shot Learners," arXiv.org, 28 05 2020. [Online]. Available: https://arxiv.org/abs/2005.14165v4.

[6] L. Bouchard, "Tesla's Autopilot Explained," Louis Bouchard, 21 08 2021. [Online]. Available: https://www.louisbouchard.ai/tesla-autopilot-explained-tesla-ai-day/.

[7] C. Outeiral Rubiera, "AlphaFold 2 is here: what's behind the structure prediction miracle | Oxford Protein Informatics Group," Blopig.com, 19 07 2021. [Online]. Available: https://www.blopig.com/blog/2021/07/alphafold-2-is-here-whats-behind-the-structure-prediction-miracle/.

[8] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, C. C. Loy, Y. Qiao and X. Tang, "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks," arXiv.org, 01 10 2018. [Online]. Available: https://arxiv.org/abs/1809.00219.

[9] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken and A. Tejani, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," arXiv.org, 15 09 2016. [Online]. Available: https://arxiv.org/abs/1609.04802.

[10] C. Pere, "What are Loss Functions? - Towards Data Science," Medium, 17 06 2020. [Online]. Available: https://towardsdatascience.com/what-is-loss-function-1e2605aeb904#:~:text=The%20loss%20function%20is%20the,be%20categorized%20into%20two%20groups..

[11] Y. Jo, S. Yang and S. Kim, "Investigating Loss Functions for Extreme Super-Resolution," 2020. [Online]. Available: https://openaccess.thecvf.com/content_CVPRW_2020/papers/w31/Jo_Investigating_Loss_Functions_for_Extreme_Super-Resolution_CVPRW_2020_paper.pdf.

[12] Seb, "Stochastic Gradient Descent vs Mini Batch Gradient Descent vs Batch Gradient Descent," Programmathically.com, 06 11 2021. [Online]. Available: https://programmathically.com/stochastic-gradient-descent-versus-mini-batch-gradient-descent-versus-batch-gradient-descent/?utm_source=rss&utm_medium=rss&utm_campaign=stochastic-gradient-descent-versus-mini-batch-gradient-descent-versus-batch-gradient-des.

[13] "Principio de Pareto," Wikipedia.org, 24 05 2005. [Online]. Available:

https://es.wikipedia.org/wiki/Principio_de_Pareto.

[14] J. Vincent, "Artificial intelligence is helping old video games look like new," The Verge, 18 4 2019. [Online]. Available: https://www.theverge.com/2019/4/18/18311287/ai-upscaling-algorithms-video-games-mods-modding-esrgan-gigapixel.

[15] C. Dong, C. C. Loy and X. Tang, "Accelerating the Super-Resolution Convolutional Neural Network," arXiv.org, 01 08 2016. [Online]. Available: https://arxiv.org/abs/1608.00367.

[16] A. Jolicoeur-Martineau, "The relativistic discriminator: a key element missing from standard GAN," arXiv.org, 02 07 2018. [Online]. Available: https://arxiv.org/abs/1807.00734v3.

[17] L. Fei-Fei, J. Deng and K. Li, "ImageNet: Constructing a large-scale image database," Journal of Vision, 22 03 2010. [Online]. Available: http://wordnet.cs.princeton.edu/papers/imagenet_cvpr09.pdf.

[18] "Android heap size on different phones/devices and OS versions – iTecNote," Itecnote.com, 03 07 2022. [Online]. Available: https://itecnote.com/tecnote/android-heap-size-on-different-phones-devices-and-os-versions/.

[19] E. T. Lite, "Github Demo Apps Tensorflow Lite," Tensorflow Lite, [Online]. Available: https://github.com/tensorflow/examples/tree/master/lite/examples/super_resolution/android.

[20] A. Dey, "Súper resolución de imagen usando ESRGAN | TensorFlow Hub," TensorFlow, 2021. [Online]. Available: https://www.tensorflow.org/hub/tutorials/image_enhancing.

[21] M. Chu, Y. Xie, J. Mayer, L. Leal-Taixé and N. Thuerey, "Learning temporal coherence via self-supervision for GAN-based video generation," 31 8 2020. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3386569.3392457.

[22] B. J. Manuel, "Redes neuronales adversarias para el reconocimiento de malezas," [Online]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/65398/Documento_completo.pdf-PDFA.pdf?sequence=1.

[23] J. S. Isaac and R. Kulkarni, "Super resolution techniques for medical image processing," 2015 International Conference on Technologies for Sustainable Development (ICTSD), 2 2015. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7095900.

[24] Fast.ai, "Decrappification, DeOldification, and Super Resolution," 03 05 2019. [Online]. Available: https://www.fast.ai/2019/05/03/decrappify/.

[25] X. Wang, K. Yu, C. Dong and C. C. Loy, "Recovering Realistic Texture in Image Super-resolution by Deep Spatial Feature Transform," arXiv.org, 09 04 2018. [Online]. Available: https://arxiv.org/abs/1804.02815.

[26] A. Watson, "Deep Learning Techniques for Super-Resolution in Video Games," arXiv.org, 17 12 2020. [Online]. Available: https://arxiv.org/abs/2012.09810.

[27] S. Sharma, "Activation Functions in Neural Networks - Towards Data Science," Medium, 06 09 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[28] A. Radford, L. Metz and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," arXiv.org, 19 11 2015. [Online]. Available: https://arxiv.org/abs/1511.06434.

[29] T. Karras, S. Laine and T. Aila, "A Style-Based Generator Architecture for Generative

Adversarial Networks," arXiv.org, 12 12 2018. [Online]. Available: https://arxiv.org/abs/1812.04948.

[30] S. Sharma, "AI Can See Clearly Now: GANs Take the Jitters Out of Video Calls," NVIDIA Blog, 05 10 2020. [Online]. Available: https://blogs.nvidia.com/blog/2020/10/05/gan-video-conferencing-maxine/.

[31] "Redes Transformer, el fin de las Redes Recurrentes," Codificando Bits, 30 06 2020. [Online]. Available: https://www.codificandobits.com/blog/redes-transformer/.

[32] N. S. C. "Loss Functions in Neural Networks," The AI dream, 02 08 2021. [Online]. Available: https://www.theaidream.com/post/loss-functions-in-neural-networks.

[33] D. Gaur, J. Folz and A. Dengel, "Training Deep Neural Networks Without Batch Normalization," arXiv.org, 18 08 2020. [Online]. Available: https://arxiv.org/abs/2008.07970.

[34] S. Park, "Is Batch Normalization harmful? Improving Normalizer-Free ResNets," Medium, 30 07 2021. [Online]. Available: https://medium.com/geekculture/is-batch-normalization-harmful-improving-normalizer-free-resnets-cf44f2fc0b2e.

[35] P. Larrañaga, "Redes Neuronales," [Online]. Available: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

[36] C. Zhen, X. Guo, P. Yat and Y. Yuan, "Super-Resolution Enhanced Medical Image Diagnosis With Sample Affinity Interaction," ResearchGate, 28 01 2021. [Online]. Available: https://www.researchgate.net/publication/348851387_Super-Resolution_Enhanced_Medical_Image_Diagnosis_With_Sample_Affinity_Interaction.

[37] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," arXiv.org, 18 05 2015. [Online]. Available: https://arxiv.org/abs/1505.04597.

[38] L. Yue, H. Shen, J. Li, Q. Yuan, H. Zhang and L. Zhang, "Image super-resolution: The techniques, applications, and future," ScienceDirect, 11 2016. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0165168416300536.

[39] "Aplicaciones de la inteligencia artificial en las empresas | ABAMobile," ABAMobile, 28 04 2021. [Online]. Available: https://abamobile.com/web/aplicaciones-de-la-inteligencia-artificial-en-las-empresas/.

[40] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert and Z. Wang, "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," arXiv.org, 16 09 2016. [Online]. Available: https://arxiv.org/abs/1609.05158.

[41] "The Dying ReLU Problem, Clearly Explained - Towards Data Science," Medium, 30 03 2021. [Online]. Available: https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24#4995.

[42] I. J. Goodfellow, "Generative Adversarial Networks," arXiv.org, 10 06 2014. [Online]. Available: https://arxiv.org/abs/1406.2661.

[43] K. Shen, "Effect of batch size on training dynamics," Medium, 19 06 2018. [Online]. Available: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e.

[44] K. You, "How Does Learning Rate Decay Help Modern Neural Networks?," arXiv.org, 05 08 2019. [Online]. Available: https://arxiv.org/abs/1908.01878.

[45] "Leaky ReLU Explained," Paperswithcode.com, 2020. [Online]. Available: https://paperswithcode.com/method/leaky-relu.

[46] "T91 Image Dataset," Kaggle.com, 2019. [Online]. Available: https://www.kaggle.com/datasets/ll01dm/t91-image-dataset.

[47] N. Adaloglou, "Intuitive Explanation of Skip Connections in Deep Learning | AI Summer," AI Summer, 23 03 2020. [Online]. Available: https://theaisummer.com/skip-connections/.