VNIVERSITAT DE VALÈNCIA

Treball Fi de Grau - Curs 2021/ 2022

Síntesis de geometrías de aortas usando redes generativas antagónicas.

Autor: **Álvaro López López**

Tutores: **Ignacio García Fernández y Pau Romero de Antonio**

Facultat de Ciències Matemàtiques

Grau en Matemàtiques

Índice general

Capítulo 1

Introducción

La arteria aorta es la principal arteria de nuestro cuerpo. Esta directamente conectada con el ventrículo izquierdo de nuestro corazón y se encarga de transportar la sangre rica en oxígeno y es desde ella de donde se generan gran parte de las arterias de nuestro cuerpo. Su trazado es ascendente los primeros centímetros y luego hace un arco que dirige la aorta de forma descendente pasando por el tórax y llegando hasta la zona abdominal. Su longitud media es de 30 cm, y el diámetro varia a lo largo de ella entre 2.5 cm y 4 cm. Cuando el ventrículo izquierdo se contrae, expulsa la sangre por la aorta cuya pared tiene cierta elasticidad y flexibilidad. Esto permite mantener una presión sanguínea.

Podemos dividir las partes de la aorta en: aorta ascendente, arco aórtico, aorta descendente, aorta torácica y aorta abdominal.

Hay diversas enfermedades relacionadas con esta arteria, pero nosotros trataremos el aneurisma aórtico. Esta enfermedad se da cuando un área de la pared de la aorta esta debilitada o sufre una dilatación y puede darse a lo largo de toda su longitud. Este cambio en el diámetro de la aorta puede producirse por diversos factores como arteriosclerosis, traumatismos o infecciones. Se considera que una arteria tiene un aneurisma cuando se ha dilatado mas de 1.5 veces su diámetro. Cuando la pared de la aorta esta débil, al ser bombeada la sangre, la presión puede ocasionar una ruptura. Esto genera intensos dolores y una hemorragia que sin la intervención conveniente ocasiona la muerte. Solo en 2018, en Estados Unidos, los aneurismas aórticos causaron casi 10000 muertes [7]. Por ello, es muy importante ser capaces de generar un diagnóstico precoz correcto.

La falta de datos en el campo de la bioingeniería es un problema a día de hoy. Esto se debe al alto coste que supone obtener esta información. Es por ello que se utilizan muchos algo-

ritmos para aumentar el tamaño de las bases de datos, conocidos como algoritmos de *data augmentation*.

Los algoritmos de inteligencia artificial y en concreto los algoritmos de machine learning tienen la capacidad de aprender patrones sin que hayan sido programados explícitamente. Estos reciben información que analizan y se ajustan para predecir unos valores de salida. Hay dos grandes grupos de algoritmos de machine learning: aprendizaje supervisado, aprendizaje no supervisado.

Con los algoritmos de aprendizaje supervisado, los datos de entrada están etiquetados y el algoritmo utiliza estas etiquetas para que a posteriori con entradas similares obtengamos salidas acorde a las etiquetas.

A diferencia de los supervisados, los no supervisados no tienen los datos etiquetados y es el propio algoritmo quien tiene que detectar los patrones, organizando los datos tratando de describir la estructura de los mismos.

Este trabajo consistirá en el ajuste de un modelo de inteligencia artificial conocido como Generative Adversarial Networks (GAN) [5], capaz de generar aortas sintéticas con aneurisma. Estas redes son algoritmos de aprendizaje supervisado, utilizaremos diversas bases de datos de imágenes médicas y entrenaremos los algoritmos. Entraremos en detalles teóricos de los mismos partiendo de una explicación sobre los algoritmos de machine learning, llegando hasta redes neuronales convolucionales.

Nuestro objetivo es ver si al utilizar las capas convolucionales obtenemos una red neuronal con una mayor eficiencia generando aortas. Para saber la eficiencia de nuestro modelo, generaremos aortas sintéticas y realizaremos mediciones a lo largo de las aortas en lo que llamamos biomarcadores. Generaremos cierta cantidad de aortas sintéticas con diversas GANs convolucionales y las compararémos con las aortas de la base de datos original y unas que fueron generadas en el paper *Clinically-Driven Virtual Patient Cohorts Generation: An Application to Aorta* [15] a través de una GAN sin capas convolucionales con una base de datos a la que se le había hecho un análisis de componentes principales (PCA).

Capítulo 2

Parametrización de la arteria aorta

La arteria aorta comienza directamente desde el corazón, exactamente en el ventrículo izquierdo, separados por las valvas semilunares. Tiene una longitud aproximada de 30 cm, de hecho, es la arteria mas grande del cuerpo humano, cuya función principal es transportar la sangre oxigenada al resto del cuerpo. Su diámetro varía desde los 2.5 cm en la aorta abdominal, pasando por 3.5 cm en la aorta ascendente y hasta los 4 cm en la raíz aórtica.

1. La aorta ascendente es el segmento que empieza en la base del ventrículo izquierdo y termina antes de las primeras ramificaciones.
2. Arco aórtico, es la continuación de la aorta ascendente. Comienza en la primera ramificación y continua hasta que arquea por completo dirigiendo la aorta de forma descendente.
3. La aorta descendente es la parte más grande que constituye la aorta. Esta dividida en la aorta torácica y la aorta abdominal por el diafragma.

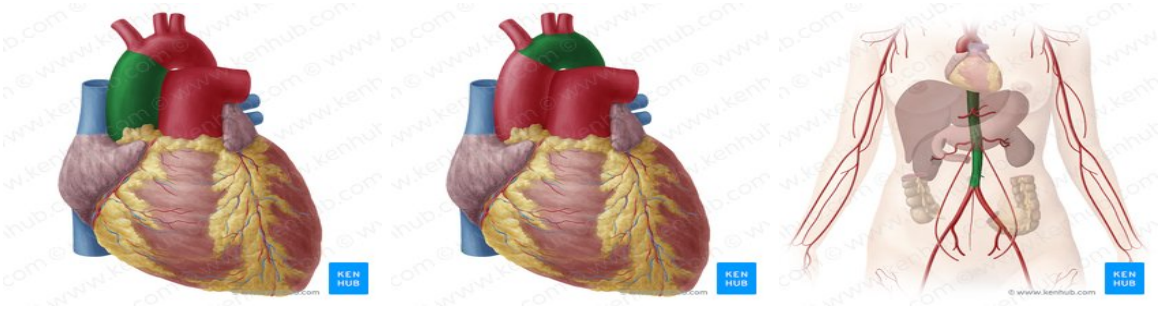


Figura 2.1: La sección de color verde es la aorta ascendente, arco aórtico y aorta descendente respectivamente [11].

Una de las enfermedades asociadas a esta arteria es el aneurisma. Esta enfermedad se da cuando la arteria sufre un abultamiento o una zona de las paredes esta frágil. Esto puede ocasionar

rupturas, las cuales se tienen que ser operadas de urgencia. Un aneurisma se puede generar por diversas causas y dar un diagnostico precoz es un importante.

2.1. Parametrización de la aorta

Para poder atajar el problema, necesitamos tener una parametrización de la aorta que sea única e invertible, es decir, que dada una aorta obtengamos una parametrización y que si tenemos una parametrización, tengamos una única aorta. A lo largo de este capítulo haremos una pequeña introducción a algunos conceptos básicos de Geometría Diferencial Clásica que nos servirán para realizar esta parametrización.

Nuestro objetivo es obtener una función $\overrightarrow{x} : \Omega \subseteq \mathbb{R}^2 \longrightarrow \mathbb{R}^3$ que parametrize la superficie de la pared aorta.

Definición 1. Diremos que una función $\alpha : [a, b] \subseteq I \longrightarrow \mathbb{R}^n$ es una curva diferenciable si es una función diferenciable del intervalo abierto Ω en $\mathbb{R}^n$

Definición 2. Llamaremos por centerline a la curva diferenciable $\alpha(s)$ que recorre el centro de la aorta.

$$\alpha : [a, b] \subseteq I \longrightarrow \mathbb{R}^3 \tag{2.1}$$

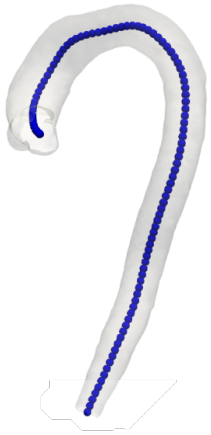


Figura 2.2: Centerline de una aorta

Definición 3. El vector tangente de una curva $\alpha(s)$, en el punto s, se define como:

$$\overrightarrow{t}(s) = \dot{\alpha}(s) \tag{2.2}$$

Definición 4. Una vez tenemos el vector tangente, podemos definir el vector normal de una curva $\alpha(s)$, en el punto s como:

$$\overrightarrow{n} = \frac{\dot{\overrightarrow{t}}(s)}{\|\overrightarrow{t}(s)\|} \tag{2.3}$$

Definición 5. Definimos el vector binomial de una curva $\alpha(s)$, en el punto s como:

$$\overrightarrow{b}(s) = \overrightarrow{t}(s) \times \overrightarrow{n}(s) \tag{2.4}$$

También será de nuestro interés tener una medida del cambio de dirección del vector tangente a lo largo de la curva. Para ello utilizaremos la curvatura.

Definición 6. Definimos el vector curvatura de una curva $\alpha(s)$ como:

- Para una curva cualquiera:

$$\overrightarrow{k}(s) = \frac{\|\alpha'(s) \times \alpha''(s)\|}{\|\alpha(s)\|^3} \tag{2.5}$$

- Si la curva viene parametrizada la longitud del arco:

$$\overrightarrow{k}(s) = \|\alpha''(s)\| \tag{2.6}$$

Al igual que nos pasa con el vector tangente, también queremos saber como cambia la dirección del vector binomial. Es por ello que definimos la torsión.

Definición 7. La torsión:

- En el caso general, calculamos la torsión como:

$$\tau(t) = -\frac{\alpha'(t) \cdot (\alpha''(t) \times \alpha'''(t))}{\|\alpha'(t) \times \alpha''(t)\|^2} \tag{2.7}$$

- Si la curva viene parametrizada por la longitud del arco:

$$\tau(s) = \frac{\alpha'(s) \cdot (\alpha''(s) \times \alpha'''(s))}{\|\alpha''(s)\|^2} \tag{2.8}$$

Definición 8. Los tres vectores $\{\overrightarrow{t}(s), \overrightarrow{n}(s), \overrightarrow{b}(s)\}$, forman una base ortonormal de $\mathbb{R}^3$, cuya orientación es la misma que la de la base canónica. A esta base la llamaremos por triedro de Frenet.

Definición 9. Dado un punto $\alpha(s)$, definimos el plano normal sobre la curva α, como el plano definido por los vectores $\{\overrightarrow{n}(s), \overrightarrow{b}(s)\}$. Este plano es perpendicular a la curva en el punto $\alpha(s)$.

Ahora que ya tenemos los conceptos teóricos necesarios, vamos a relacionarlos para construir nuestra parametrización. Dado el centerline $\alpha(s)$, tenemos que para cada punto del centerline, podemos obtener la triedro de Frenet. A nosotros nos interesa el plano normal a ese punto ya que ese plano cortará con la pared de la aorta generando una curva cerrada que la llamaremos como la sección transversal del punto $\alpha(s)$.

Cada una de las secciones transversales se puede parametrizar en coordenadas polares, teniendo en cuenta la base del plano normal, $\{\overrightarrow{n}(s),\ \overrightarrow{b}(s)\}$ y el punto $\alpha(s)$. Consideramos la semirrecta que une el punto $\alpha(s)$ con el punto de la pared en la dirección $\overrightarrow{n}(s)$. A partir de ahí, podemos parametrizar la sección transversal en polares en un punto s con la ecuación:

$$seccion\_transversal(s,\theta) = \alpha(s) + \rho(s,\theta)(\overrightarrow{n}(s)\cos\theta + \overrightarrow{b}(s)\sin\theta), \quad \theta \in [0, 2\pi] \tag{2.9}$$

siendo $\rho(s,\theta)$ la distancia de $\alpha(s)$ a la pared de la aorta con ángulo θ.

Podríamos pensar en parametrizar la pared de la aorta como:

$$\overrightarrow{x}(s,\theta) = \alpha(s) + \rho(s,\theta)(\overrightarrow{n}(s)\cos\theta + \overrightarrow{b}(s)\sin\theta), s \in [0,1], \theta \in [0, 2\pi] \tag{2.10}$$

Sin embargo, a pesar de la efectividad de usar la orientación del triedro de Frenet para generar las secciones transversales, no puede ser utilizada para construir la pared al rededor del centerline debido a los bruscos cambios de orientación de la triedro de Frenet como se observa en la Figura 2.3 extraída del paper *A Framework for Geometric Analysis of Vascular Structures: Application to Cerebral Aneurysms* [5].

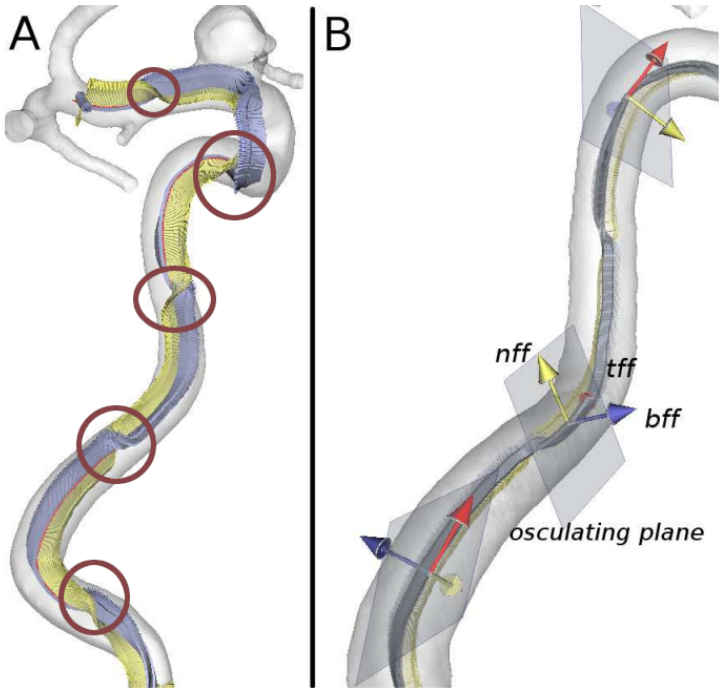


Figura 2.3: A. Triedro de Frenet a lo largo del centerline de la arteria carótida; B. Ampliación de la imagen A mostrando el plano osculador rotando a lo largo del centerline; siendo tff, nff, bff los vectores tangente, normal y binormal respectivamente. [14]. En las zonas marcadas se pueden observar los cambios bruscos en la orientación.

Es por ello que tenemos que trabajar con una base que no genere cambios bruscos en su orientación, para solventarlo trabajaremos con el transporte paralelo como hacen en el paper *A Framework for Geometric Analysis of Vascular Structures: Application to Cerebral Aneurysms* [14].

Definición 10. Sea S una superficie parametrizada y sea $\alpha : I \longrightarrow S$ una curva diferenciable. Definimos un campo vectorial tangente a S al largo de la curva α como una aplicación diferenciable $\overrightarrow{v}(t) : I \longrightarrow \mathbb{R}^3$ tal que para todo $t \in I$, $\overrightarrow{v}(t) \in T_{\alpha(t)}S$.

Definición 11. Definimos la norma unitaria a lo largo de una superficie parametrizada por $\overrightarrow{x}(u, v)$ como el vector $\overrightarrow{N} : U \subseteq \mathbb{R}^2 \longrightarrow \mathbb{R}^3$ tal que:

$$\overrightarrow{N} = \frac{\overrightarrow{x}_u \wedge \overrightarrow{x}_v}{\|\overrightarrow{x}_u \wedge \overrightarrow{x}_v\|} \tag{2.11}$$

Definición 12. Sea S una superficie parametrizada regular, $\alpha : I \longrightarrow S$ una curva diferenciable y $\overrightarrow{N}$ la norma unitaria de S. Definimos la derivada covariante, $\frac{D\overrightarrow{\mathbf{v}}}{dt}(t)$ de un campo vectorial

tangente a lo largo de α, $\overrightarrow{\mathbf{v}}$, como la proyección sobre el plano tangente de la derivada, es decir:

$$\frac{D\overrightarrow{\mathbf{v}}}{dt}(t) = \frac{d\overrightarrow{\mathbf{v}}}{dt}(t) - < \frac{d\overrightarrow{\mathbf{v}}}{dt}(t), \overrightarrow{N}(\alpha(t)) > \overrightarrow{N}(\alpha(t)) \tag{2.12}$$

Definición 13. Diremos que un campo vectorial tangente a lo largo de una curva α es paralelo si su derivada covariante es 0.

Definición 14. Sea S una superficie regular, $\alpha : I \longrightarrow S$ una curva diferenciable, $t_0, t_1 \in I, p = \alpha(t_0), q = \alpha(t_1)$ y $\overrightarrow{v_0} \in T_pS$. Sea $\overrightarrow{\mathbf{v}}$ el campo vectorial tangente a lo largo de la curva α y paralelo que cumple que $\overrightarrow{\mathbf{v}}(t_0) = \overrightarrow{v_0}$, que sabemos que es único.

Definimos el transporte paralelo del $\overrightarrow{v_0}$ al punto q a lo largo de la curva α como el vector $\overrightarrow{\mathbf{v}}(t_1) \in T_pS$ y lo denotaremos como:

$$T^{\alpha}_{pq}(\overrightarrow{v_0}) = \overrightarrow{\mathbf{v}}(t_1) \tag{2.13}$$

De forma similar a como se explica en el paper *A Framework for Geometric Analysis of Vascular Structures: Application to Cerebral Aneurysms*[14], podemos escoger una base ortonormal a través del centerline sin que se vea afectada por la torsión de la misma. En cada punto, los vectores unitarios que componen la base se rotan por la cantidad exacta que la curvatura requiere, minimizando así posibles cambios en la dirección.

De la misma forma que habíamos obtenido la sección transversal utilizando la triedro de Frenet, lo podemos hacer con la nueva base utilizando el transporte paralelo.

$$seccion\_transversal(s, \theta) = \alpha(s) + \rho(s, \theta)(v_1(s)\cos\theta + v_2(s)\sin\theta), \theta \in [0, 2\pi] \tag{2.14}$$

Y obtendríamos la parametrización de la pared aórtica como:

$$\overrightarrow{x}(s, \theta) = \alpha(s) + \rho(s, \theta)(v_1(s)\cos\theta + v_2(s)\sin\theta), s \in [0, 1], \theta \in [0, 2\pi] \tag{2.15}$$

siendo v_1 y v_2 los vectores que vamos escogiendo a través del transporte paralelo. Evitando cambios bruscos en la orientación de la base como se muestra en la Figura 2.4 en el caso de una arteria carótida.

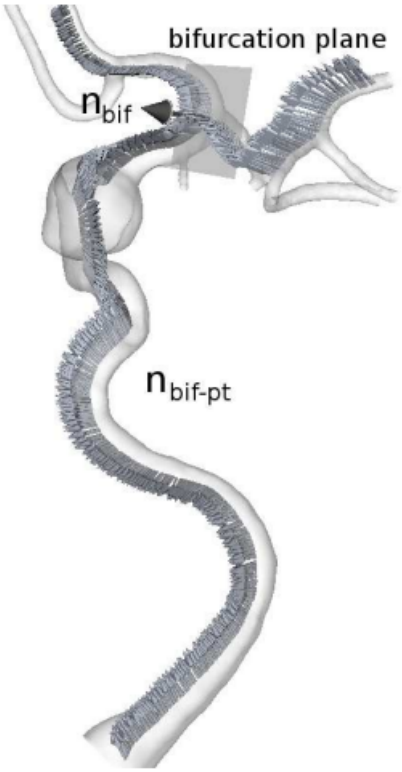


Figura 2.4: Una base transportada paralelamente a lo largo del centerline de una arteria carótida. [14]

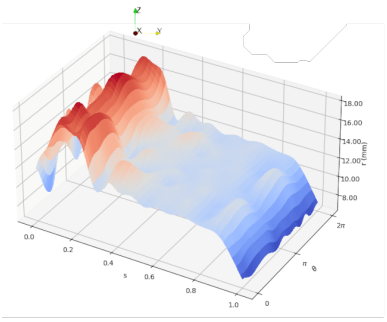
En la Figura 2.4 se puede observar que la base que utilizamos ya no tiene cambios bruscos en la orientación, es por ello que nos permitirá tener una mejor representación.

Capítulo 3

Representación de la aorta por medio de B-Splines

Una imagen médica de una aorta tiene mas de 350.000 vértices, y además, estos no son constantes, es decir, cada arteria tendrá una cantidad diferente de vértices. Esto supone un gran problema ya que tenemos una gran dimensionalidad en nuestro problema y no constante. La parametrización que hemos comentado en la sección 2.1, es una parametrización continua que no podemos representar computacionalmente debido a la memoria de los ordenadores, hay que discretizarla. Es por ello que necesitamos aproximar las arterias de forma que obtengamos una dimensionalidad menor pero que sea lo suficientemente significativa como para no perder información y además que sea constante para poder trabajar mejor con nuestro modelo de *Machine Learning*. Necesitamos tener una representación discretizada del centerline y de la pared de la aorta, para trabajar con la pared aórtica decidimos utilizar el mapa de distancias de la pared al centerline y es lo que discretizaremos.

(a) Centerline de una aorta



(b) Mapa de distancias de la pared al centerline.

Figura 3.1: Centerline de una arteria aorta y su mapa de distancias.

En este capítulo haremos una introducción a la aproximación de curvas en $\mathbb{R}^2$ con curvas B-spline y llegaremos a aproximar superficies en $\mathbb{R}^3$ con B-splines. Permitiéndonos estas aproximaciones obtener discretizaciones del centerline y del mapa de distancias.

3.1. Parametrización de curvas

Antes de comenzar a aproximar la curva del centerline y el mapa de distancias repasaremos algunos conceptos básicos. Dados dos puntos $c_0 = (x_0, y_0)$ y $c_1 = (x_1, y_1)$, podemos unirlos con una linea recta con la siguiente parametrización.

$$q(t|c_0, c_1; t_0, t_1) = \frac{t_1 - t}{t_1 - t_0} c_0 + \frac{t - t_0}{t_1 - t_0} c_1 \quad t \in [t_0, t_1] \tag{3.1}$$

Siendo esto una combinación convexa, pudiendo tener una representación tal que para cada número real le podemos asociar un punto en $\mathbb{R}$.

$$y = f(x) = \frac{x_1 - x}{x_1 - x_0} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 \tag{3.2}$$

Este representación presenta algunos problemas, si $x_0 = x_1$, es decir, si la linea es vertical.

3.1.1. Interpolación de curvas polinomiales

Lo mas natural cuando tratamos de construir una curva a lo largo de unos puntos es que queremos que la curva pase por los puntos.

Interpolación cuadrática de tres puntos

Tratemos de construir una curva que interpole tres puntos. Para ello necesitaremos tres puntos c_0, c_1, c_2 y tres parámetros t_0, t_1, t_2.

Comenzamos construyendo dos rectas que unan los puntos c_0 con c_1 y c_1 con c_2, $q_{0,1}(t|c_0, c_1; t_0, t_1)$ y $q_{1,2}(t|c_1, c_2; t_1, t_2)$. Con estas dos podemos construir una curva que pase por los tres puntos.

$$q_{0,2}(t|c_0, c_1, c_2; t_0, t_1, t_2) = \frac{t_2 - t}{t_2 - t_0} q_{0,1}(t) + \frac{t - t_0}{t_2 - t_0} q_{1,2}(t) \tag{3.3}$$

Siendo esto una curva cuadrática en t, comprobemos que pasa por los tres puntos:

$$q_{0,2}(t_0) = c_0$$

$$q_{0,2}(t_1) = \frac{t_2 - t_1}{t_2 - t_0} q_{0,1}(t_1) + \frac{t_1 - t_0}{t_2 - t_0} q_{1,2}(t_1) = \frac{t_2 - t_1}{t_2 - t_0} c_1 + \frac{t_1 - t_0}{t_2 - t_0} c_1 = c_1$$

$$q_{0,2}(t_2) = q_{1,2}(t_2) = c_2$$

Como podemos ver en la Figura 3.2, dados tres puntos $(c_i)_{i=0}^2$ y tres parámetros $(t_i)_{i=0}^2$, podemos interpolar los puntos. En las Figuras a) y b), tenemos los mismos tres puntos, pero los parámetros $(t_i)_{i=0}^2$ son diferentes, es por ello que obtenemos curvas diferentes. Lo mismo pasa con las Figuras c) y d).

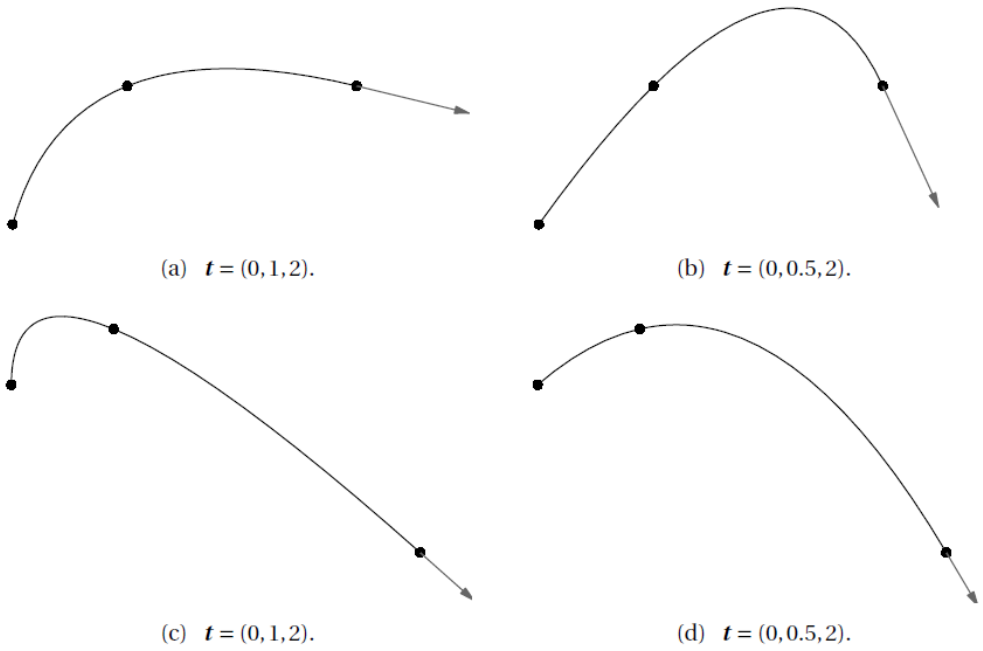


Figura 3.2: Ejemplos de interpolaciones cuadráticas de tres puntos [9].

3.2. Curvas de Bezier

Las curvas de Bezier son una alternativa a la interpolación polinomial, este método también son curvas polinomiales pero evitan el problema de abultamientos ya que se tratan de combinaciones convexas. Estas curvas nos permitirán aproximar curvas con una gran cantidad de puntos ya que se pueden unir suavemente unas con otras, evitando así tener que usar polinomios de un alto grado.

3.2.1. Curvas cuadráticas de Bezier

Supongamos que tenemos tres puntos $(c_i)_{i=0}^2$ contenidos en un plano y queremos construir una curva suave formada por combinaciones convexas.

Comencemos definiendo los segmentos que unen c_0 con c_1 y c_1 con c_2 en el mismo intervalo $[0, 1]$.

$$p_{0,1}(t) = p(t|c_0, c_1) = (1-t)c_0 + tc_1 \tag{3.4}$$

$$p_{1,2}(t) = p(t|c_1, c_2) = (1-t)c_1 + tc_2 \tag{3.5}$$

De esta forma podemos construir una combinación convexa tal que así:

$$p_{0,2}(t) = p(t|c_0, c_1, c_2) = (1-t)p_{0,1}(t) + tp_{0,2}(t) \tag{3.6}$$

Desarrollando lo tenemos que:

$$p_{0,2}(t) = (1-t)^2 c_0 + 2t(1-t)c_1 + t^2 c_2 \tag{3.7}$$

En la Figura 3.3 hemos hecho la construcción basada en 15 puntos equidistantes de los valores de t. La Figura 3.4 son dos curvas cuadráticas de Bezier de tres puntos.

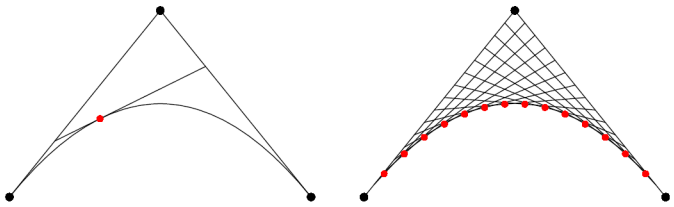


Figura 3.3: Curva de Bezier basada en tres puntos [9]

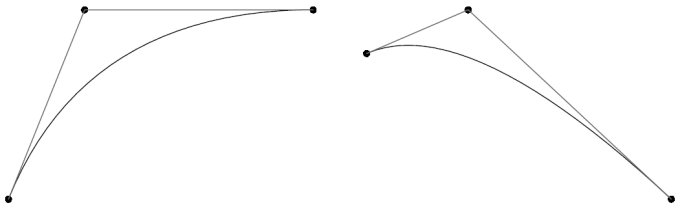


Figura 3.4: Dos curvas cuadráticas de Bezier [9]

3.3. Construcción geométrica de curvas spline

Las curvas de Bezier presentan la desventaja que solo pueden ser controladas por los puntos de control elegidos apropiadamente.

En las curvas cuadráticas de Bezier, los puntos $(c_i)_{i=0}^2$ se les llaman puntos de control de la curva y la curva linea que conecta los puntos de control se llama polígono de control de la curva. Si ajustamos la construcción de las curvas de Bezier, podemos generar curvas polinomiales que se ajustan automáticamente suavemente. A estas curvas polinomiales las llamaremos curvas spline.

3.3.1. Curvas spline lineales

Para generar estas curvas necesitaremos una generalización de las curvas de Bezier. Anteriormente, hemos introducido una representación de una linea que conecta dos puntos.

$$p(t|c_1, c_2; t_2, t_3) = \frac{t_3 - t}{t_3 - t_2} c_1 + \frac{t - t_2}{t_3 - t_2} c_2 \quad t \in [t_2, t_3] \tag{3.8}$$

siendo $t_2 < t_3$, tenemos en cuenta que si $t_2 = 0$ y $t_3 = 1$ volvemos a tener una curva de Bezier lineal.

Basándonos en esa idea, es intuitivo como podemos construir una curva que pase por todos los puntos $(c_i)_{i=1}^n$, simplemente los vamos conectando con rectas los puntos que son vecinos. Esta curva se puede parametrizar de esta forma:

$$f(t) = \begin{cases} p(t|c_1, c_2; t_2, t_3), & t \in [t_2, t_3) \\ p(t|c_2, c_3; t_3, t_4), & t \in [t_3, t_4) \\ \dots & \dots \\ p(t|c_{n-1}, c_n; t_n, t_{n+1}), & t \in [t_n, t_{n+1}) \end{cases} \tag{3.9}$$

siendo $(c_i)_{i=1}^n$ los puntos y $(t_i)_{i=2}^{n+1}$ números tales que $t_i < t_{i+1} \quad \forall i \in [2, n]$.

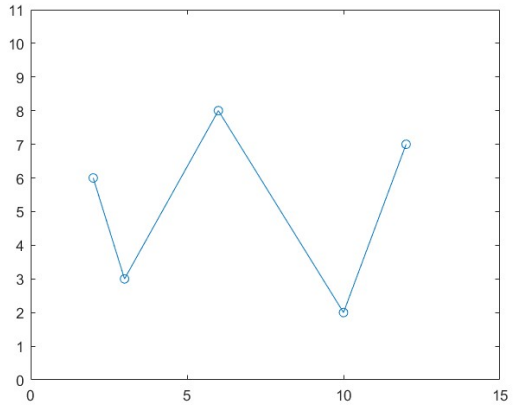


Figura 3.5: Curva spline lineal de los puntos $(c_i)_{i=1}^5 = [6, 3, 8, 2, 7]$ y $(t_i)_{i=1}^5 = [2, 3, 6, 10, 12]$

A los puntos $(c_i)_{i=1}^n$ los llamamos puntos de control de la curva, y a los parámetros $(t_i)_{i=2}^{n+1}$ los llamamos *knots*, o tambien *knot vector* de la curva.

Introduciendo la función $B_{i,0}(t)$

$$B_{i,0}(t) = \begin{cases} 1 & t \in [t_i, t_{i+1}) \\ 0 & t \notin [t_i, t_{i+1}) \end{cases} \tag{3.10}$$

podemos escribir la función $f(t)$ de forma mas compacta siendo $p_{i,1} = p(t|c_{i-1}, c_i; t_i, t_{i+1})$ de esta forma:

$$f(t) = \sum_{i=2}^{n} p_{i,1}(t) B_{i,0}(t) \tag{3.11}$$

3.3.2. Curvas spline cuadráticas

Construiremos una curva spline basada en tres puntos de control c_1, c_2, c_3 con cuatro *knots* $(t_i)_{i=2}^5$ teniendo en cuenta que $t_2 \leq t_3 < t_4 \leq t_5$. Representamos la linea que conecta dos puntos c_i, c_{i+1} por $p(t|c_i, c_{i+1}; t_{i+1}, t_{i+3})$ para $t \in [t_i, t_{i+2}]$. Esto nos permite combinar dos lineas en una combinación convexa para formar una curva cuadrática.

$$p(t|c_1, c_2, c_3; t_2, t_3, t_4, t_5) = \frac{t_4 - t}{t_4 - t_3} p(t|c_1, c_2; t_2, t_4) + \frac{t - t_4}{t_4 - t_3} p(t|c_3, c_4; t_3, t_5) \quad t \in [t_3, t_4] \tag{3.12}$$

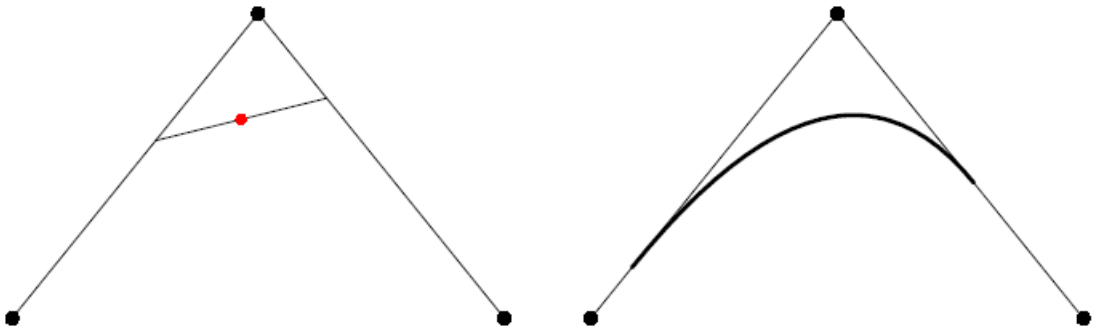


Figura 3.6: Construcción de un segmento de una curva spline cuadrática [9].

En la Figura 3.6 podemos ver la construcción de una curva spline cuadrática de tres puntos de control. Hay que tener en cuenta que si $t_1 = t_2 = 0$ y si $t_3 = t_4 = 1$ volvemos a tener una curva de Bezier. Al igual que con las curvas de Bezier, a los puntos dados los llamamos puntos de control y la curva lineal que une esos puntos lo llamamos polígono de control.

Por lo tanto podemos generalizarlo, supongamos que tenemos n puntos de control $(c_i)_{i=1}^{n}$ y el *knots vector* $(t_i)_{i=2}^{n+2}$, podemos definir la curva spline cuadrática $f(t)$ como

$$f(t) = \begin{cases} p(t|c_1, c_2, c_3; t_2, t_3, t_4, t_5), & t \in [t_3, t_4) \\ p(t|c_2, c_3, c_4; t_3, t_4, t_5, t_6), & t \in [t_4, t_5) \\ \dots & \dots \\ p(t|c_{n-2}, c_{n-1}, c_n; t_{n-1}, t_n, t_{n+1}, t_{n+2}), & t \in [t_n, t_{n+1}) \end{cases} \tag{3.13}$$

Y de la misma forma a como hemos hecho con la ecuación 3.11, podemos definirnos la función $B_{i,0}(t)$

$$B_{i,0}(t) = \begin{cases} 1 & t \in [t_i, t_{i+1}) \\ 0 & t \notin [t_i, t_{i+1}) \end{cases} \tag{3.14}$$

y la abreviación $p_{i,2}(t) = p(t|c_{i-2}, c_{i-1}, c_i; t_{i-1}, t_i, t_{i+1}, t_{i+2})$ para escribir de forma compacta $f(t)$.

$$f(t) = \sum_{i=3}^{n} p_{i,2}(t) B_{i,0}(t) \tag{3.15}$$

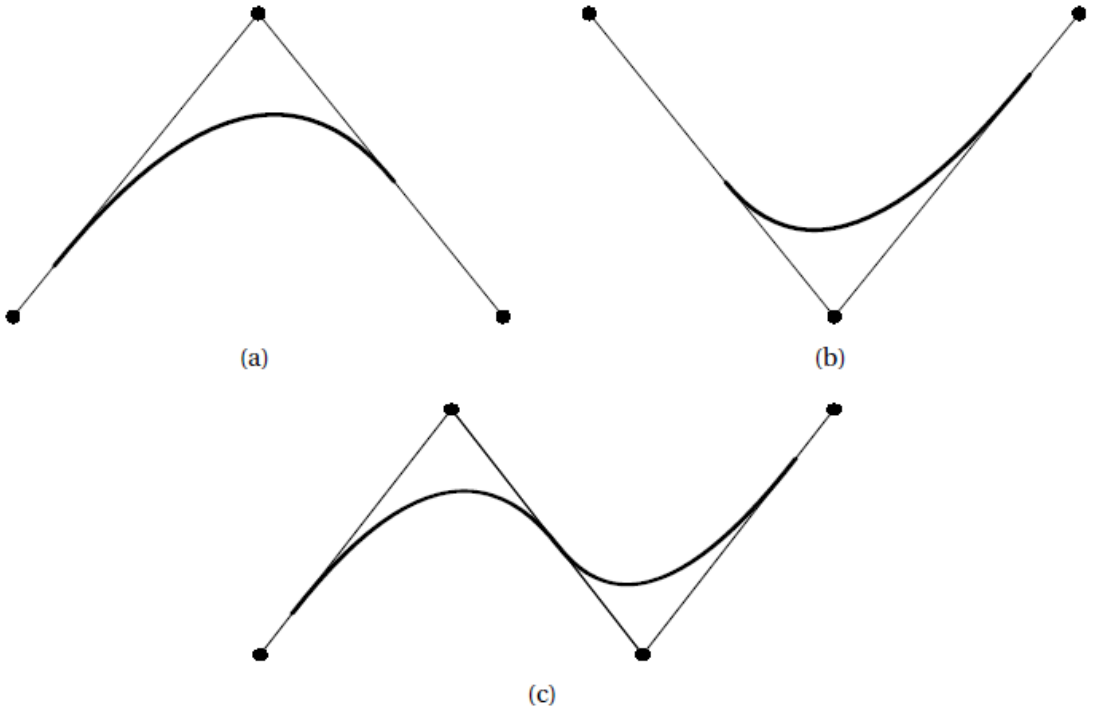


Figura 3.7: a). Curva cuadrática definida en $[t_2, t_3]$; b).Curva cuadrática definida en $[t_3, t_4]$; c). Curva spline cuadrática resultado de la unión de las curvas a) y b).

Permitiendo así la unión suave de diversas curvas como se muestra en la Figura 3.7. Por lo tanto, supongamos que tenemos n puntos de control $(c_i)_{i=1}^{n}$ y n *knots* $(t_i)_{i=2}^{n+2}$. Hasta ahora hemos visto que podemos expresar un spline linear como

$$f(t) = \sum_{i=3}^{n} p_{i,2}(t) B_{i,0}(t) \quad t \in [t_2, t_{n+2}] \tag{3.16}$$

siendo $\{B_{i,0}\}_{i=3}^{n}$ como hemos definido en 3.14. Teniendo en cuenta el desarrollo en 3.9 tenemos que:

$$f(t) = \sum_{i=2}^{n}(\frac{t-t_i}{t_{i+1}-t_i}p_iB_{i,0}(t) + \frac{t_{i+1}-t}{t_{i+1}-t_i}p_{i-1}B_{i,0}(t)) =$$
$$\sum_{i=2}^{n}(\frac{t-t_i}{t_{i+1}-t_i}B_{i,0}(t) + \frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}B_{i+1,0}(t))p_i + \tag{3.17}$$
$$\frac{t_3-t}{t_3-t_2}B_{2,0}(t)p_1 + \frac{t-t_n}{t_{n+1}-t_n}B_{n,0}(t)p_n$$

Por lo que si definimos las funciones

$$B_{i,1}(t) = \frac{t-t_i}{t_{i+1}-t_i}B_{i,0}(t) + \frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}B_{i+1,0}(t) \quad i \in [2,n] \tag{3.18}$$

podemos reescribir la función $f(t)$ como

$$f(t) = \sum_{i=1}^{n} p_iB_{i,1}(t) \tag{3.19}$$

Estas funciones se llaman *basic splines*, y abreviadamente, B-splines. Y podemos repetir el proceso en el caso de cuadráticas obteniendo B-splines cuadráticos con las funciones

$$B_{i,2}(t) = \frac{t-t_i}{t_{i+2}-t_i}B_{i,1}(t) + \frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}B_{i+1,1}(t) \tag{3.20}$$

definidas por recurrencia empezando por $B_{i,0}$ definidas en 3.14.

3.4. B-Splines

En la sección 3.3.2 hemos visto que las curvas cuadráticas spline pueden ser representadas en base de la funciones B-spline. Estas funciones pueden definirse para cualquier grado d.

Definición 15. Sea d un entero no negativo y $t = (t_j)_{j=1}^{n+d+1}$ el *knot vector*. Si $1 \leq j \leq n$, el j-ésimo B-spline de grado d con *knot vector* t esta definido por

$$B_{j,d,t}(x) = \frac{x-t_j}{t_{j+d}-t_j}B_{j,d-1,t}(x) + \frac{t_{j+1+d}-x}{t_{j+1+d}-t_{j+1}} \tag{3.21}$$

siendo x un numero real y $B_{j,0,t}$

$$B_{j,0,t} = \begin{cases} 1 & x \in [t_j, t_{j+1}) \\ 0 & x \notin [t_j, t_{j+1}) \end{cases} \tag{3.22}$$

Proposición 1. *Sea $t = (t_j)_{j=1}^{n+d+1}$ el knot vector de un total de n $B-splines$. El espacio lineal de todas las combinaciones de estos B-splines es el espacio spline $\mathbf{S}_{d,t}$ definido por*

$$\mathbf{S}_{d,t} = span\{B_{1,d}, ..., B_{n,d}\} = \{\sum_{j=1}^{n} c_j B_{j,d} | c_j \in \mathbb{R}, j \in [1, n]\}$$

Un elemento $f = \sum_{j=1}^{n} c_j B_{j,d}$ de $\mathbf{S}_{d,t}$ se le llama función spline de grado d , knots t y los $(c_j)_{j=1}^{n}$ se llaman los coeficientes de f.

Definición 16. Polígono de control de la función spline: Sea $f = \sum_{j=1}^{n} c_j B_{j,d}$ una función spline de $\mathbf{S}_{d,t}$. Los puntos de control de f son los puntos con coordenadas (t_j^*, c_j) $j = 1, ..., n$, siendo

$$t_j^* = \frac{t_{j+1} + \cdots + t_{j+d}}{d} \tag{3.23}$$

El polígono de control es la función lineal que une los puntos de control vecinos con lineas rectas.

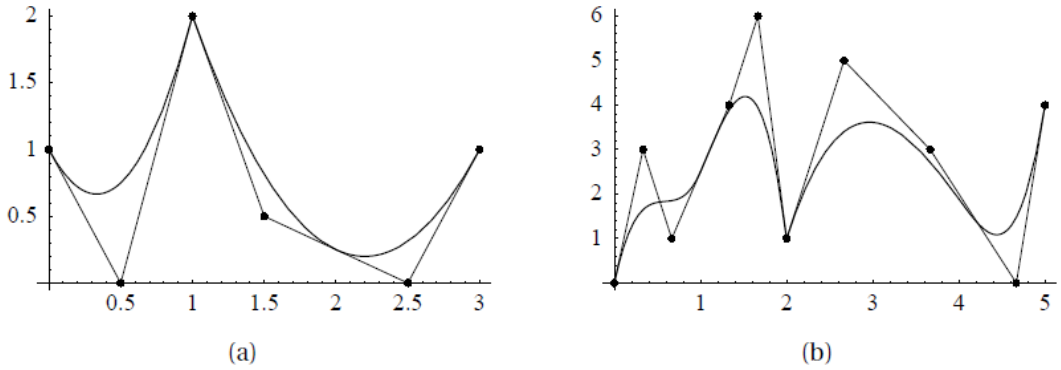


Figura 3.8: Splines con su correspondientes polígonos de control. El spline a) es cuadrático con *knots* $t = (0, 0, 0, 1, 1, 2, 3, 3, 3)$ y coeficientes del B-spline $c = (1, 0, 2, 1/2, 0, 1)$. El spline b) es cúbico y con *knots* $t = (0, 0, 0, 0, 1, 1, 2, 2, 2, 4, 5, 5, 5, 5)$ y los coeficientes del B-spline son $c = (0, 3, 1, 4, 6, 1, 5, 3, 0, 4)$. [9]

Por lo tanto si tenemos una curva como puede ser en nuestro problema el centerline, podemos obtener una aproximación de este cogiendo una cierta cantidad de puntos de la curva, $(c_i)_{i=1}^{n}$,

que serán los puntos de control y un *knot vector* t. De forma que obtengamos una función spline, de la cual nosotros aproximaremos cada componente (x, y, z) del centerline por una función spline y nos quedaremos con sus coeficientes.

Figura 3.9: Coeficientes de la función spline que aproxima cada componente (x, y, z) del centerline. En total tenemos 3×20 coeficientes.

Ya tenemos una discretización del centerline, ahora nos interesa extenderlo al mapa de distancias.

3.4.1. B-splines para superficies

Ahora que ya sabemos como aproximar curvas en $\mathbb{R}^2$, veamos como extenderlo para que aproximar superficies en $\mathbb{R}^3$.

Recapitulando, recordamos que si tenemos dos puntos b_0 y b_1, podemos unirlos con una recta.

$$b_0(1-x) + b_1 x \quad x \in [0,1] \tag{3.24}$$

.

Ahora supongamos que b_0 y b_1 dependen de una variable y, es decir, que tenemos $b_0(y)$ y $b_1(y)$ con $y \in [c, d]$. Por lo tanto, para cada $y \in [c, d]$, tenemos la recta $b_0(y)(1-x) + b_1(y)x$ $x \in [0, 1]$ que une los puntos b_0 con b_1. Por lo tanto, esto parametriza una superficie.

$$z = b_0(y)(1-x) + b_1(y)x \quad x \in [0,1] \quad y \in [c,d] \tag{3.25}$$

Definamos una familia de funciones splines para construir superficies spline. Supongamos que tenemos un entero d y un *knot vector* σ, definimos el espacio spline como:

$$\mathbf{S}_1 = \mathbf{S}_{d,\sigma} = span\{\varphi_1, \cdots, \varphi_{n_1}\} \tag{3.26}$$

siendo $\{\varphi\}_{i=1}^{n_1}$ la familia de B-splines. Consideramos la función spline f en $\mathbf{S}_1$

$$f(x,y) = \sum_{i=1}^{n_1} c_i(y)\varphi_i(x) \tag{3.27}$$

cuyos coeficientes $(c_i)_{i=1}^{n_1}$ dependen de y. Por lo tanto, para cada valor de y tenemos un spline en $\mathbf{S}_1$. Cualquier función para los coeficientes $(c_i)_{i=1}^{n_1}$ hace que f sea una superficie, pero es de gran utilidad que $(c_i)_{i=1}^{n_1}$ sean splines.

Por lo tanto, supongamos que tenemos otro espacio spline de grado l y con *knots* τ.

$$\mathbf{S}_2 = \mathbf{S}_{l,\tau} = span\{\psi_1, \cdots, \psi_{n_2}\} \tag{3.28}$$

siendo $\{\psi_j\}_{j=1}^{n_2}$ B-splines. Entonces, para función de coeficientes $c_i(y)$ podemos aproximarlo con un spline en $\mathbf{S}_2$.

$$c_i(y) = \sum_{j=1}^{n_2} c_{i,j}\psi_j(y) \tag{3.29}$$

siendo $(c_{i,j})_{j=1}^{n_2}$ los coeficientes de $c_i(y)$. Entonces si combinamos 3.27 y 3.29 obtenemos:

$$f(x,y) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} c_{i,j}\phi_i(x)\varphi_j(y) \tag{3.30}$$

Definición 17. El producto tensorial de dos espacio spline $\mathbf{S}_1$ y $\mathbf{S}_2$ se define como la familia de funciones

$$f(x,y) = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} c_{i,j}\phi_i(x)\varphi_j(y) \tag{3.31}$$

siendo $(c_{i,j})_{i,j=1}^{n_1,n_2}$ coeficientes que pueden ser cualquier numero real.

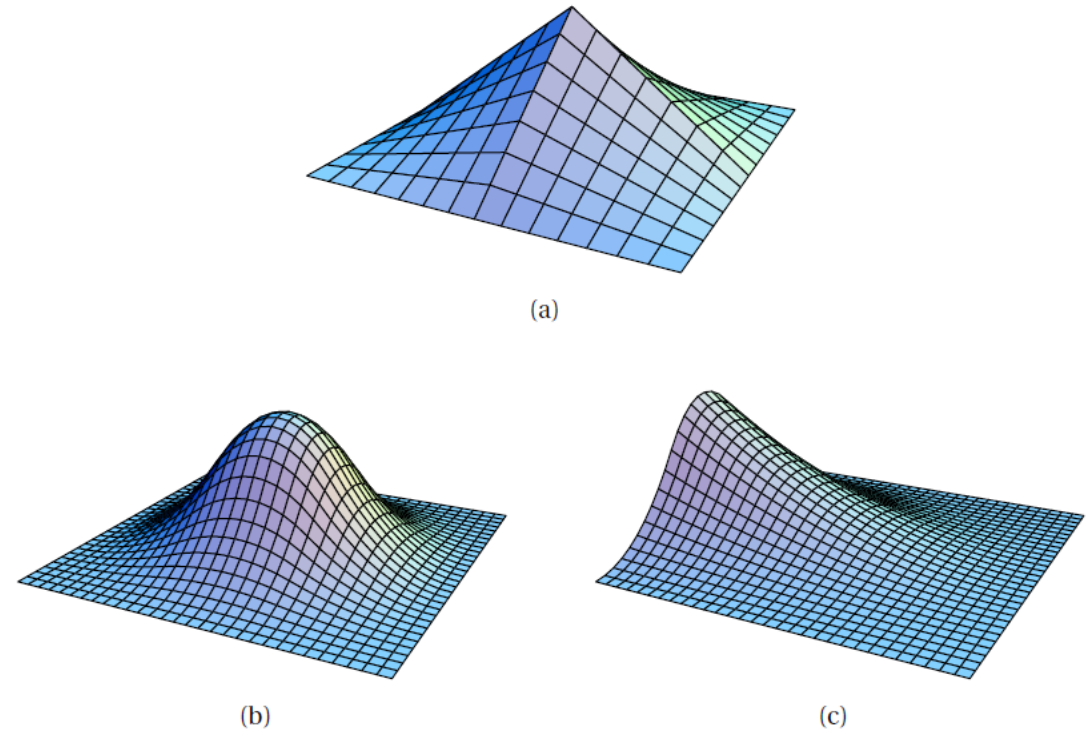


Figura 3.10: a) B-spline bilinear; b) B-spline bicuadrático; c) B-spline bicuadrático con un *knot* triple en un dirección, es decir, que un *knot* se repite tres veces en la secuencia.

Entonces, para nuestro problema de aproximación y discretización de la malla de una arteria aorta, por un lado tenemos el centerline y por otro el mapa de distacias de la pared aórtica al centerline. Para poder trabajar con estos datos, necesitamos tener una representación discretizada del centerline y del mapa de distancias. Para ello aproximaremos el centerline por una curva B-spline de tercer orden y el mapa de distancias por un *B-Spline* de superficies de tercer orden.

Dado un numero fijo de puntos de control del B-Spline (n_c), obtendremos coeficientes n_c para cada componente (x, y, z) del centerline, por lo tanto un total de $3 \times n_c$ coeficientes para el centerline. Y para la pared tenemos que fijar un numero de puntos de control para la variable s y θ a los que denotaremos por n_{x^s} y n_{x^θ}, obteniendo una matriz de $n_{x^s} \times n_{x^\theta}$ valores.

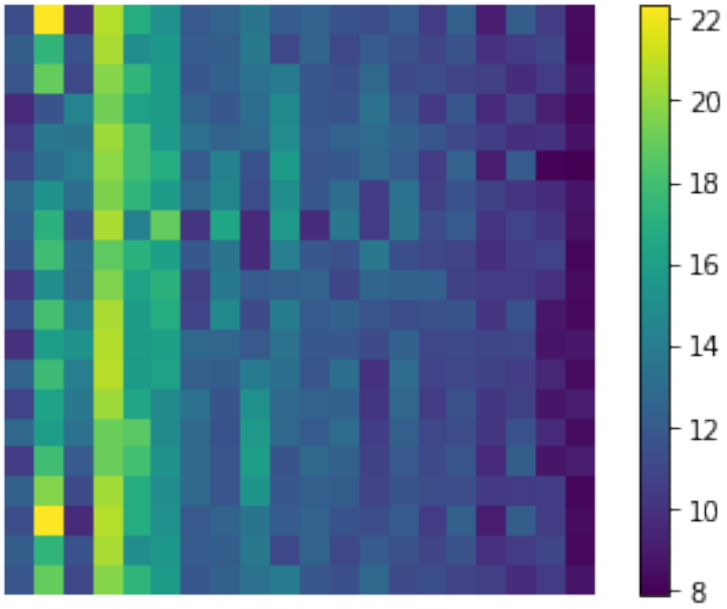


Figura 3.11: Coeficientes del mapa de distancias de la pared aórtica al mapa de distancias.

Figura 3.12: Diagrama de flujo de la obtención de los coeficientes de los B-Splines. Partiendo de la imagen médica obtenida a través de un TAC, siguiendo con la extracción del centerline y el mapa de distancias de la pared al centerline, terminando con los coeficientes de los B-Splines del centerline y del mapa de distancias.

Lo cual ya es una representación discretizada y de dimensiones constantes de la malla de una arteria aorta con la que vamos a poder trabajar y entrenar nuestros modelos.

Capítulo 4

Material y métodos

En este capítulo hablaremos sobre como hemos modificado nuestra base de datos para poder trabajar con ella. Además comentaremos que medidas utilizaremos para validar las aortas que nuestro modelo genere. Cerraremos el capítulo describiendo a la metodología con la que vamos a trabajar.

4.1. Base de datos

La base de datos que tenemos es de 34 casos clínicos reales con aneurisma, que nos ha facilitado el equipo de CoMMLab. Estas aortas fueron obtenidas a través del método de Tomografía Axial Computerizada (TAC), en un estudio retrospectivo previo en colaboración con el Hospital Universitario La Fe de Valencia.

Figura 4.1: Arterias aortas con aneurisma.

4.1.1. Transformación de los datos

Las aortas que tenemos son una triangulación de los vértices obtenidos a través del TAC. Para trabajar con ellas utilizaremos las herramientas mencionadas en los capítulos 2.1 y 3. Para cada

una de las aortas extraeremos su centerline, la curva que va por el centro de la aorta. Además podemos tomar el mapa de distancias que hay de la pared al centerline como los valores de la función $\rho(s, \theta)$ en 2.15.

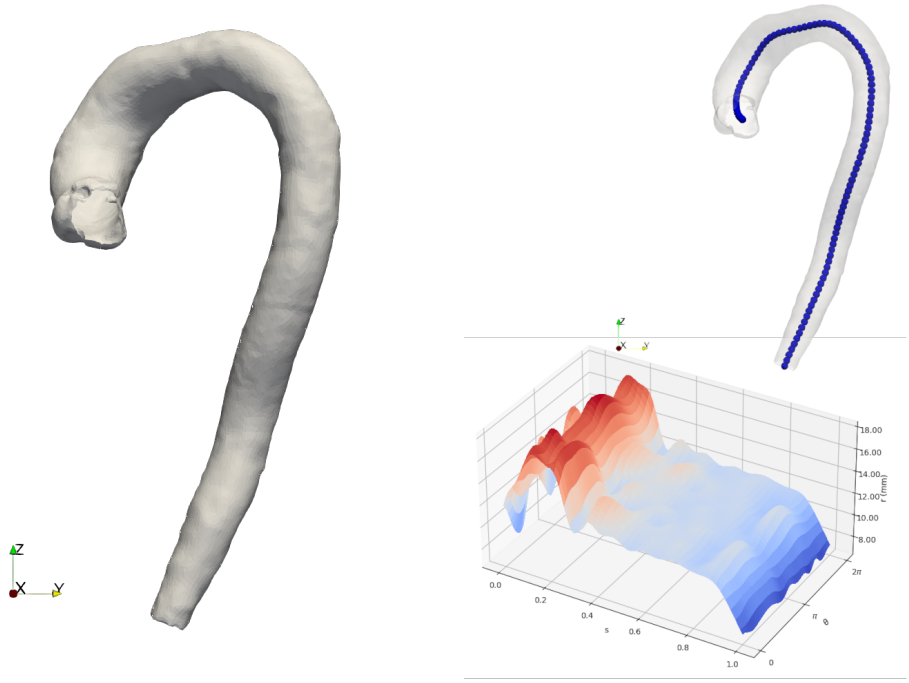
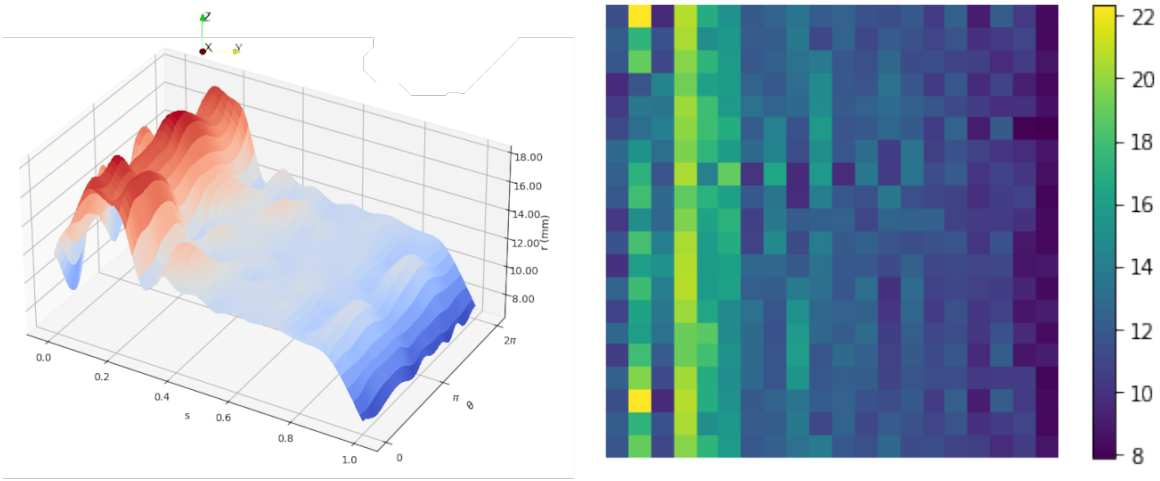


Figura 4.2: Extracción del centerline y mapa de distancia de la pared aórtica al centerline. [8]

Una vez tenemos el centerline y el mapa de distancia de la pared al centerline, para cada una de las componentes (x, y, z) del centerline podemos aproximar los con una curva B-spline como se explica en el apartado 3.3.2. Una vez tenemos las componentes aproximadas nos quedamos con los coeficientes del B-spline, con los cuales obtenemos imágenes como 4.3.

Figura 4.3: Representación de los coeficientes del B-Spline que aproxima al centerline, cada columna indica la posición (x, y, z) respectivamente.

En el caso del mapa de distancias de la pared al centerline estamos ante una superficie en $\mathbb{R}^3$ por lo que la aproximaremos como se indica en el apartado 17. Al igual que hemos hecho con el centerline, nos quedamos con los coeficientes de la función $f(t)$ de 17, $(c_{i,j})_{i,j}^{n_1,n_2}$. Por lo que para cada mapa de distancias obtenemos imágenes como la Figura 4.4b.



(a) Mapa de distancias de la pared al centerline.

(b) Coeficientes del B-spline que aproxima al mapa de distancias.

Figura 4.4: Representación de los coeficientes del B-Spline que aproxima al mapa de distancias de la pared al centerline

Nosotros queremos que nuestro modelo sea capaz de generar centerlines y paredes que estén relacionadas, es por ello que las dos imágenes anteriores las concatenaremos, siendo las tres primeras filas el centerline y el resto el mapa de distancias.

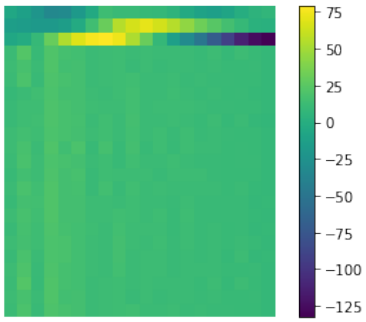


Figura 4.5: Representación de una arteria aorta utilizando los coeficientes del centerline en las tres primera filas y el resto de filas son los coeficientes del mapa de distancias. Concatenación de 4.3 y 4.4b.

Como se puede observar, hay una gran diferencia en la escala del centerline y de la pared. Por motivos que luego mencionaremos vamos a normalizar la imagen. Una vez ya tenemos todas representaciones concatenadas, para normalizarlas, tras diversas pruebas, los mejores resultados los hemos obtenido normalizando de la siguiente forma:

- El centerline lo hemos normalizado por filas, es decir, hemos cogido el máximo y mínimo de todas las primeras filas y las hemos normalizado, luego cogemos el máximo y mínimo de todas las segundas filas y las normalizamos y lo mismo para la tercera.
- El mapa de distancias lo hemos normalizado por columnas, es decir, hemos cogido el máximo y mínimo de todas las primeras columnas correspondientes a la pared y las hemos normalizado, luego cogemos el máximo y mínimo de todas las segundas columnas y las normalizamos y lo mismo para el resto de columnas.

Obteniendo imágenes como:

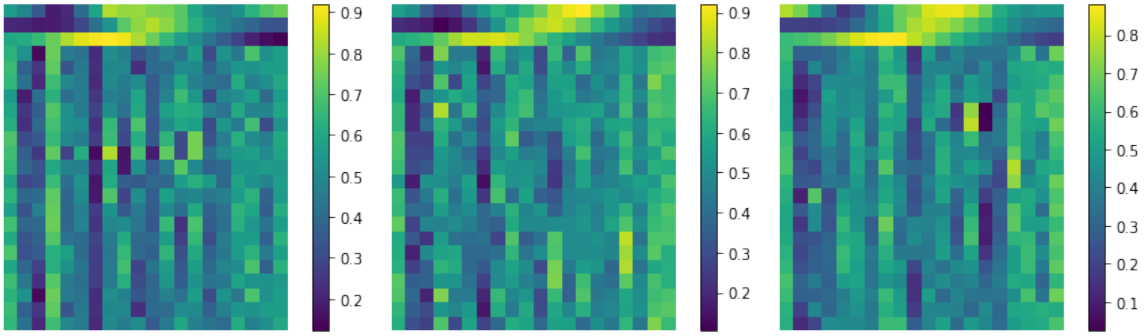


Figura 4.6: Centerlines y paredes de tres aortas diferentes normalizadas

Una vez tenemos imágenes como las de la Figura 4.6 podemos retroceder en el proceso y

reconstruir una aproximación de la aorta debido a las propiedades de los B-Splines.

Figura 4.7: Aorta reconstruida

Por lo tanto, somos capaces de representar aortas en imágenes como las de la Figura 4.6, reduciendo así la dimensionalidad del espacio ya que una imagen médica de una arteria aorta contiene mas de 350.000 vértices y nosotros lo hemos pasado a imágenes de 23×20. Además, aparte de haber reducido dimensionalidad, esta es constante, lo que nos permite usar modelos de *Machine Learning*. Usaremos la representación de la Figura 4.6 para entrenar nuestro modelo, el cual querremos que genere representaciones similares.

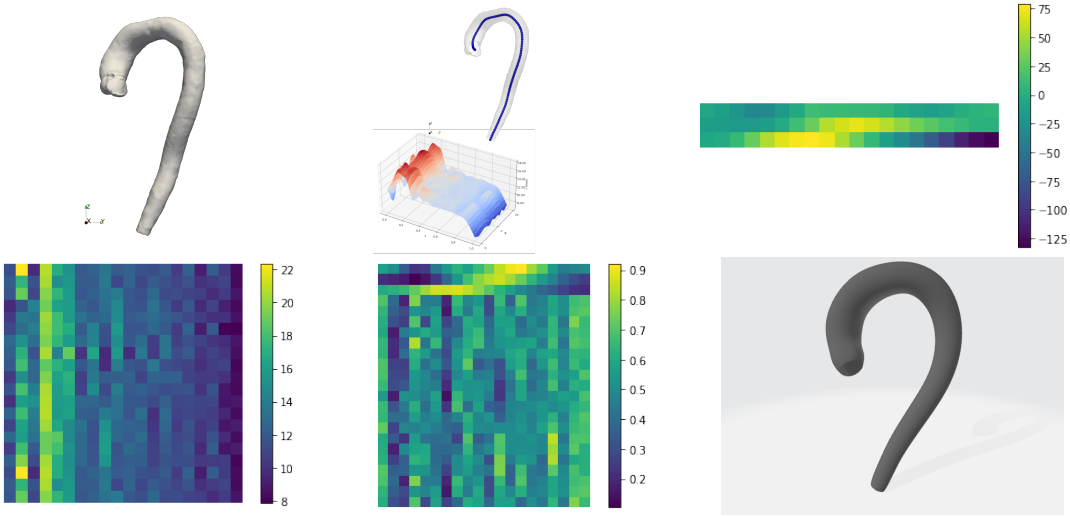


Figura 4.8: Diagrama de flujo de la obtención de los coeficientes de los B-Splines y reconstrucción de la aorta. Partiendo de la imagen médica obtenida a través de un TAC, siguiendo con la extracción del centerline y el mapa de distancias de la pared al centerline y sus coeficientes de los B-splines, los concatenamos y los normalizamos, terminando con una reconstrucción.

4.1.2. Biomarcadores

Un biomarcador es una característica objetiva que nos permiten saber el estado biológico de un órgano u otro aspecto relacionado con la salud. Su principal uso es la detección de enfermedades.

Para saber la eficiencia de nuestro modelo realizaremos mediciones a lo largo de las 1000 aortas sintéticas que generemos. Las mediciones se hacen en ciertas características de la aorta que nos permiten saber como de plausibles son las aortas generadas, estos son biomarcadores físicos.

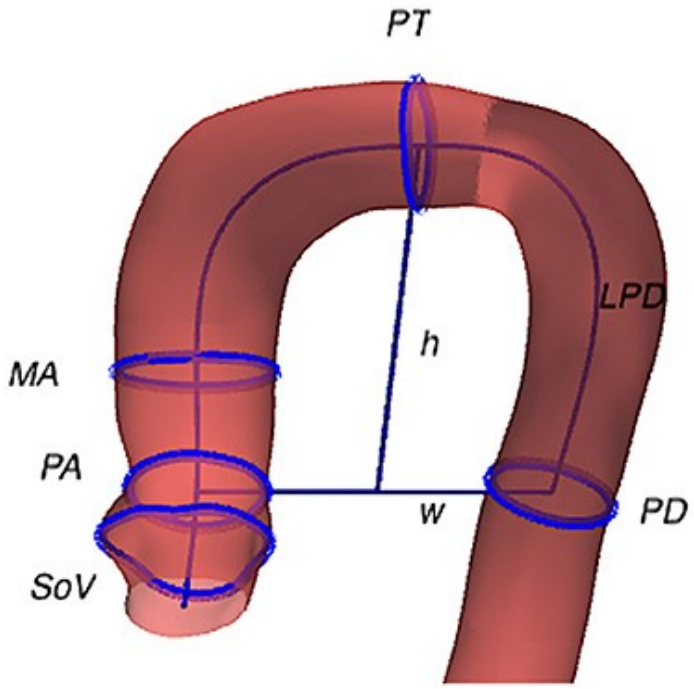


Figura 4.9: Biomarcadores de la arteria aorta

Biomarcador	Descripción
SoV	Radio de la aorta en el medio de los senos de Valsalva (mm).
PA	Radio de la aorta ascendente, cerca de la unión seno tubular (mm).
MA	Radio en la mitad de la aorta ascendente (mm).
PT	Radio en la parte superior del arco aortico (mm).
PD	Radio en la aorta descendete, en el lado contrario a PA (mm).
LPD	Longitud del centerline desde la valvula al PD (mm).
k	Media de la curvatura del centerline desde PA a PD (1/mm).
h	Altura desde PT a PA/PD (mm).
w	Anchura del arco, distancia desde PD hasta PA (mm).
h/w	Ratio altura-anchura.
tor	Tortuosidad, definido como $1 - \frac{w}{LPD}$.

Cuadro 4.1: Tabla explicativa de cada biomarcador.

Para cada uno de los biomarcadores tomaremos mediciones de las arterias generadas y de las de la base de datos. Generaremos las distribuciones de probabilidad y eso nos permitirá saber como de plausibles son las arterias aorta que nuestro modelo generador esta generando. Y así sera como sabremos la eficiencia de nuestro modelo.

4.2. Metodología

La base de datos original consta de 34 arterias aortas. Como ya hemos mencionado nuestro objetivo principal es ser capaces de aumentar esta base de datos y a su vez, ver la influencia de las capas convolucionales en este proceso.

Para ello, utilizaremos la base de datos de 1.000 aortas generadas en el paper *Clinically-Driven Virtual Patient Cohorts Generation: An Application to Aorta)* [15] con una GAN no convolucional que había sido entrenada con una base de datos en la que las aortas eran vectores de características a las que se les había aplicado un PCA y después con técnicas de boostraping se amplió hasta 3.000 aortas (bloque izquierdo en la Figura 4.10). Entrenaremos una GAN convolucional con la base de datos original y luego otra GAN convolucional con la base de datos de 3.000 aortas mencionada anteriormente, y generaremos 1.000 aortas en ambos casos.

En la Figura 4.10 podemos ver un esquema de la metodología. Todo viene de la necesidad de ampliar la base de datos original de 34 aortas. En trabajo previo tenemos el recuadro azul publicado en el paper *Clinically-Driven Virtual Patient Cohorts Generation: An Application to Aorta* [15]. En este trabajo final de grado nos centraremos en el recuadro de color rojo (bloque central), en el que trabajamos con GANs convolucionales y compraremos los resultados con el trabajo previo. Para trabajo futuro queda la generación de aortas sintéticas con GANs no convolucionales con la base de datos original (bloque derecho en la Figura 4.10).

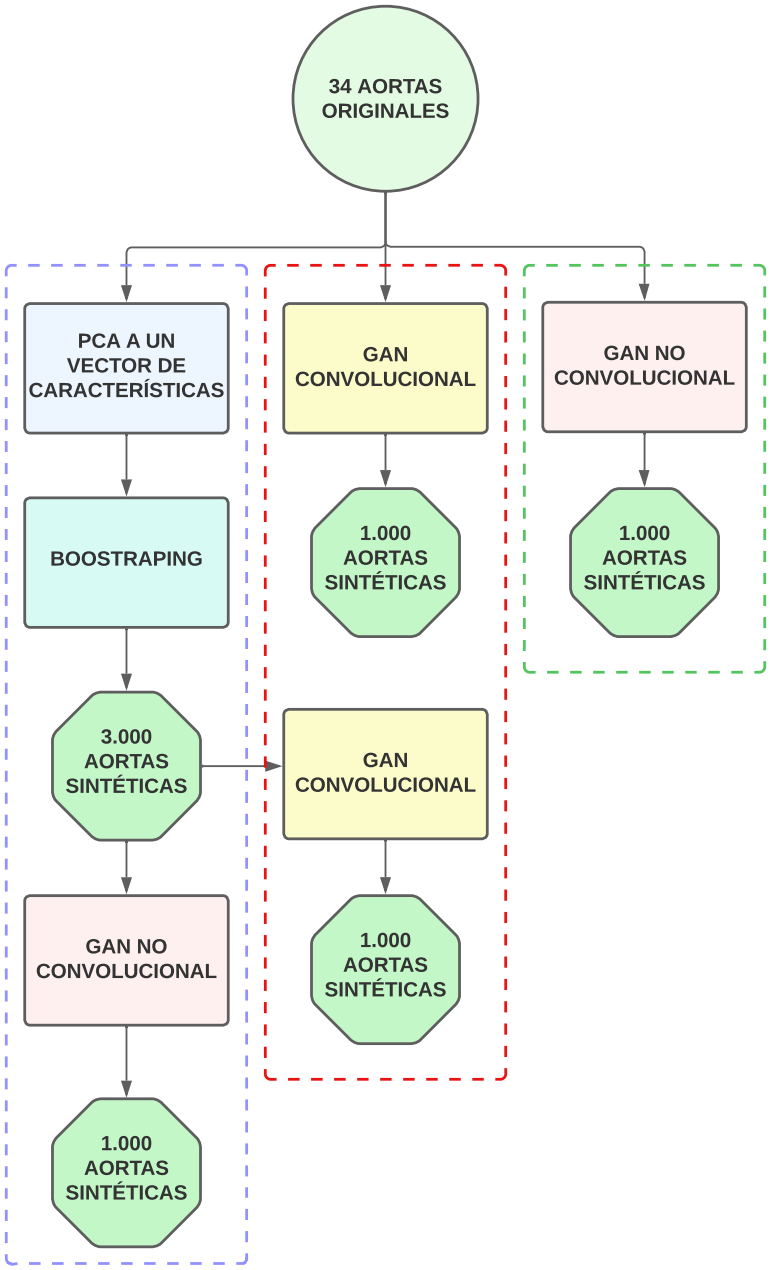


Figura 4.10: Diagrama que resume la metodología que vamos a llevar a lo largo del trabajo. Este TFG es parte de un proyecto que se compone de tres bloques principales, el bloque izquierdo corresponde a trabajo previo publicado cuyos resultados se publicaron en [15], el bloque central corresponde a este TFG, parte de estos resultados se han publicado en [8]. Como trabajo futuro tenemos el bloque izquierdo.

Capítulo 5

Generación de arterias aorta con redes neuronales

Este trabajo entraremos en detalle explicando que es una generative adversarial networks, también conocidas como GANs, que son un tipo de redes neuronales que nos han permitido resolver el problema que llevamos acabo. Para ello necesitamos adentrarnos y tener claros diversos conceptos de machine learning y es en este capítulo donde los trataremos.

Podemos resumir el concepto de machine learning como un campo de la informática en la que se desarrollan algoritmos capaces de aprender patrones de los datos, sin haber sido explícitamente programado para ellos.

Los algoritmos de Machine Learning se dividen en dos categorías principales.

- Aprendizaje supervisado. Estos algoritmos tienen los datos etiquetados, tienen la solución deseada para cada dato, a estas soluciones se les conoce como labels. Un ejemplo típico de modelo supervisado es un clasificador de la bandeja de spam. El algoritmo es entrenado con datos que han sido etiquetados con anterioridad si se trata de spam o no, y cuyo objetivo es aprender a clasificar nuevo emails.
- Aprendizaje no supervisado. A diferencia de los anteriores, estos no tienen los datos etiquetados. Estos algoritmos suelen ser utilizados para detectar agrupaciones, detección de anomalías, problemas de visualización y reducción de dimensional.

5.1. El perceptron

Basandose en la idea de una neurona biológica Frank Rosenblatt desarrollo entre 1950 y 1960 la neurona artificial conocida como perceptron en el paper *The perceptron: A probabilistic model*

for information storage and organization in the brain. [16]. El perceptron es una función que toma varios *inputs* $x_1, \cdots, x_n$ de tipo binario y devuelve un *output* tambien de tipo binario.

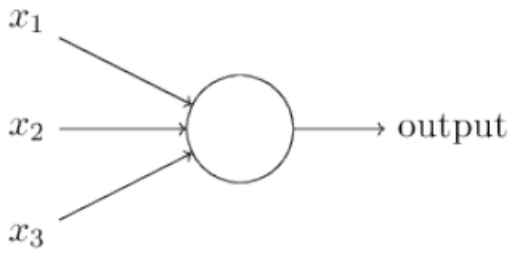


Figura 5.1: Esquema de perceptron, x_1, x_2, x_3 son tres *inputs* de tipo binario y el *output* también es binario. [13]

F.Rosenblatt propuso el compute del *output* introduciendo pesos $w_1, \cdots, w_n$, números reales, que determinan la importancia de cada input. El *output* tiene debía tener como valor 0 o 1, este se termina bajo la función 5.1.

$$output = \begin{cases} 0 & \sum_j^n w_j x_j \leq threshold \\ 1 & \sum_j^n w_j x_j > threshold \end{cases} \tag{5.1}$$

El *threshold* es un valor entre 0 y 1, deforma que si la suma es inferior a ese valor el *output* sera 0 y 1 en el caso contrario. El *threshold* era determinado con anterioridad.

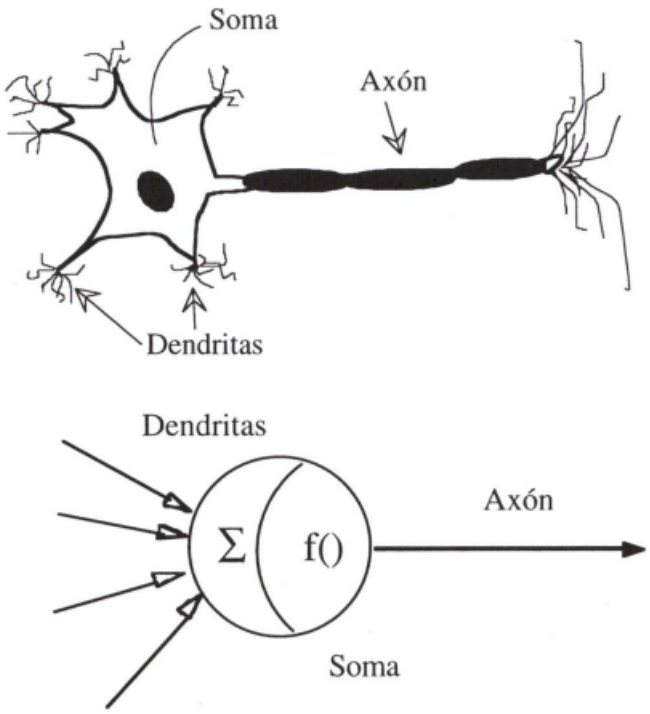


Figura 5.2: Comparación de una neurona real y un perceptron. [10]

De esta forma el perceptron es una función que ajustando los pesos y el *threshold* puede tomar decisiones en base a los valores del output. Es obvio que el perceptron no es un modelo que sea capaz de tomar decisiones como las que puede tomar el cerebro humano es por ello que necesitamos mayor complejidad en nuestro modelo. Por ende surge la idea de crear una red de perceptrones que puedan tomar mejores decisiones.

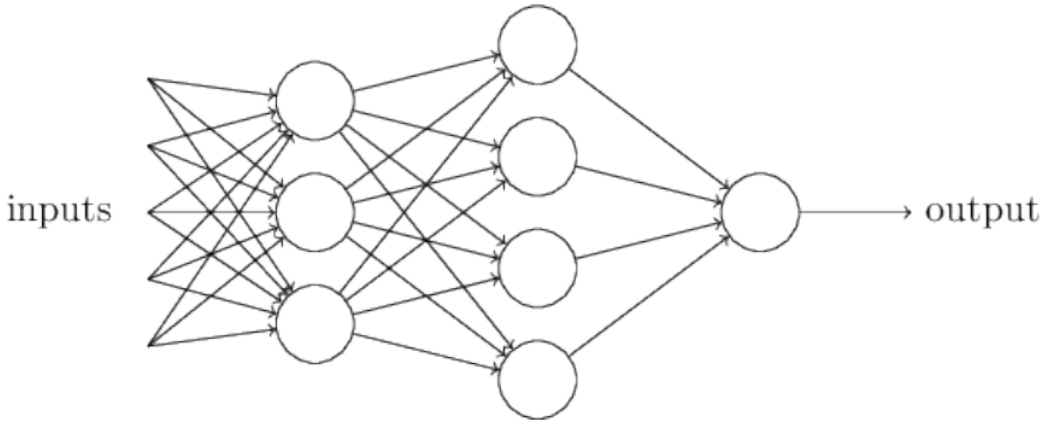


Figura 5.3: Red de perceptrones [13]

La primera capa de perceptrones toma decisiones en base los *inputs* que recibe, sin embargo la segunda capa tiene que tomar decisiones mas complejas ya que se basan en los valores abstractos de la primera capa y así en cada una de las capas que vayamos añadiendo. Por lo tanto, el último perceptron toma decisiones en base a todas las capas anteriores lo que permite mayor abstracción.

5.1.1. Arquitectura de una red neuronal

Supongamos que tenemos una red de perceptrones que queremos que aprenda a resolver un problema en concreto. En la sección 5.1 hemos introducido el perceptron y hemos dicho que tiene unos pesos que son los que determinan el *output* de la red. Nosotros queremos que nuestra red sea capaz de ajustar esos pesos automáticamente siendo capaz de clasificar correctamente el output.

Podemos comenzar con un ajuste de los pesos de forma aleatoria o predeterminada, la clave para que nuestra red sea capaz de aprender es realizar pequeños cambios en estos pesos he ir evaluando como afectan al *output* e ir ajustando para obtener *output* cada vez más óptimos.

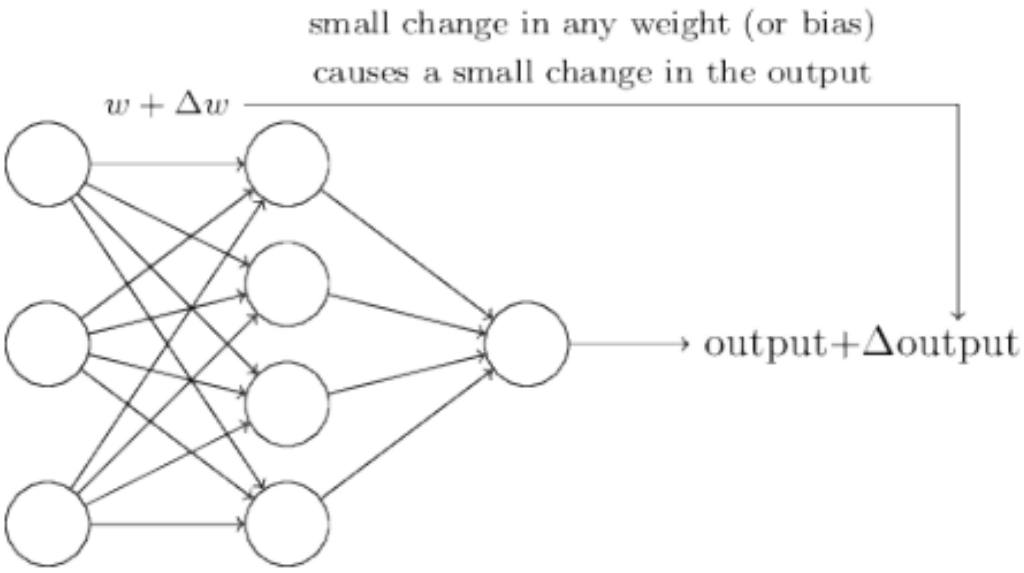


Figura 5.4: Proceso de ajuste de los pesos de los perceptrones [13]

5.2. Redes neuronales artificiales

El nombre de redes neuronales artificiales proviene de la idea de construir una maquina inteligente basándose en la arquitectura del cerebro.

Entre sus capacidades podemos encontrar la clasificación de billones de imágenes de Google, reconocimiento de voz e incluso ganar a juegos como el Go o el ajedrez.

Una red neuronal esta formada por diversas capas de neuronas. Una capa que recibe las entradas *(input layer)*, diversas capas a las que llamaremos capas ocultas *(hidden layers)* y una capa final *(output layer)*. Si todas las neuronas de una capa están conectadas con todas las neuronas de la siguiente capa, lo llamaremos capas densas.

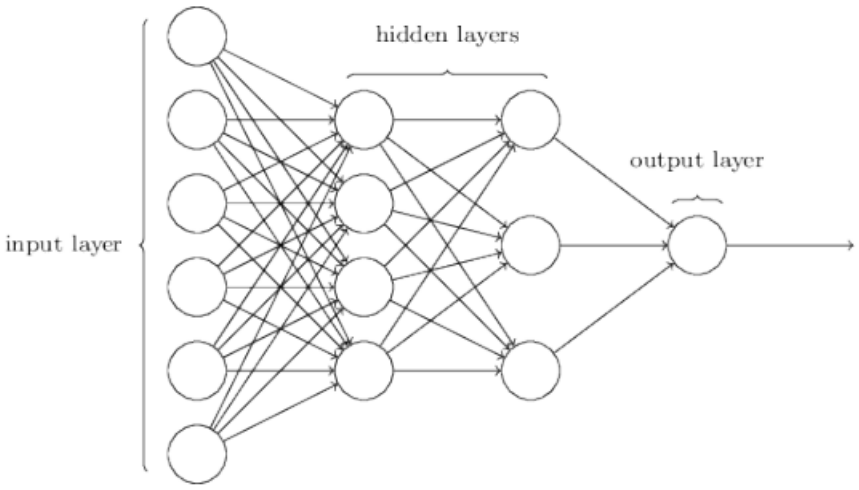


Figura 5.5: Red neuronal [13].

Si hay una gran cantidad de capas ocultas, lo llamaremos una red neuronal profunda (DNN). Las DNN son modelos con gran capacidad de generalización.

Como hemos mencionado anteriormente, el principal objetivo de los algoritmos de *machine learning* es ser capaz de realizar predicciones lo mas realistas posibles. Es equivale a reducir la tasa de error. Esto lo hacemos con algoritmos de descenso de gradiente, el algoritmo funciona con dos pasadas a través de la red neuronal, primero hace una predicción y mide el error que se ha cometido. Después va capa por capa midiendo el error cometido en cada conexión y realiza un paso en descenso de gradiente ajustando los pesos de cada neurona con tal de reducir el error. De esta forma vamos consiguiendo que nuestra red sea capaz de resolver el problema.

5.2.1. Capas de activación

Las funciones de activación son funciones que tras cada capa de neuronas evalúan sus *outputs* y se los pasan a la siguiente capa. Estas funciones de activación permitirán ajustar nuestro modelo y sus salidas.

Hay dos tipos de capas de activación las no lineales y las lineales. Para las redes neuronales se utilizan comúnmente las funciones no lineales.

Funciones no lineales

- Sigmoide.

  Esta función transforma los valores introducidos en una escala de 0 a 1.

$$sigmoide(x) = \frac{1}{1 + e^x} \tag{5.2}$$

  Su uso principal es el obtener probabilidades debido a que tiende asintóticamente a 1 y a 0.

$$\lim_{x \to \infty} sigmoide(x) = 1$$
$$\lim_{x \to -\infty} sigmoide(x) = 0$$

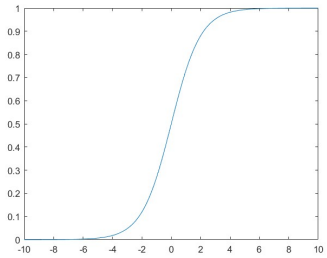


Figura 5.6: Función de activación Sigmoide

- Tangente hiperbólica. también conocida como Tanh, transforma los valores en un rango entre $[-1, 1]$.

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{5.3}$$

  Y de forma similar a como pasaba con la función sigmoide, tiende asintóticamente a 1 y -1. Es por ello que se utiliza en capas de clasificación.

$$\lim_{x \to \infty} tanh(x) = 1$$
$$\lim_{x \to -\infty} tanh(x) = -1$$

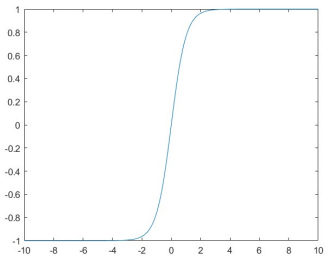


Figura 5.7: Función de activación tanh

Funciones Lineales

- ReLU - Rectified Linear Unit

$$ReLU(x) = \begin{cases} x \ si \ x > 0 \\ 0 \ otro \ caso \end{cases} \tag{5.4}$$

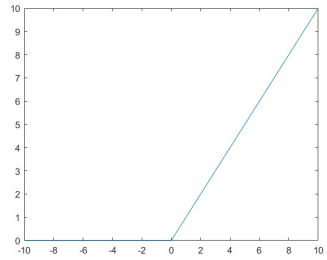


Figura 5.8: Función de activación ReLU

Se trata de una función identidad para los valores positivos y nula para los valores negativos. Esta permite un aprendizaje muy rápido ya que su gradiente es 1 si son positivos y 0 si son negativos. sin embargo, eso puede generar que algunas neuronas se mueran ya que si el gradiente podría ser 0, con el descenso de gradiente no variarían. Es por ello que hay una opción alternativa llamada LeakyReLU.

- LeakyReLU

$$ReLU(x) = \begin{cases} x \ si \ x > 0 \\ \lambda \ otro \ caso \end{cases} \tag{5.5}$$

Que ha diferencia de la función ReLU, su gradiente no será 0.

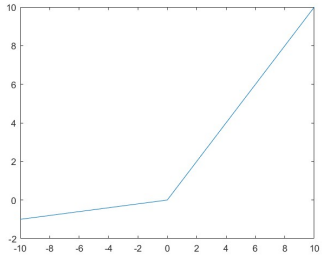


Figura 5.9: Función de activación LeakyReLU

5.2.2. Batch Normalization

El algoritmo de backpropagation recorre desde la capa de *output* hasta la capa de *input* , propagando el error gradiente. Una vez ya tenemos el gradiente de la función de coste correspondiente a cada parámetro de la red, aplicamos un paso del descenso de gradiente para actualizar los parámetros.

Desafortunadamente, los gradientes suelen tender a ser mas pequeños cada vez que el algoritmo progresa hasta las capas mas bajas. Eso hace que esas capas no se actualicen y los parámetros no cambien, haciendo que el algoritmo no converja a la buena solución. También puede ocurrir lo contrario, que el gradiente sea mas grande cada vez y haciendo que algunas capas tengan pesos muy grandes haciendo que el algoritmo diverja. Estos problemas son conocidos como *the vanishing/exploding gradients problems*.

En 2015, Sergey Ioffe y Cristian Szegedy, propusieron en un paper, una técnica llamada Batch Normalization [6] que solucionaba *the vanishing/exploding gradients problems*. Consiste añadir una operación antes del modelo o después de cada capa de activación que consiste en centrar en el cero, normalizar cada *input* y luego escalarlo y cambiar el resultado. Permitiendo así que el modelo aprenda la escala y media óptima de cada capa.

Sergey Ioffe y Cristian Szegedy demostraron que esta técnica mejora considerablemente el proceso de entrenamiento de las redes neuronales, en concreto con los algoritmos de clasificación de imágenes.

En una red neuronal, *the batch normalization* se hace fijando las medias y las variaciones de cada lote en la entrada de cada capa. Intuitivamente, la normalización se debería hacer en base a todo el conjunto de datos, pero no es tan eficiente como si restringimos la normalización a cada minilote (*mini-batch*) a lo largo del proceso de entrenamiento.

Sea $\mathfrak{B}$ un minilote de tamaño m del conjunto de entrenamiento. Calculamos su media empírica y la varianza de $\mathfrak{B}$ como:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{5.6}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{5.7}$$

Y para capa de la red con una dimensión d de entrada, $x = (x^{(1)}, ..., x^{(d)})$. Para cada dimensión de su entrada la normalizamos por separado.

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}}, \quad k \in [1, d], \quad \epsilon \text{ constante pequeña} \tag{5.8}$$

Lo que resulta en que la activación normalizada tiene media cero y varianza uno si $\epsilon = 0$.

Para posteriormente recuperar las representaciones utilizamos una función/transformación:

$$y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)} \tag{5.9}$$

siendo $\gamma^{(k)}$ y $\beta^{(k)}$ parámetros que se aprenden a posteriori en el proceso de optimización.

Si entrenamos nuestra red neuronal utilizando *batch normalization* esta será capaz de aprender y llegar a resultados mas óptimos en menos iteraciones. Sergey Ioffe y Cristian Szegedy lo demostraron en su artículo *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* [6] para el caso de clasificación de carácteres con la base de datos *MNIST*.

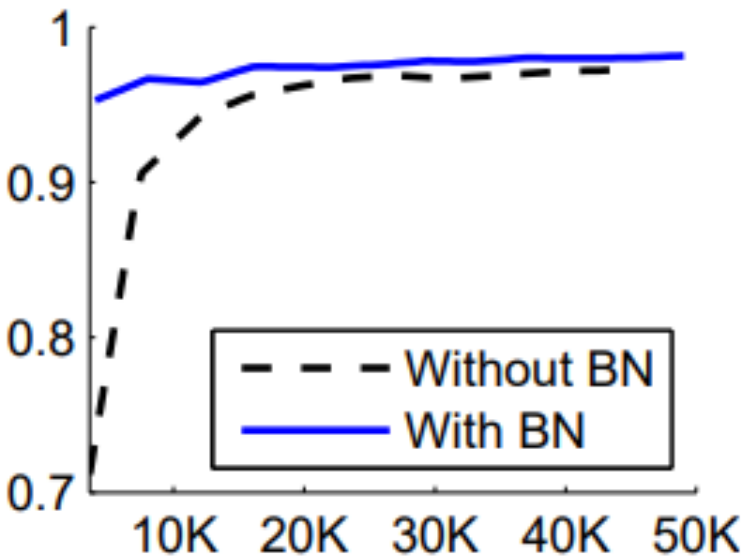


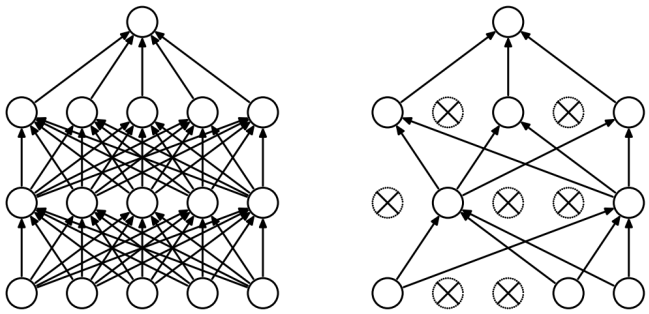
Figura 5.10: Test de acierto de la base de datos *MNIST* sin usar *Batch Normalization* y usando *Batch Normalization* a lo largo de las iteraciones de aprendizaje [6]

5.2.3. Dropout

Es una técnica de regulación que fue propuesta por Nitish Srivastava en 2014 en un paper titulado *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* [17].

Este algoritmo consiste en que en cada paso de entrenamiento, cada neurona tiene una probabilidad p de estar incluida en ese paso de entrenamiento, siendo ignorada en ese paso y reactivada en el siguiente paso. Una vez el entrenamiento a concluido, todas las neuronas estarán incluidas. Esto hace que las neuronas no puedan adaptarse a las neuronas que tienen a su alrededor y tengan que ser eficientes por ellas solas, prestando mucha atención a cada uno de sus *inputs* correspondientes. Permitiendo así que sean menos sensibles ligeros cambios en los *inputs* , consiguiendo una red mas robusta.

Para entenderlo mejor, si cada neurona puede estar presente o no en cada uno de los pasos del entrenamiento, eso quiere decir que hay un total de $2^{n^\circ \ de \ neuronas}$ posibles redes neuronales que en cada paso del entrenamiento se van compartiendo los pesos. La red neuronal final es un ensamblado de todas las todas las redes neuronales anteriores.



(a) Standard Neural Net (b) After applying dropout.

Figura 5.11: a) Una red neuronal con dos capas ocultas densas. b) La misma red neuronal tras haber aplicado dropout. [17]

Además, estas capas ayudan a reducir el error. Esto genera gran impacto en los resultados obtenidos de la red. En el paper *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* [17] hicieron el experimento con diversas bases de datos y redes neuronales a las que primero le aplicaron *Dropout* y luego no, viendo asi si los resultados mejoraban.

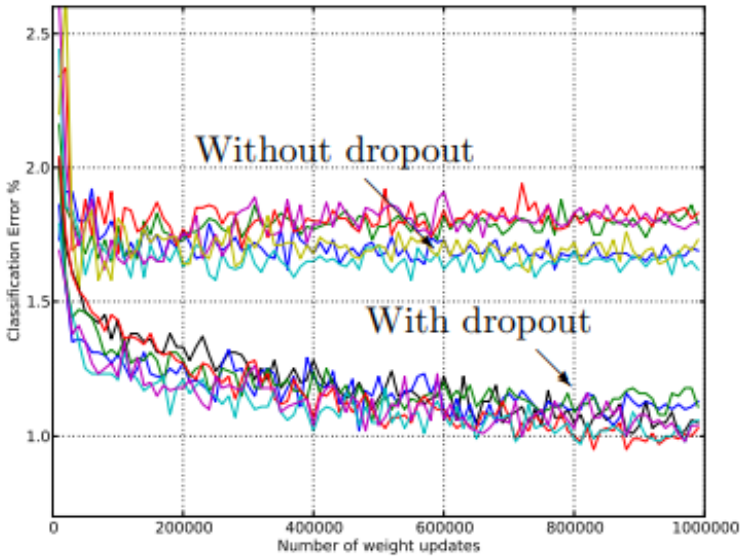


Figura 5.12: Test de error a diversas arquitecturas con y sin *dropout* a diversas bases de datos. [17]

5.2.4. Binary Cross Entropy Loss

Es necesario saber como nuestro modelo (red neuronal), esta estimando y haciendo las predicciones. Muchas veces, el objetivo de nuestro entrenamiento es hacer un modelo que estime altas probabilidades para casos positivos ($y = 1$) y bajas probabilidades para casos negativos ($y = 0$). Es por ello que podemos establecer la siguiente función como función de coste.

$$Loss(x) = \begin{cases} -\log(\hat{p}) & y = 1 \\ -\log(1 - \hat{p}) & y = 0 \end{cases} \tag{5.10}$$

Siendo x el *input* de nuestro modelo, es decir, el predictor, cuya predicción denotamos por $\hat{p}$ que es una probabilidad y y es su valor real.

Esta función es consistente ya que si $\hat{p} \approx y$ obtenemos que $Loss(x) \approx 0$. Por lo contrario, si los casos positivos los predice con valores cercanos a 0, la función se dispara y lo mismo pasa para los casos negativos si los predice con valores cercanos a 1.

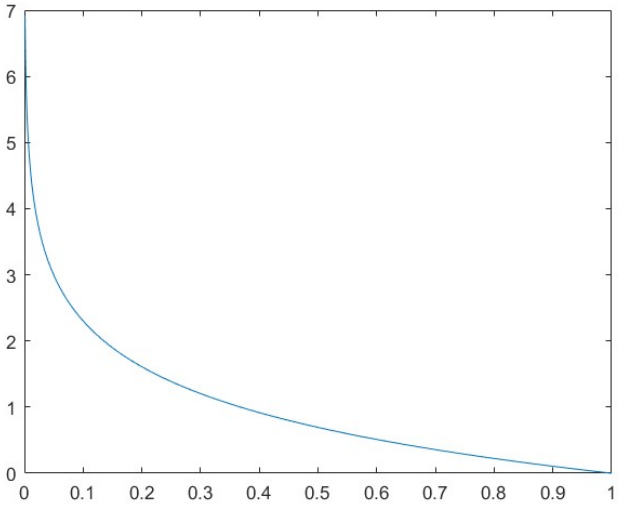


Figura 5.13: Función de pérdida, $Loss(x)$

Esta función de perdida sirve para un caso, y a nosotros nos interesa medir el error a través de toda la base de datos de entrenamiento. Para ello realizaremos la media. Podemos formularlo con la siguiente función a la que llamaremos como *log loss*.

$$J(x) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\hat{p}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{p}^{(i)})] \tag{5.11}$$

El problema que tiene esta función de coste es que existe forma de averiguar que x minimiza la función de coste. Sin embargo, se trata de una función convexa por lo que el descenso de gradiente garantiza encontrar el mínimo global.

5.2.5. Overfitting

Suele ser un problema bastante común en los algoritmos de machine learning y hay que tenerlo en consideración. Esto se da si el modelo predice muy bien los datos de entrenamiento, pero sin embargo, no ha sido capaz de generalizar bien a ejemplos que no ha visto antes.

Modelos mas complejos como pueden ser las redes neuronales profundas son capaces de detectar patrones en los datos, pero si los datos tienen ruido o algunos casos tienen poca representación en el conjunto de datos, el modelo puede detectar patrones en el ruido que luego no se generalizan en nuevos casos.

Hay diversas formas de atajar este problema, entre ellas, las mas frecuentes son:

1. Ajustar los parámetros del modelo haciendo el modelo mas simple y reduciendo su capacidad de aprendizaje.
2. Ampliar la base de datos con casos que antes tenían poca representación.
3. Reducir el ruido de los datos, eliminación de outliers e incluso corrección de errores.

5.2.6. Underfitting

Es el caso contrario al overfitting, esto ocurre si nuestro modelo no es lo suficientemente complejo como para aprender los patrones de nuestra base de datos. Se dice que el modelo es demasiado simple, por ello nuestras predicciones futuras no reflejaran la realidad e incluso con la base de datos propia.

Algunas opciones para solucionar este problema pasan por:

1. Crear un modelo mas complejo, con mas parámetros.
2. Proporcionarle mejores datos al modelo. Mejores características de lo que queremos predecir.
3. Reducir las restricciones que se le hayan impuesto al modelo. Como pueden ser: aumentar el tiempo de entrenamiento o aumentar capacidad de calculo computacional.

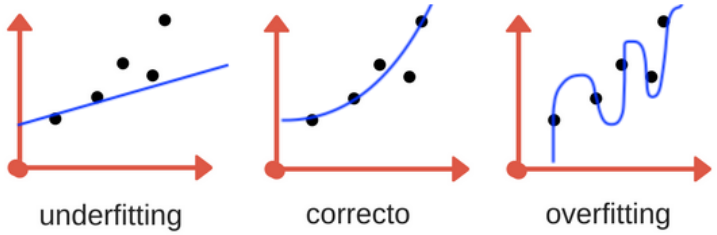


Figura 5.14: Ejemplos de overfitting y underfitting. [12]

5.2.7. Prueba y validación

La forma de saber si nuestro modelo a aprendido a generalizar bien es poniendolo a prueba con datos con los que no ha sido entrenado. Por lo general, la mejor opción es dividir la base de datos en dos: la de entrenamiento (training set) y la de prueba (test set).

Durante el entrenamiento, los parámetros del modelo se ajustan con el training set, se van probando con el test set. De esta forma se puede evaluar la eficacia de nuestro modelo y no solo eso, también podemos saber en que casos es mas o menor eficiente.

De esta forma podemos detectar si nuestro modelo sufre de overfitting, si la tasa de error con los datos de entrenamiento es baja, pero con los de prueba es alta.

5.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales, o también conocidas por sus siglas en inglés (CNN), han sido usadas para el reconocimiento de imágenes, clasificación de vídeos, condución autonoma, reconocimiento de voz y muchas mas áreas.

5.3.1. Capa Convolucional

Estas capas tienen la característica de que no todas las neuronas de la primera capa están conectadas con cada píxel de la imagen. Para cada píxel solo serán algunas neuronas las que estarán conectadas con él. Para las siguientes capas, trabajamos de forma análoga, no todas las neuronas de una capa están conectadas con todas las neuronas de la siguiente, cada neurona de la siguiente capa esta conectada con solo algunas neuronas que están en rectángulo de la anterior. Esto permite que la red se concentre en ciertas características de la imagen y lo transmita por las siguientes capas.

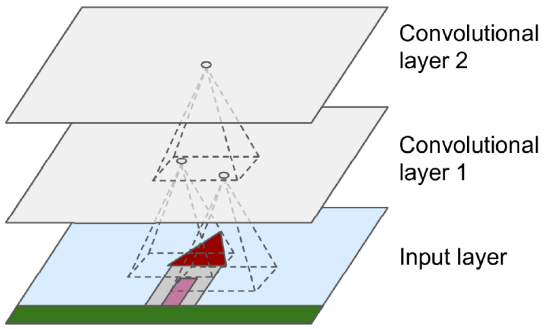


Figura 5.15: Esquema de una capa convolucional extraido del libro [3]

Debido a que no todas las capas tienen porque tener la misma cantidad de neuronas que las anteriores o posteriores, existen diversas técnicas para que se pueda realizar la segmentación de la imagen.

Si tenemos una capa convolucional mas pequeña que la anterior, por lo general se le añaden ceros en la frontera. A esta técnica se le conoce como *zero-padding*.

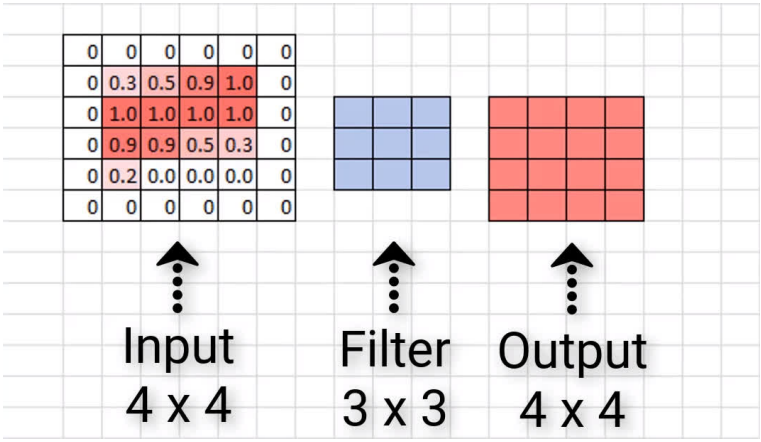


Figura 5.16: Ejemplo de *zero-padding* extraído de [18]

Y si lo que queremos es reducir la dimensionalidad, lo que aplicamos es una técnica llamada *stride*. Exactamente se trata de un filtro que modifica la cantidad de píxeles que se analizan en cada movimiento a través de la imagen como se puede ver en la Figura 5.17.

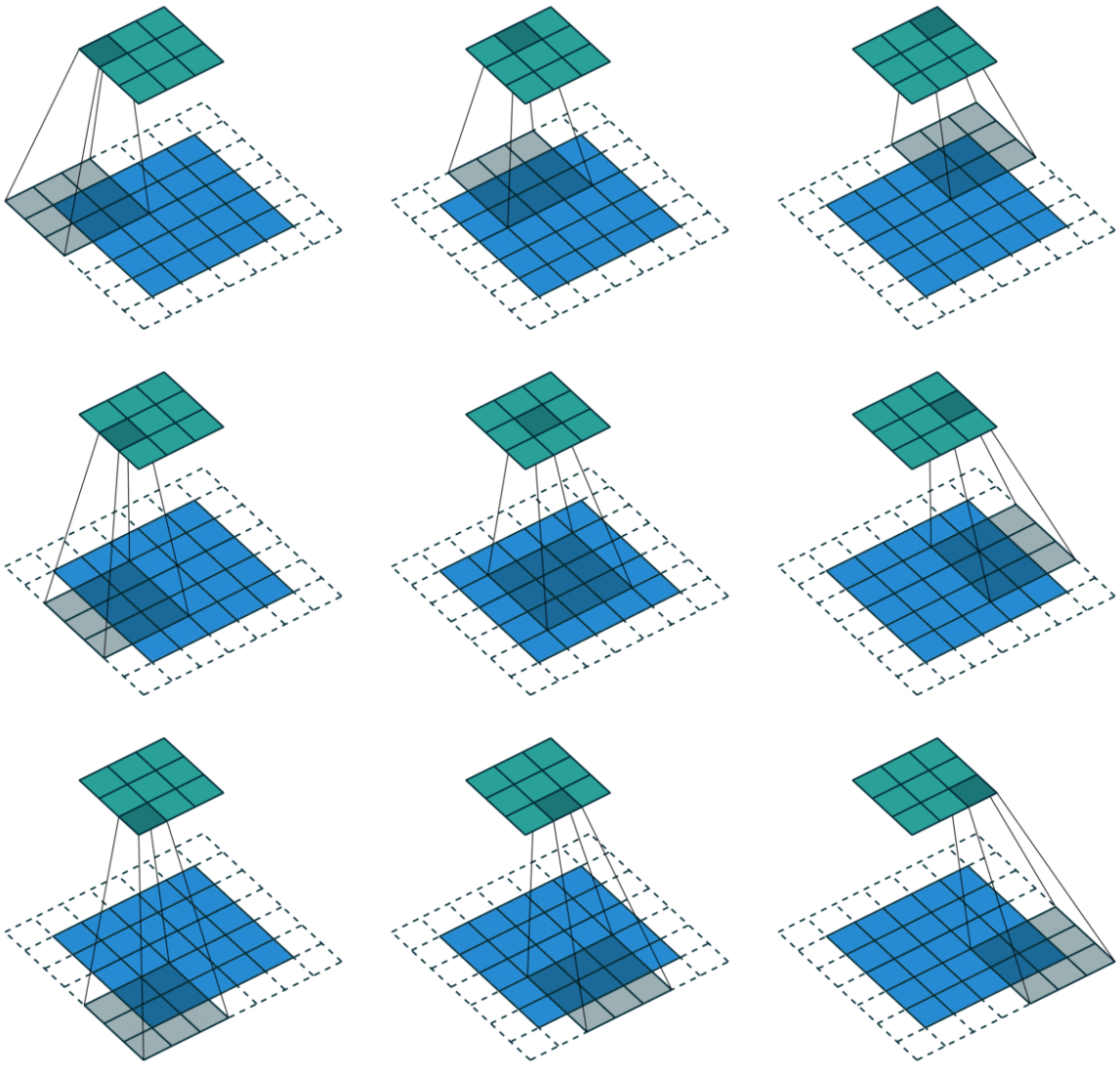


Figura 5.17: Convolucionando un kernel 3×3 sobre un 5×5 input con un padding de 1×1 usando un stride de 2×2 [2].

5.4. Redes Generativas Antagónicas

Las redes generativas antagónicas (GANs), o también conocidas como GANs, se desarrollaron en 2014 y se propusieron el paper *Generative Adversarial Networks* [5]. Estas redes son capaces de detectar patrones en los datos de entrenamiento y generar datos nuevos que compartan características. Haciendo referencia a la teoría de juegos, en realidad se trata de dos redes que compiten entre sí en juego de suma cero. A estos dos modelos se les conoce debido a sus

características y objetivos como el modelo generador y el discriminador.

El modelo discriminador desempeña la tareas de clasificar y en general etiquetar los datos. El discriminador tiene acceso a las imágenes reales y las sintéticas generadas por el generador, y su objetivo es ser capaz de diferenciar cuales son reales y cuales son sintéticas.

Por el contrario, el modelo generador tiene la tareas de generar imágenes o datos nuevos a partir de una muestra aleatoria a la que llamaremos ruido. Es importante tener en cuenta que el generador no tiene acceso a las imágenes reales, el solo va a aprendiendo a base de interactuar con el discriminador. Haciendo referencia a la competición, el generador quiere engañar al discriminador y que las catalogue como reales las imágenes sintéticas que genera.

En una GAN básica, el discriminador es una función que caracteriza si una imagen es real o no en base a una probabilidad, $\mathfrak{D}(x) \longrightarrow (0, 1)$. Si tenemos un generador, $\mathfrak{G}$, el discriminador $\mathfrak{D}$, entrenará clasificando como 0 si la imagen es falsa y 1 si es real.

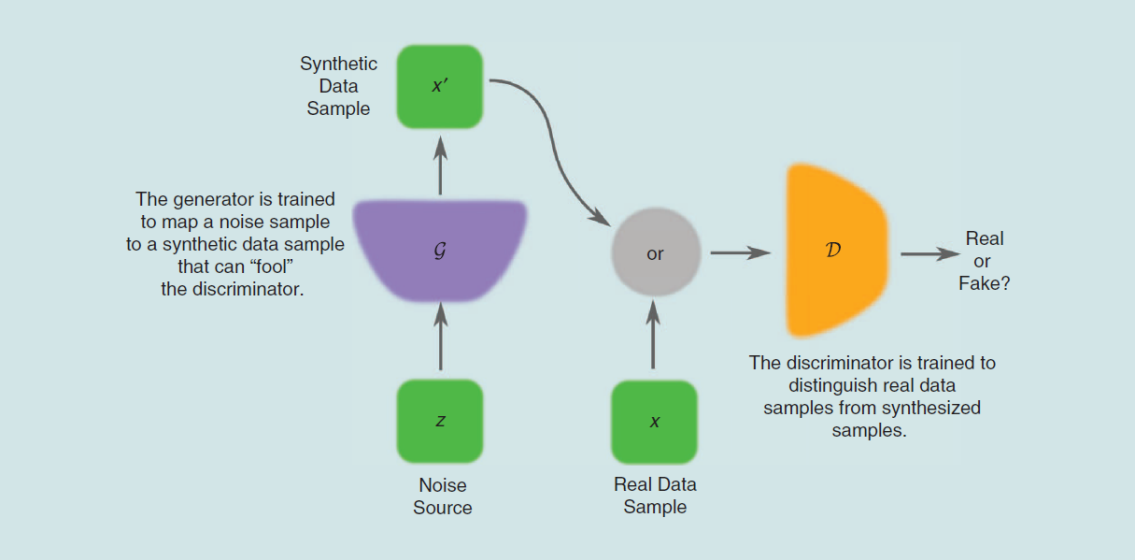


Figura 5.18: Estructura de una GAN [1]

5.4.1. Modelo de generación de arterias aortas

Para entrenar nuestro modelo dividimos las bases de datos en paquetes o lotes de forma aleatoria, a estos paquetes de datos se les llaman *batches*. Haremos un total de 100.000 fases de entrenamiento, a las que llamaremos épocas o en inglés *epochs*. En cada época, nuestros modelos evaluaran cada uno de los *batches*. Así es como se define de forma general como se puede observar en el libro [4].

Algorithm 1 Algoritmo de la fase de entrenamiento de una GAN

for EPOCH **do**
  for IMAGE_BATCH FROM DATASET **do**
    Generar un conjunto de m vectores de ruido gaussiano $Z = (z^{(1)}, \ldots, z^{(m)})$.
    Utilizar el $\mathfrak{G}$ para sintetizar una muestra $\mathfrak{G}(Z) = (\mathfrak{G}^{(1)}, \ldots, \mathfrak{G}^{(m)})$ utilizando Z.
    Calculamos la función de pérdida del $\mathfrak{D}$ con $X = IMAGE\_BATCH$ y $\mathfrak{G}(Z)$.

$$Loss(\mathfrak{G}(Z), X) = \frac{1}{m}\sum_{i=1}^{m}[\log(\mathfrak{D}(x^{(i)}) + \log(1 - \mathfrak{D}(\mathfrak{G}(z^{(i)}))))]$$

    Calculamos la función de pérdida del generador utilizando $\mathfrak{D}(\mathfrak{G}(Z))$.

$$Loss(\mathfrak{G}(Z)) = \frac{1}{m}\sum_{i=1}^{m}[\log(1 - \mathfrak{D}(\mathfrak{G}(x^{(i)})))]$$

    Actualizamos el discriminador y el generador por gradiente estocástico.
  end for
end for

Para cuantificar el error de cometen nuestros modelos utilizamos la función *Binary Cross Entropy Loss* explicada en la sección 5.2.4

5.4.2. Generador con capas convolucionales

Nuestro modelo generador tiene como *input* un vector aleatorio de 100 componentes con una distribución normal con media 0 y desviación estándar 1.

La capa de entrada (*input layer*) es una capa densa con 6400 neuronas. En la siguiente capa aplicamos *Batch Normalization* y continuamos con una capa LeakyReLU de activación. Reordenamos el *output* con la función reshape a matrices 5×5.

Ahora que ya tenemos una matriz, podemos empezar a dotar de información espacial, de forma que dos puntos que son vecinos deben de tener cierta información en común. Aplicamos una capa convolucional con 128 filtros, con un ancho y alto de 5×5 en la convolución y stride 1. Volvemos a aplicar *Batch Normalization* y añadimos una capa LeakyReLU de activación.

Repetimos el proceso, aplicamos otra capa convolucional pero esta vez con 68 filtros, con un ancho y alto de 5×5 en la convolución y stride 2, aplicando *Batch Normalization* y LeakyReLU.

Para terminar, utilizamos una capa convolucional con un filtro y ajustado de forma que obtengamos imágenes con las mismas dimensiones que las de la base de datos y una capa de activación de tangente hiperbólica de forma que los valores que devuelve el modelo estén entre

$[0, 1]$.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 6400)	640000
batch_normalization (BatchNormalization)	(None, 6400)	25600
leaky_re_lu (LeakyReLU)	(None, 6400)	0
reshape (Reshape)	(None, 5, 5, 256)	0
conv2d_transpose (Conv2DTranspose)	(None, 5, 5, 128)	819200
batch_normalization_1 (BatchNormalization)	(None, 5, 5, 128)	512
leaky_re_lu_1 (LeakyReLU)	(None, 5, 5, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 10, 10, 64)	204800
batch_normalization_2 (BatchNormalization)	(None, 10, 10, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 10, 10, 64)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 20, 20, 1)	256

Total params: 1,690,624
Trainable params: 1,677,440
Non-trainable params: 13,184

Figura 5.19: Resumen del modelo generador

Como se indica en la Figura 5.19, el modelo generador tiene un total de 1.690.624 de parámetros de los cuales 1.677.440 serán los que se van ajustando conforme se va entrenando la red.

5.4.3. Discriminador con capas convolucionales

El modelo de discriminador es un modelo sencillo. Se trata de un modelo secuencial, al igual que el generador, pero no tiene tantas capas.

Su capa de entrada *input layer*, es una matriz con las dimensiones como las de la base de datos y las que genera el generador. Comenzamos con una capa convolucional con 64 filtros y una

venta de 5×5 con un stride de 2. Aplicamos LeakyReLU y añadimos una capa de dropout con una probabilidad del 30 %.

Repetimos con otra capa convolucional con 128 filtros con una ventana de 5×5 y un stride de 2. Volvemos a aplicar una capa de LeakyReLU y un dropout del 30 %.

Para terminar, transformamos la matriz en un vector aplicando la función Flatten y con una capa densa de una neurona con una función de activación sigmoide para simular una función de probabilidad. De esta forma, nuestro modelo discriminador devolverá como 0 las imágenes que considere como falsa y 1 si las considera como reales.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 10, 10, 64)        1664

 leaky_re_lu_3 (LeakyReLU)    (None, 10, 10, 64)        0

 dropout (Dropout)            (None, 10, 10, 64)        0

 conv2d_1 (Conv2D)            (None, 5, 5, 128)         204928

 leaky_re_lu_4 (LeakyReLU)    (None, 5, 5, 128)         0

 dropout_1 (Dropout)          (None, 5, 5, 128)         0

 flatten (Flatten)            (None, 3200)              0

 dense_1 (Dense)              (None, 1)                 3201

=================================================================
Total params: 209,793
Trainable params: 209,793
Non-trainable params: 0
_________________________________________________________________
```

Figura 5.20: Resumen del modelo discriminador

Como se puede observar en la Figura 5.20, este modelo tiene 209.793 parámetros de los cuales todos se irán ajustando conforme se entrena el modelo.

Capítulo 6

Resultados

El objetivo planteado en este trabajo es saber la influencia de las capas convolucionales para la generación de aortas sintéticas. Como hemos mencionado en 4.1.2, para saber si las aortas que nuestro modelo esta generando son plausibles, es decir, si pueden corresponder a la población, haremos mediciones a los biomarcadores y compararemos sus distribuciones.

Para evitar que hubiera un sobreajuste en nuestro modelo, ampliamos nuestra base de datos inicial aplicando técnicas de boostraping hasta obtener una nueva base de datos de 3.000 aortas y volvimos a reentrenar nuestro modelo durante 100.000 épocas.

6.1. Entrenamiento

En el proceso de entrenamiento de la GAN no tenemos un indicador que nos permita saber cuando debemos parar de entrenar nuestro modelo. Es de gran interés tener un indicador cuantitativo que nos permita saber que nuestro modelo ya cumple ya el objetivo propuesto y que podamos evitar el *overfitting*. Actualmente la única condición de parada que tenemos en nuestro entrenamiento es que se hagan 100.000 épocas. Tratando de solventar este problema hemos ido almacenando cada 10.000 épocas el modelo generador, el discriminador, 1.000 aortas sintéticas y las funciones de perdida del generador y del discriminador.

Un factor a tener en cuenta es la definición de *batch* que dimos en 5.4.1, para la base de datos original tenemos un *batch size* de 17, es decir, 2 *batches* de 17 aortas. Sin embargo, para la base de datos remuestreada tenemos un *batch size* de 300, es decir, 10 *batches* de 300 aortas. Esto ocasiona que en cada época la cantidad de evaluaciones que hacen el discriminador y el generador sean diferentes dependiendo de la base de datos.

Observando las distribuciones de algunos biomarcadores vamos a discutir en que época los modelos ya han alcanzando el objetivo propuesto y si seguimos entrenando comenzamos a ver *overfitting*.

6.1.1. Criterio de parada

Cada 10.000 épocas hemos generado 1.000 aortas sintéticas a las que hemos medido sus biomarcadores. Discutamos en que época las distribuciones de frecuencia de cada biomarcador comienzan a no tener la misma distribución que la base de original. Las distribuciones de frecuencia de los biomarcadores de 1.000 aortas sintéticas por la GAN entrenada con la base de datos original.

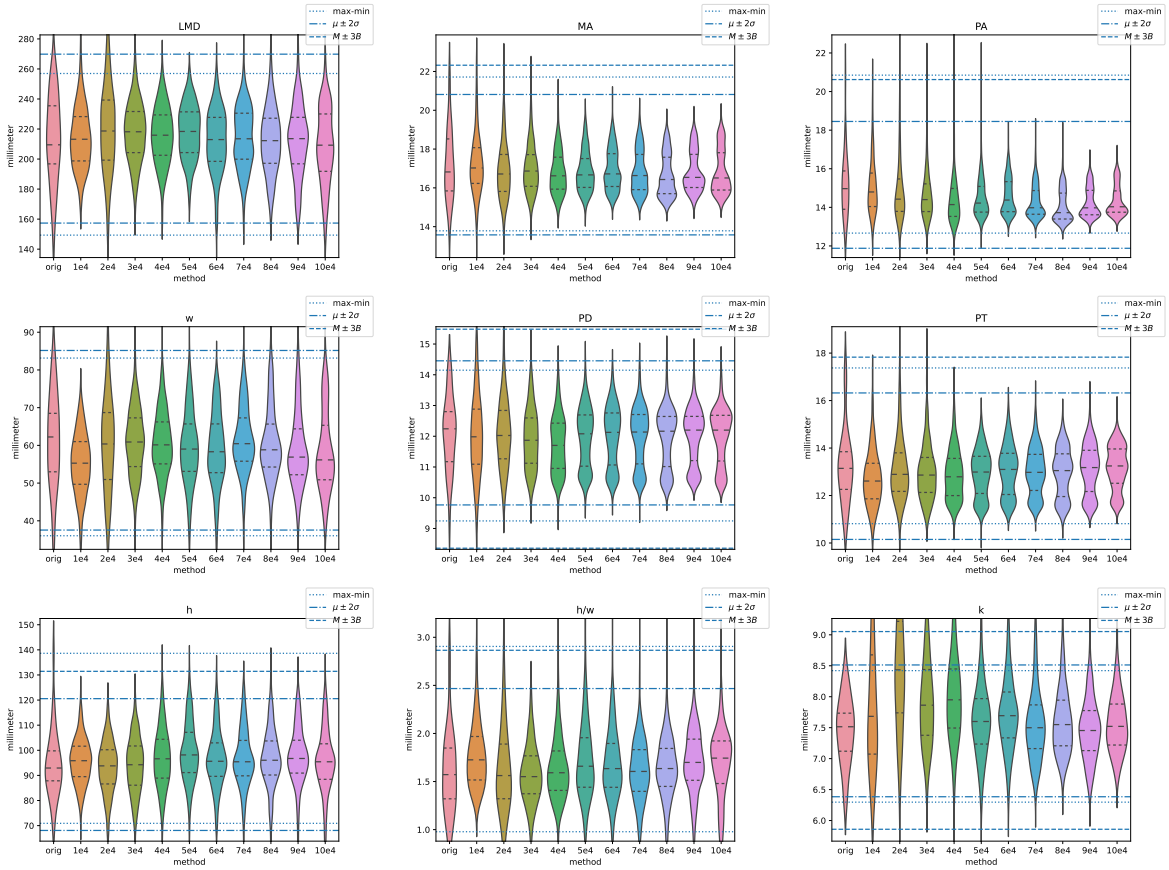


Figura 6.1: *Violin plots* de cada biomarcador de la base de datos original, de 1.000 aortas generadas con la GAN entrenada con la base de datos a la que se le había hecho un PCA y de 1.000 aortas generadas por la GAN convolucional entrenada con la base de datos original cada 10.000 épocas.

Un signo de que se esta ocasionando *overfitting* puede ser que la distribucion de frecuencia comienza a tener una distribución biomodal. En la Figura 6.1 observamos que en biomarcadores como LMD, MA, PA, PD, PT y w, apartir de la época 30.000 comenzamos a observar como los violin plots tienden a tener esa distribución bimodal, sin embargo el resto de biomarcadores

mantienen su forma a lo largo de las épocas. Por lo tanto decidimos que la época de parada para la GAN que es entrenada con la base de datos original es el época 30.000.

Las distribuciones de frecuencia de los biomarcadores de 1.000 aortas sintéticas por la GAN entrenada con la base de datos remuestreada.

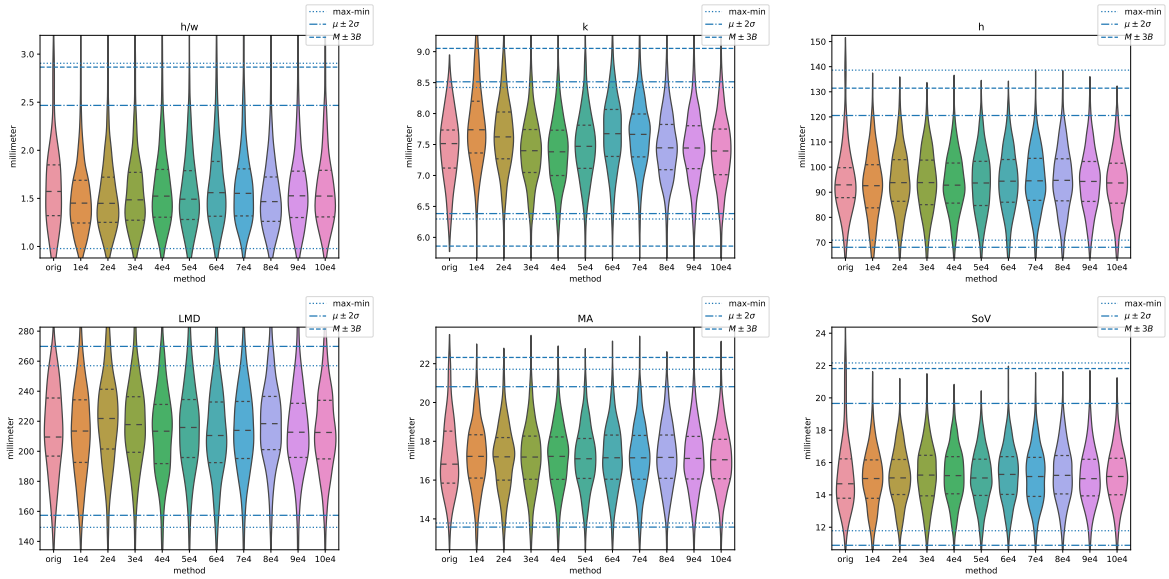


Figura 6.2: *Violin plots* de cada biomarcador de la base de datos original, de 1.000 aortas generadas con la GAN entrenada con la base de datos a la que se le había hecho un PCA y de 1.000 aortas generadas por la GAN convolucional entrenada con la base de datos remuestreada cada 10.000 épocas.

En la Figura 6.2 vemos que las distribuciones de frecuencia se ajustan muy bien desde el principio. Cabe mencionar que los cuartiles de los biomarcadores h/w y k varían en las primera épocas y se estabilizan en las últimas. Viendo esas dos figura decidimos que el época de parada debería ser 90.000

6.2. Arterias aortas sintéticas

Una vez definidos los criterios de parada de nuestros modelos, vamos a generar 1.000 aortas sintéticas con cada uno de los modelos y mediremos sus biomarcadores. Mostraremos algunas arterias que generan estos modelos y las compararemos basándonos en los biomarcadores.

6.2.1. Base de datos original

Nuestro modelo ha sido entrenado con 34 aortas que tenían aneurisma. Recordemos que queremos tener un modelo generador capaz de generar imágenes como las que muestra la Figura 4.5 las cuales eran los puntos de control de los B-spline, los que nos permite reconstruir aortas como las de la Figura 4.7.

Basándonos en la argumentación realizada en el apartado 6.1.1 realizamos el entrenamiento hasta 30.000 épocas.

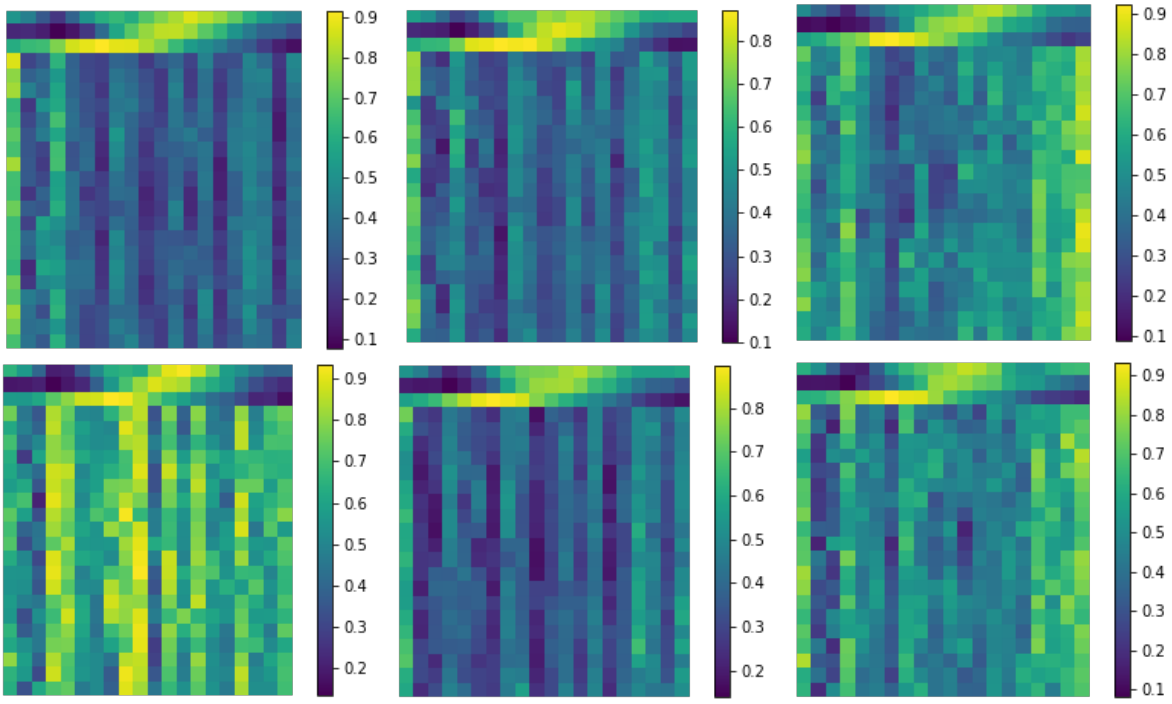


Figura 6.3: Seis representaciones de los puntos de control de arterias aortas generadas con una GAN tras 30.000 épocas.

Las representaciones de la Figura 6.3 podemos reconstruirlas en aortas utilizando las propiedades de los B-splines mencionados en el capítulo 3.4.1, su correspondiente reconstrucción la podemos ver en la Figura 6.4.

Figura 6.4: Reconstrucción de las representaciones de la Figura 6.3.

¿Como evalúa el discriminador las aortas generadas por el generador? Esta pregunta es de gran interés ya que el discriminador ha ido aprendiendo si las representaciones que genera el generador son arterias o no lo son.

Para responder esa pregunta, generamos 1.000 aortas sintéticas y las evaluamos en el discriminador. Obteniendo así histogramas como el de la Figura 6.5 y la Tabla 6.1. Podemos observar que mas del 75 % de las aortas sintéticas, el discriminador las evalúa por debajo de 0.2. Es decir, que gran parte las considera no reales, esto no quiere decir que realmente lo sean o no, simplemente es como las esta clasificando el discriminador.

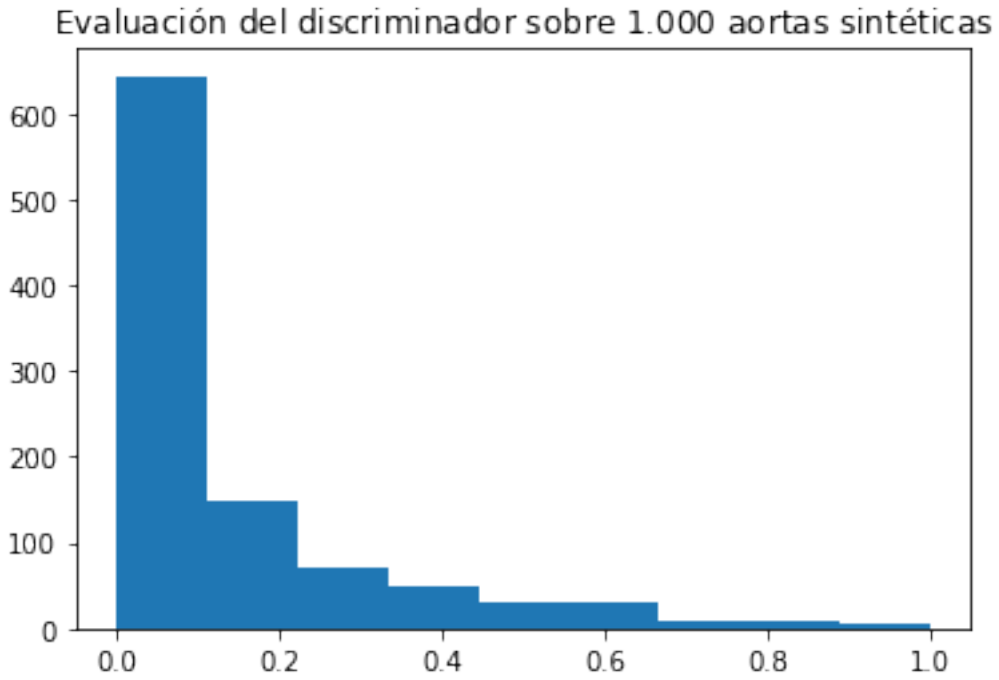


Figura 6.5: Histograma que resulta de la evaluación del discriminador sobre 1.000 aortas generadas por el generador.

Media	Std	Min	25 %	50 %	75 %	Max
0.14	0.17	0.005	0.03	0.07	0.17	0.93

Cuadro 6.1: Tabla resumen de la evaluación del discriminador sobre 1.000 aortas sintéticas.

¿Y las aortas de la base de datos, como las evalúa? Repitamos el proceso anterior.

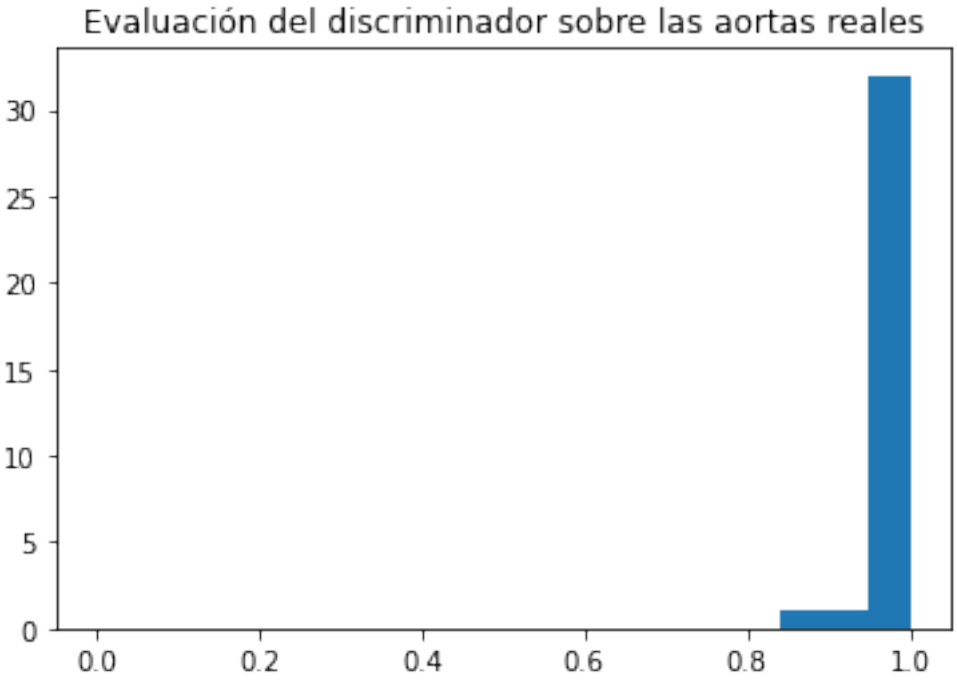


Figura 6.6: Histograma que resulta de la evaluación del discriminador sobre las aortas reales

Media	Std	Min	25 %	50 %	75 %	Max
0.99	0.03	0.84	0.98	0.99	0.99	1

Cuadro 6.2: Tabla resumen de la evaluación del discriminador sobre las aortas reales

De la Figura 6.6 y de la tabla 6.2 podemos decir que el discriminador si que sabe cuales son las aortas reales. Esto se puede deber a un sobreajuste de los parámetros en el discriminador ya que el discriminador ha evaluado todas las aortas reales en cada uno de los épocas y la base de datos original solo consta de 34 aortas, una cantidad muy pequeña para los grados de libertad que tienen nuestros modelos.

6.2.2. Base de datos remuestreada

Para evitar que hubiera un sobreajuste en nuestro modelo, ampliamos nuestra base de datos inicial aplicando técnicas de boostraping hasta obtener una nueva base de datos de 3.000 aortas y bajo la justificación realizada en el apartado 6.1.1 realizamos el entrenamiento hasta 90.000 épocas.

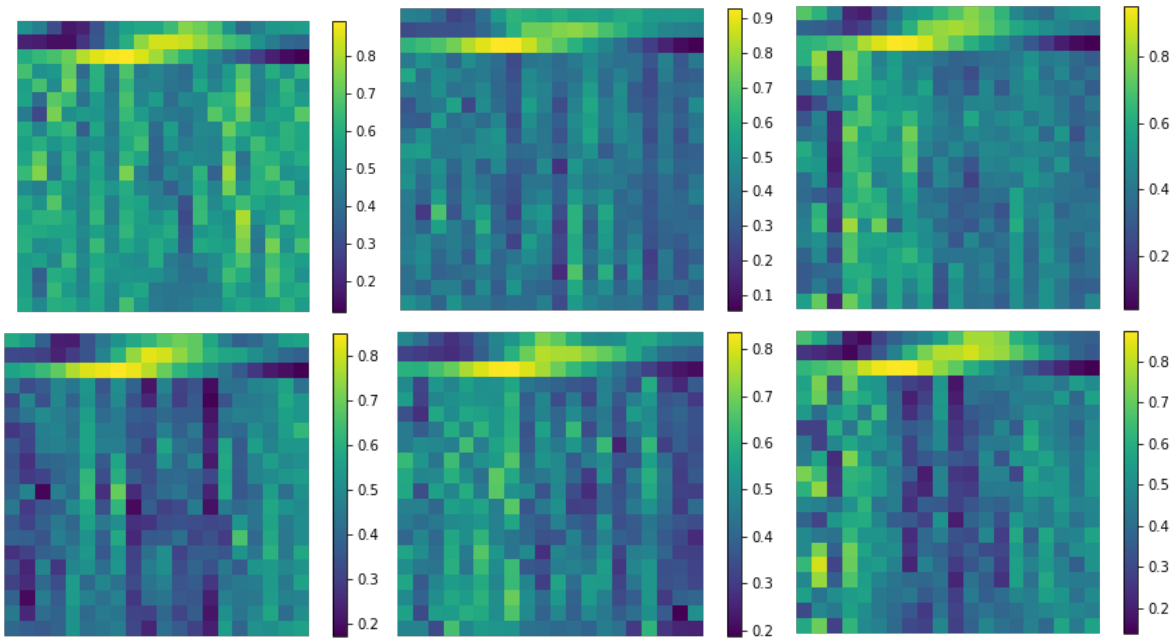


Figura 6.7: Seis representaciones de los puntos de control de arterias aortas generadas con una GAN tras 90.000 épocas.

Las representaciones de la Figura 6.7 podemos reconstruirlas en aortas, su correspondiente reconstrucción la podemos ver en la Figura 6.8.

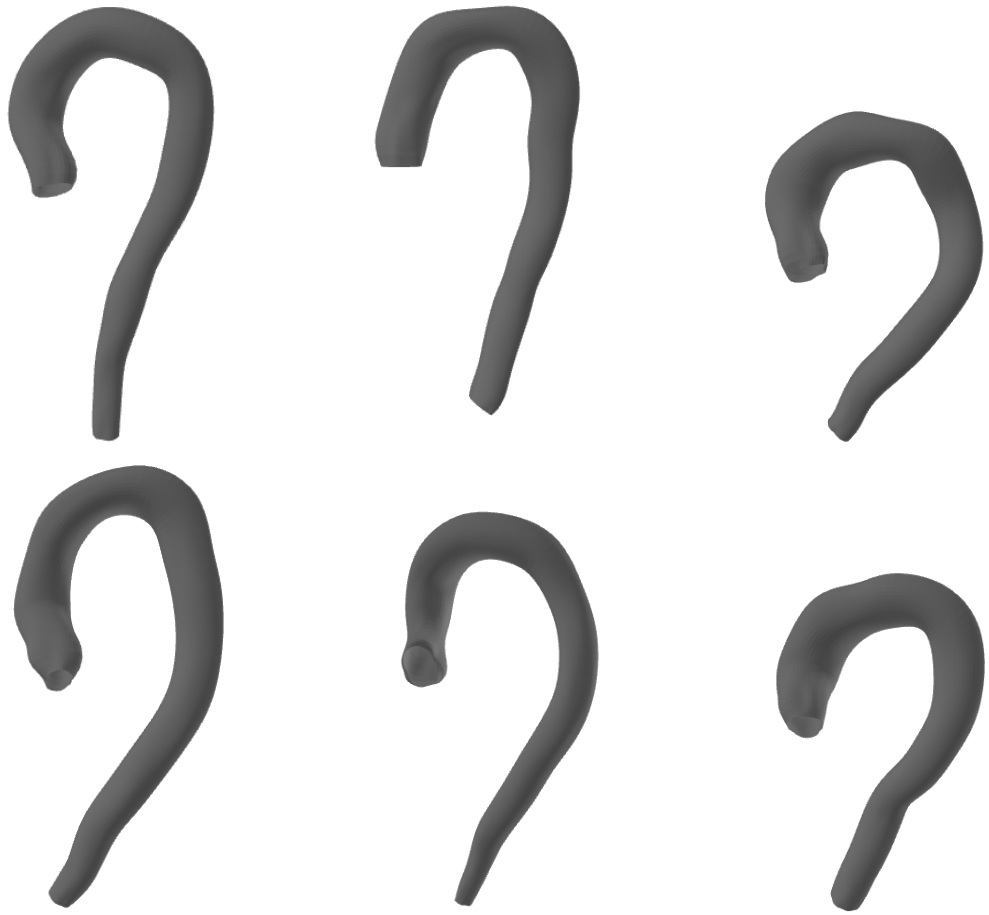


Figura 6.8: Reconstrucción de las representaciones de la Figura 6.7.

Repitamos el mismo proceso que en el capítulo 6.2.1 evaluando con el discriminador.

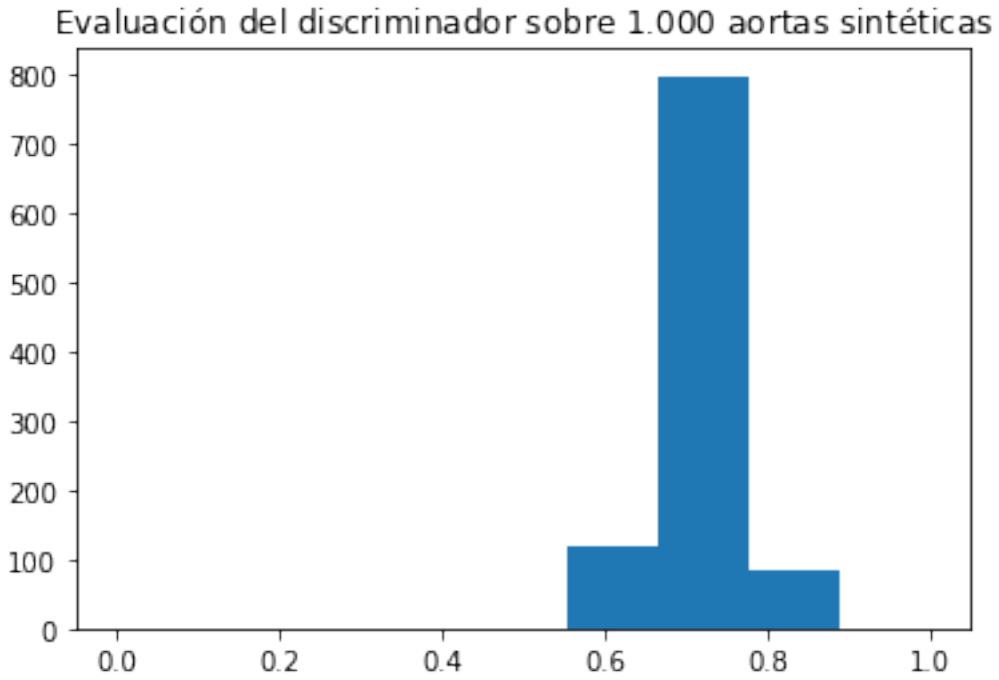


Figura 6.9: Histograma que resulta de la evaluación del discriminador sobre 1.000 aortas generadas por el generador.

Media	Std	Min	25 %	50 %	75 %	Max
0.71	0.04	0.58	0.68	0.71	0.74	0.86

Cuadro 6.3: Tabla resumen de la evaluación del discriminador sobre 1.000 aortas sintéticas.

La Figura 6.9 a diferencia de 6.5 tiene los datos mucho mas concentrados, además observamos que las evaluaciones son siempre superiores a 0.5 cosa que en 6.5 no pasaba.

Respecto a las tablas 6.3 y 6.1, estas nos indican que el discriminador esta evaluando con mayor puntación las aortas sintéticas generadas con las GAN que ha sido entrenado con la base de datos remuestreada. Esto se puede deber a dos posibles casos, que el generador sea mejor o que el discriminador no sea tan bueno diferenciando entre reales y falsas. Veamos que dice sobre las aortas reales.

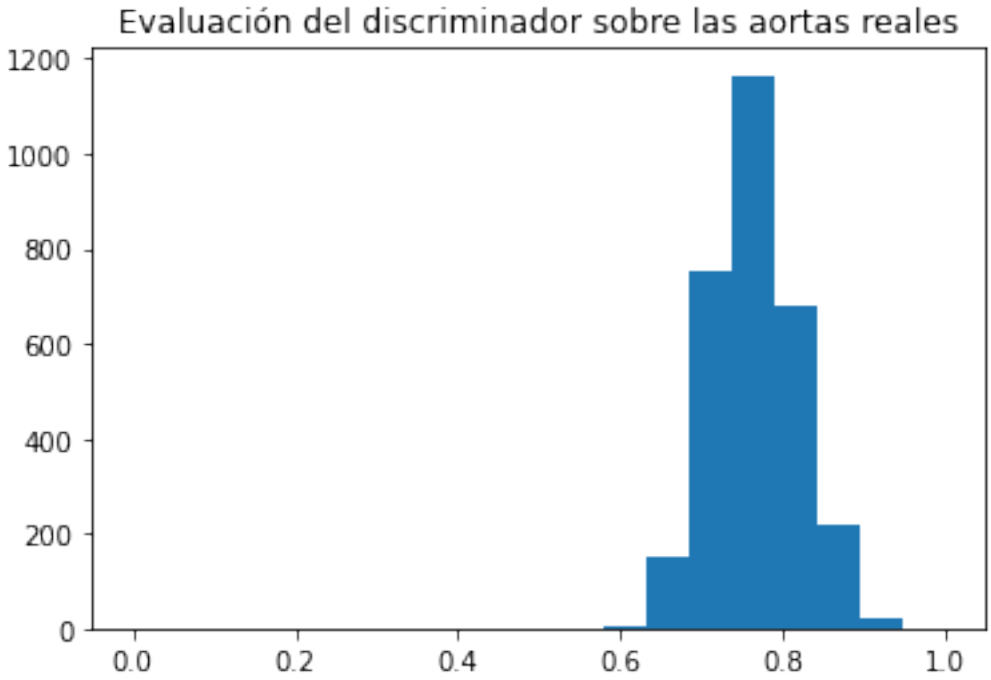


Figura 6.10: Histograma que resulta de la evaluación del discriminador sobre las aortas reales

Media	Std	Min	25 %	50 %	75 %	Max
0.77	0.05	0.61	0.73	0.76	0.80	0.93

Cuadro 6.4: Tabla resumen de la evaluación del discriminador sobre las aortas reales

Si comparamos la Figura 6.10 y la tabla 6.4 con la Figura 6.6 y la tabla 6.2, observamos que el discriminador no tiene tan claro si las aortas son reales. Esto se puede deber a que el generador que había sido entrenado con la base de datos original no generaba imágenes tan realistas, como se puede ver en el histograma 6.5 y por lo tanto al discriminador le era mas fácil diferenciar las reales de las sintéticas. Al haber aumentado la base de datos y el numero de épocas con la base de datos remuestreada, el generador ha sido capaz de hacer aortas mas realistas y es por ello que obtenemos valores por parte del discriminador inferiores en la Figura 6.10 y superiores en la Figura 6.9, en comparación con 6.6 y 6.5.

6.2.3. Comparación de resultados

Ahora que ya hemos decidido en que época parar de entrenar nuestro modelo para cada uno de la base de datos, comparemos los biomarcadores de 1.000 aortas generadas por las GANs convolucionales y la GAN no convolucional, con la base de datos original.

De la Figura 6.11 podemos decir que los biomarcadores de las aorta sintéticas generadas por las GANs convolucionales están centrados alrededor de la media de la base de datos original. Con la GAN no convolucional entrenada con la base de datos con un PCA, también ocurría en gran parte de los biomarcadores.

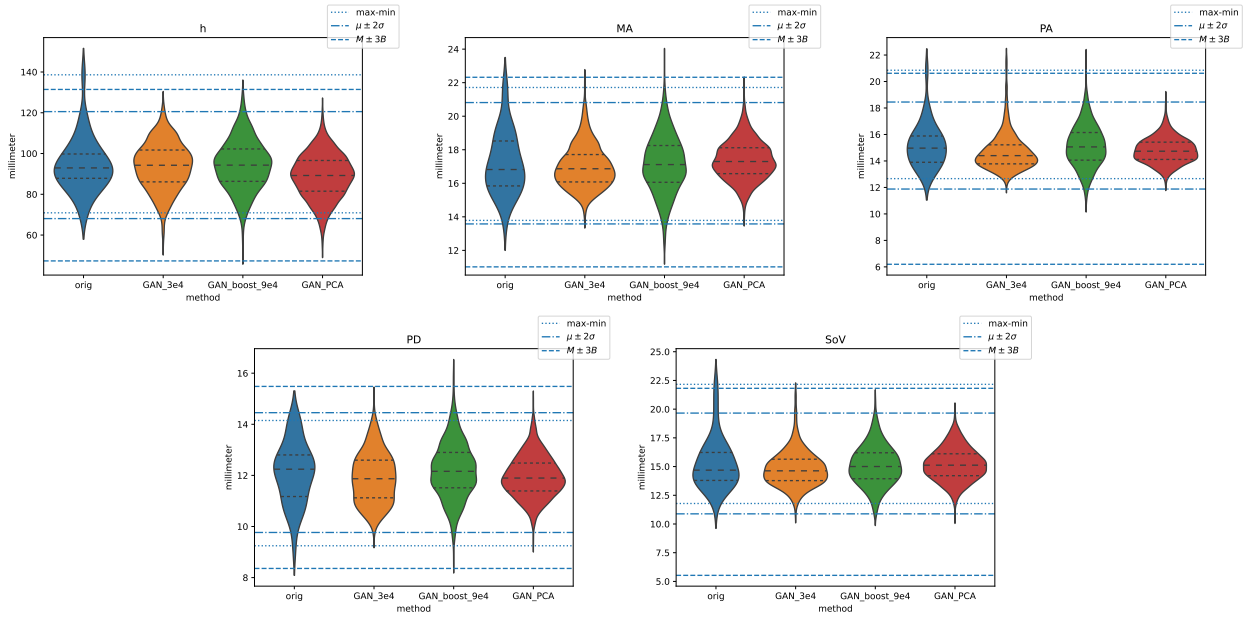


Figura 6.11: Distribuciones de frecuencia de los biomarcadores h, LMD ,MA, PA, PD, SoV para cada base de datos. Las aortas originales, 1.000 generadas por las GANs convolucionales y las del paper [15].

Sin embargo, la Figura 6.12 podemos decir que los biomarcadores h/w, k ,tor, w, las distribuciones tienen sus diferencias. Cabe destacar que en el biomarcador h/w la GAN entrenada con la base de datos remuestreada puede generar aortas con un gran ratio entre la altura y la anchura. Esto se puede observar en la Figura 6.8, tenemos aortas, como la cuarta que tienen un h alto, pero la anchura del arco es mas estrecha. Para el biomarcador k, la GAN entrenada con la base de datos original también podemos obtener valores muy altos, pudiendo generar aortas con una gran curvatura como puede ocurrir en la Figura 6.4. Comparadno los resultados a la GAN entrenada con la base de datos con un PCA vemos una gran mejora, los datos si que están alrededor de la media, lo cual no ocurría en el trabajo previo con estso biomarcadores.

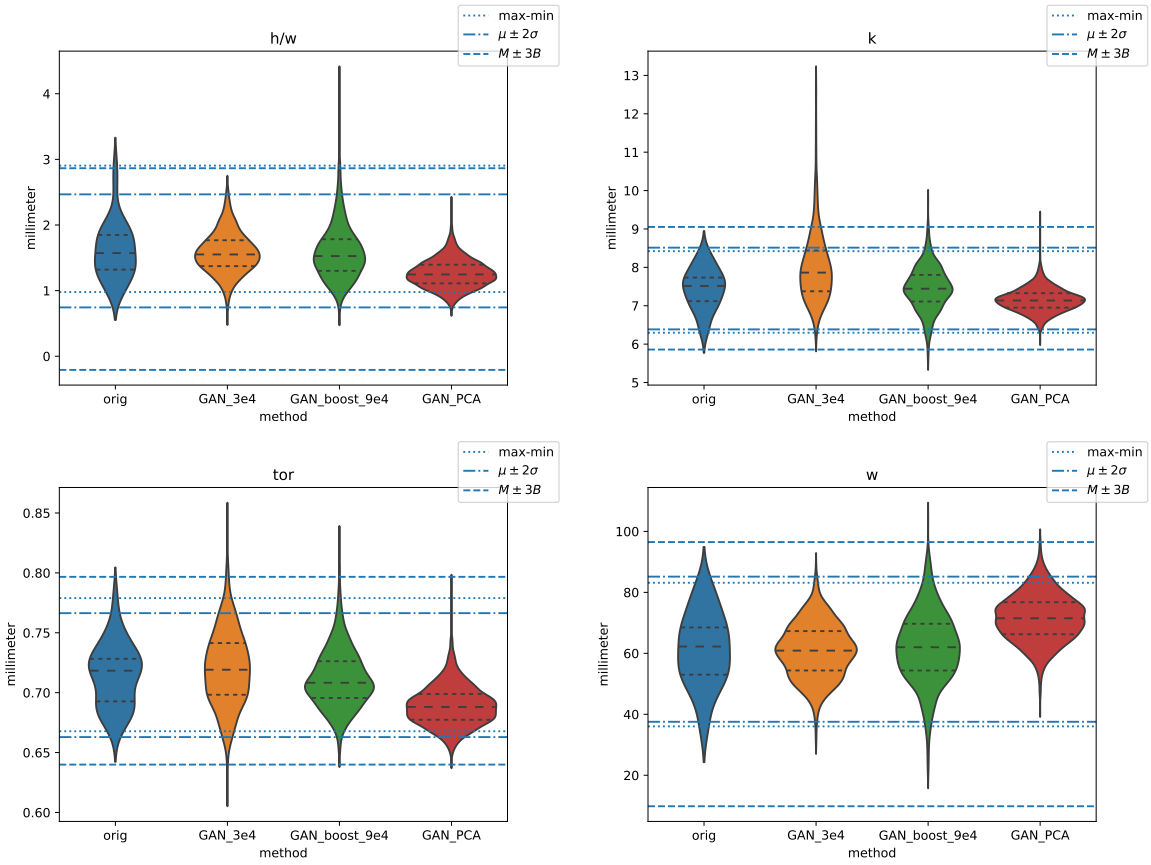


Figura 6.12: Distribuciones de frecuencia de los biomarcadores h/w, k ,tor, w, para cada base de datos. Las aortas originales, 1.000 generadas por las GANs convolucionales y las del paper [15].

Capítulo 7

Conclusión

La finalidad principal de este trabajo ha consistido en ser capaces de ampliar la base de datos original con 34 arterias aortas. Esto es de interés porque la obtención de imágenes médicas tiene un alto coste y no siempre se pueden llevar a cabo. Para tratar de solventar este problema, hemos tenido que obtener un representación de las aortas que reduzca su dimensionalidad y además que sea constante. Para ello utilizamos B-splines con los cuales teníamos un biyección entre cada representación bidimensional y cada aorta.

Esto nos ha permitido ajustar un modelo de *machine learning* conocida como red generativa antagónica la cual hemos entrenado y nos genera representaciones sintéticas bidimensionales la cuales podíamos reconstruir en aortas sintéticas. Es en este trabajo donde presentamos los resultados obtenidos entre los cuales se encuentran aortas sintéticas, las evaluaciones de los biomarcadores y la comparación entre diversas bases de datos originales y sintéticas. El objetivo secundario que nos planteamos en este trabajo era conocer el impacto que tienen las redes convolucionales a la hora de resolver este problema.

Cabe mencionar que hay distribuciones de frecuencia de los biomarcadores de las aortas generadas por las GANs convolucionales que tienen la misma correspondencia con la base de datos original que las generadas por la GAN no convolucional como se muestran en la Figura 6.11, sin embargo hay otros biomarcadores que su correspondencia es mayor con la base de datos original que las generadas por la GAN no convolucional del paper [15] como se puede ver en la Figura 6.12.

Esto nos puede estar indicando que las capas convolucionales tienen un impacto positivo en la generación de aortas sintéticas. Para poder sacar conclusiones queda como trabajo futuro la tercera columna de la Figura 4.10, entrenar con las aortas reales una GAN sin capas convolucionales y generar aortas sintéticas. Esto nos permitirá medir sus biomarcadores y realizar una comparación. Como trabajo futuro también queda obtener una medida cuantitativa que nos permita saber cuando parar de entrenar nuestro modelo, evitando así el sobreajuste que

podríamos estar cometiendo.

Por lo general, podemos decir que el objetivo principal de obtener un modelo que nos genere aortas sintéticas se ha cumplido. Añadir que parte de los resultados de este trabajo han sido presentados como ponencia en el *7th International Conference on Computational and Mathematical Biomedical Engineering (CMBE22)* [8].

Apéndice A

Código

Listing 1 Importamos las librerias necesarias

```
import tensorflow as tf
tf.test.gpu_device_name()
import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras import layers
import time
from tqdm import tqdm
import pandas as pd
import pyvista as pv

from tensorflow.keras.utils import plot_model
from feat_vector import AortaFeatureVector
```

Listing 2 Cargamos la base de datos.

```
base_de_datos = np.load('aortas/aortas_originales.npy')
base_de_datos = base_de_datos.astype('float64')
base_de_datos=np.expand_dims(base_de_datos, axis = -1)
```

Listing 3 Normalizamos la base de datos como se explica en 4.1.1.

```
minimo_filas=[]
maximo_filas=[]

minimo_column=[]
maximo_column=[]
for i in range(3):
    minimo_filas.append(np.amin(base_de_datos[:, i,:, 0]))
    maximo_filas.append(np.amax(base_de_datos[:, i,:, 0]))

for i in range(20):
    minimo_column.append(np.amin(base_de_datos[:, 3:,i, 0]))
    maximo_column.append(np.amax(base_de_datos[:, 3:,i, 0]))

minimo_filas=np.transpose(minimo_filas)
maximo_filas=np.transpose(maximo_filas)

minimo_filas=np.expand_dims(minimo_filas, axis = -1)
maximo_filas=np.expand_dims(maximo_filas, axis = -1)

minimo_column=np.expand_dims(minimo_column, axis = 0)
maximo_column=np.expand_dims(maximo_column, axis = 0)

normalizadas=np.empty((34,23,20))
for i in range(34):
    normalizadas[i,:3,:]=(base_de_datos[i, :3,:, 0]-minimo_filas)/
        (maximo_filas-minimo_filas)
    normalizadas[i,3:,:]=(base_de_datos[i, 3:,:, 0]-minimo_column)/
        (maximo_column-minimo_column)

normalizadas=np.expand_dims(normalizadas, axis = -1)
```

Listing 4 Función para desnormalizar una aorta.

```
def desnormalizar(imagen, maximo_filas=maximo_filas,minimo_filas=minimo_filas,
    maximo_column=maximo_column,minimo_column=minimo_column):

    original=np.empty((23,20))
    original[:3,:]=(imagen[:3,:]*(maximo_filas-minimo_filas)) + minimo_filas
    original[3:,:]=(imagen[3:,:]*(maximo_column-minimo_column)) + minimo_column

    return original
```

Listing 5 Función para desnormalizar una cantidad n de aortas.

```
def desnormalizar_cantidad(aortas):
    cantidad=len(aortas)
    aortas_sinteticas=np.empty((cantidad,23,20))
    acc=0
    while acc<cantidad:
        imagenes_generadas=desnormalizar(aortas[acc,:,:,0])
        aortas_sinteticas[acc]=imagenes_generadas
        acc=acc+1
    return aortas_sinteticas
```

Listing 6 Hiperparámetros del entrenamiento.

```
X_train=normalizadas
BUFFER_SIZE = 1000
BATCH_SIZE = 17
train_dataset = tf.data.Dataset.from_tensor_slices(X_train).
    shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
noise_dim = 100
```

Listing 7 Modelo generador.

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(5*5*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((5, 5, 256)))
    assert model.output_shape == (None, 5, 5, 256)  #None es el batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),
        padding='same', use_bias=False))
    assert model.output_shape == (None, 5, 5, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
        padding='same', use_bias=False))
    assert model.output_shape == (None, 10, 10, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 2), strides=(2, 2),
        padding='valid', use_bias=False, activation='sigmoid'))
    assert model.output_shape == (None, 23, 20, 1)

    return model
```

Listing 8 Modelo discriminador.

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                    input_shape=[23, 20, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1,activation='sigmoid'))

    return model
```

Listing 9 Funciones de perdida del modelo generador y del discriminador.

```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

Listing 10 Entrenamiento en una época.

```
@tf.function
def train_step(images):

    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss,
        generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss,
        discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator,
        generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
        discriminator.trainable_variables))

    return gen_loss, disc_loss
```

Listing 11 Función de entrenamiento.

```
def train(dataset, epochs):

  vector_gen_perdida=[]
  vector_dis_perdida=[]
  for epoch in tqdm(range(epochs)):
    start = time.time()

    for image_batch in dataset:
      gen_perdida,dis_perdida=train_step(image_batch)
      vector_gen_perdida.append(np.mean(gen_perdida.numpy()))
      vector_dis_perdida.append(np.mean(dis_perdida.numpy()))

    if (epoch+1) % 10000 == 0:

      generator.save('modelos/generador_{:04d}.h5'.format(epoch+1))
      discriminator.save('modelos/discriminador_{:04d}.h5'.format(epoch+1))

      noise = tf.random.normal([1000, 100])
      imagenes_generadas = generator(noise, training=False)
      aortas_sint=desnormalizar_cantidad(imagenes_generadas)
      np.save('modelos/1000_aortas_sint_{}.npy'.format(epoch+1),aortas_sint)
```

Listing 12 Entrenamiento de los modelos.

```
EPOCHS = 100000
with tf.device('/GPU:0'):
    train(train_dataset, EPOCHS)
```

Listing 13 Función que reconstruye la aorta desde una imagen.

```
def image_to_aorta(imagen,nombre_stl='aorta.stl', maximo_filas=maximo_filas,
    minimo_filas=minimo_filas, n_slices=200 ,points_per_slice=100):

    imagen=desnormalizar(imagen, maximo_filas,minimo_filas)
    f = AortaFeatureVector()

    fv_arr = np.hstack([[16,16,16,500], imagen.ravel()])
    f.decompose_feature_vector(fv_arr=fv_arr, per=True)
    f.build_splines()
    a_mesh = f.create_aorta_mesh(n_slices,points_per_slice)

    wall = pv.PolyData(a_mesh.wall_polydata)
    p = pv.Plotter()
    p.add_mesh(wall)
    p.show()

    wall.save(nombre_stl)
```

Índice de figuras

Índice de cuadros

List of Listings

Bibliografía

[1] Antonia Creswell y col. «Generative Adversarial Networks: An Overview». En: *IEEE Signal Processing Magazine* 35.1 (2018), págs. 53-65. DOI: 10.1109/MSP.2017.2765202.

[2] Vincent Dumoulin y Francesco Visin. «A guide to convolution arithmetic for deep learning». En: *ArXiv e-prints* (mar. de 2016). eprint: 1603.07285.

[3] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019. ISBN: 9781492032618. URL: https://books.google.es/books?id=HHetDwAAQBAJ.

[4] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017. ISBN: 978-1491962299.

[5] Ian J. Goodfellow y col. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[6] Sergey Ioffe y Christian Szegedy. «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». En: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[7] *La aorta: El Pulso de la Vida*. Abr. de 2021. URL: https://www.goredforwomen.org/es/health-topics/aortic-aneurysm/your-aorta-the-pulse-of-life.

[8] Álvaro López y col. «Generation of virtual patient's aorta anatomy using Convolutional Neural Networks». En: *7th International Conference on Computational & Mathematical Biomedical Engineering*. CMBE. Milan, 2022.

[9] Tom Lyche y Knut Mørken. *Spline Methods Draft*. Department of Mathematics University of Oslo, 2018.

[10] Bonifacio Martín del Brío y Carlos Serrano Cinca. «Fundamentos de redes neuronales artificiales: hardware y software». En: *Scire: representación y organización del conocimiento* 1.1 (jun. de 1995), págs. 103-125. DOI: 10.54886/scire.v1i1.1036. URL: https://www.ibersid.eu/ojs/index.php/scire/article/view/1036.

[11] Cinthia Serrano MD. *Aorta*. 2022. URL: https://www.kenhub.com/es/library/anatomia-es/aorta-es.

[12] Na8. *Qué es overfitting y underfitting y cómo solucionarlo*. Dic. de 2017. URL: https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/.

[13] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: http://neuralnetworksanddeeplearning.com/.

[14] Marina Piccinelli y col. «A Framework for Geometric Analysis of Vascular Structures: Application to Cerebral Aneurysms». En: *IEEE Trans. Medical Imaging* 28.8 (2009), págs. 1141-1155. DOI: 10.1109/TMI.2009.2021652. URL: https://doi.org/10.1109/TMI.2009.2021652.

[15] Pau Romero y col. «Clinically-Driven Virtual Patient Cohorts Generation: An Application to Aorta». En: *Frontiers in Physiology* 12 (2021). ISSN: 1664-042X. DOI: 10.3389/fphys.2021.713118. URL: https://www.frontiersin.org/articles/10.3389/fphys.2021.713118.

[16] F. Rosenblatt. «The perceptron: A probabilistic model for information storage and organization in the brain.» En: *Psychological Review* 65.6 (1958), págs. 386-408. ISSN: 0033-295X. DOI: 10.1037/h0042519. URL: http://dx.doi.org/10.1037/h0042519.

[17] Nitish Srivastava y col. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». En: *Journal of Machine Learning Research* 15.56 (2014), págs. 1929-1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[18] *Zero padding in Convolutional Neural Networks explained*. URL: https://deeplizard.com/learn/video/qSTv_m-KFk0.