DEEP LEARNING PARA LA GENERACIÓN DE IMÁGENES HISTOPATOLÓGICAS REALISTAS MEDIANTE ARITMÉTICA DE VECTORES CONCEPTUALES

Autor:
Rubén Fernández Blanco

Directores:
Esteban Vegas Lozano
Ferran Reverter Comes

Máster en Bioinformática y Bioestadística
Trabajo Final de Máster – Machine Learning

Título del trabajo:	Deep Learning para la generación de imágenes histopatológicas realistas mediante aritmética de vectores conceptuales
Nombre del autor:	Rubén Fernández Blanco
Directores:	Esteban Vegas Lozano Ferran Reverter Comes
Fecha de entrega (mm/aaaa):	06/2019
Titulación:	Máster en Bioinformática y Bioestadística
Área del Trabajo Final:	Machine Learning
Idioma del trabajo:	Castellano
Palabras Clave	Deep Learning, Redes Generativas Antagónicas, Histopatología
Abstract	Generative Adversarial Networks (GANs) can offer a way to tackle chronic lack of labeled samples in the medical imaging field, not only using unbounded generation but also by using conditional generation on different attributes of our choice or by modifying the results of this generation with the application of conceptual arithmetic operations between images. We train a Deep Convolutional GAN and a conditional DCGAN on a breast cancer dataset and we make use of some of its properties in order to edit histopathological images by arithmetically operating its latent vectors. The breast cancer dataset is created out of several patients WSI images, and it contains annotations for positive and negative samples. By using this arithmetical properties, we are able to perform several operations on the images, like inversion of the class of a sample (from tumorous to normal or vice versa), smooth interpolations of two samples that can show the transition between two states or two types of them or transference of features from one sample to another by combining additions and substractions of latent vectors. Aside from its utility to augment labeled datasets for supervised algorithms, this type of edition may be useful in other situations, for example being used as didactic or informative material or as a way to deal with the frequent privacy and anonymity problems that can be encountered when working with this type of medical data.

Índice general

1 Introducción

1.1. Contexto y motivación

Deep Learning en imagen médica. Las técnicas de Deep Learning, y especialmente las Redes Neuronales Convolucionales (CNNs), han encontrado un gran campo de aplicación en el ámbito de la imagen médica. Estos métodos se muestran muy adecuados para la clasificación o detección de distintas condiciones médicas, en tareas de segmentación de órganos, células u otras características visuales importantes para el diagnóstico de patologías a partir de imágenes. Todo esto, con precisiones similares o superiores a las de patólogos expertos[1] pero sobro todo, de manera rápida y automatizada.

Limitaciones. Hasta la aparición de las Redes Generativas Antagónicas (GANs) la utilización de algoritmos de Deep Learning para el análisis de imágenes médicas se centraba en métodos de aprendizaje supervisado. Este tipo de entrenamiento implica la utilización de una gran cantidad de datos etiquetados y/o segmentados. Este aspecto es especialmente relevante cuando se trata de imágenes histológicas (imágenes de la estructura microscópica de los tejidos), que pueden contener miles o decenas de miles de células, lo que hace que etiquetar esos datos sea una tarea muy exhaustiva y subjetiva incluso para patólogos expertos. La falta de profesionales que se dediquen a estas tareas unido a la laboriosidad de las mismas hace que la escasez de muestras de entrenamiento positivas sea una de las principales limitaciones con la que se encuentran los investigadores a la hora de usar algoritmos supervisados[2][3].

Contribuciones de las Redes Generativas Antagónicas. Las Redes Generativas Antagónicas pueden ser en ciertos casos simplemente una alternativa con algunas ventajas respecto a los métodos supervisados[4] pero en otros casos su contribución va mas allá y demuestran ser útiles también a la hora de ayudar a los métodos supervisados a superar sus propias limitaciones. Las Redes Generativas Antagónicas han sido usadas con éxito para mejorar sistemas supervisados que usan técnicas de análisis de imagen[3], por ejemplo:

- Aportando muestras sintéticas de manera general[5] o para equilibrar clases desbalanceadas[6], aumentando así el rendimiento del modelo.
- Reconstruyendo o reduciendo el ruido en imágenes médicas[7].
- Transfiriendo estilos, por ejemplo, normalizando el color de imágenes histológicas (tinciones diferentes) o trasladando imágenes de escáneres CT a MRI o viceversa, permitiendo así incluir en un proceso de entrenamiento imágenes de pacientes para los cuales se dispone de solo uno de los dos tipos de imagen[8].

La principal motivación para este trabajo es abordar la eventual problemática de la carencia de datos adecuadamente etiquetados, ofreciendo un proceso mediante el cual sea posible generar imágenes realistas en las que, a diferencia de los procesos habituales de aumento de datos, seamos capaces de integrar características que sean de nuestro interés en las muestras generadas. Los métodos habituales como rotaciones, volteos, recortes o distorsiones nos ofrecen una cantidad limitada de variación en torno a las imágenes originales y sobre las que tenemos poco control. Generando nuevas muestras mediante la aplicación de operaciones aritméticas se podría aumentar la riqueza de los datos aumentados permitiéndonos además conocer, o al menos acotar, la clase de la imagen generada.

1.2. Objetivos

Objetivos generales:

- Implementación y entrenamiento de una Red Generativa Antagónica (GAN) a partir de imágenes histopatológicas.

- Edición de las imágenes obtenidas mediante la aplicación de aritmética conceptual.
- Evaluación de los resultados

De manera más concreta los objetivos específicos de este proyecto, con algunas rectificaciones respecto al plan de trabajo inicial, son:

- Implementar una Red Generativa Antagónica.
- Entrenar la red a partir de un dataset de imágenes histopatológicas de pacientes con cáncer de mama. Inicialmente se escogió el dataset BreCAHad[9]. Este dataset contiene imágenes con anotaciones para seis características distintas, que pueden ser usadas para indicar la presencia o no de tres aspectos clave para la determinación del Índice Pronóstico de Nottingham (pleomorfismo nuclear, formación de túbulos y conteo mitótico)[10].

  En la fase inicial del proyecto, se observó que el número de muestras del dataset escogido era insuficiente (N=162) para entrenar una red de este tipo, incluso aplicando técnicas de aumento de datos. Las imágenes además fueron obtenidas con zoom elevado, con lo que parchearlas para obtener un número mayor de muestras de menor tamaño no era viable ya que las imágenes resultantes eran prácticamente a nivel de una decena de células, perdiendo así mucha de la información acerca de los patrones celulares que nos interesaban.

  Por tanto este primer objetivo, implementar y entrenar una GAN a partir de imágenes histopatológicas, se concretó de manera ligeramente diferente a lo planificado en el plan de trabajo. La mala elección del dataset era uno de los riesgos detectados en la planificación y por eso figuraba entre los soluciones de contingencia el uso de un dataset diferente. Se creó un nuevo dataset compuesto por parches obtenidos a partir de imágenes Whole Slide Images (WSI) con tinción hematoxilina-eosina (H&E) de pacientes con cáncer de mama. Las imágenes están acompañadas de anotaciones indicando las coordenadas de las zonas metastatizadas.

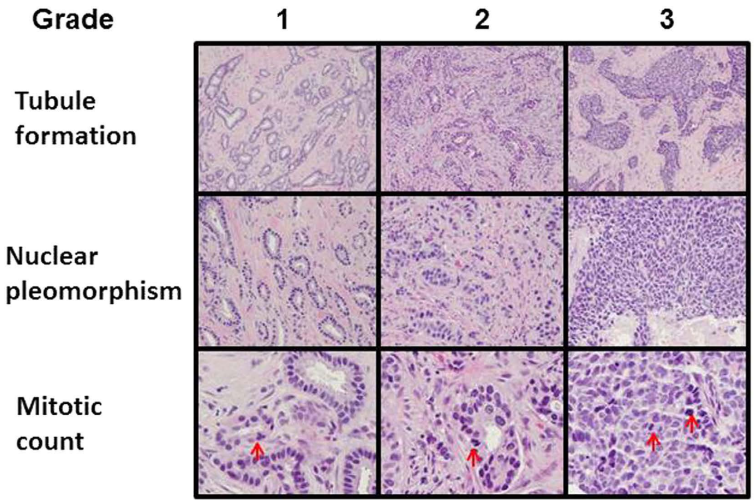


Figura 1.1: Carcinoma ductal invasivo de distintos grados de Nottingham[11]

- Generar nuevas muestras a partir de la red entrenada y aplicar sobre estas distintas operaciones aritméticas para editar las imágenes resultantes, por ejemplo, eliminando la presencia de túbulos de un tejido con ellos, añadiendo mitosis a un tejido en el que no aparezcan o lo hagan en menor medida o generando imágenes que se correspondan con fases intermedias de una patología a partir de imágenes de fases anteriores y posteriores.

  Con el nuevo dataset disponemos de un número mucho más elevado de imágenes lo que nos garantiza el correcto entrenamiento de la GAN, siendo sin embargo más desafiante la aplicación de las operaciones aritméticas sobre las imágenes al disponer de solo dos clases (normal, canceroso).

  Aunque inicialmente no estaba previsto que las anotaciones del dataset fuesen usadas en el entrenamiento sino con el propósito de ayudarnos a identificar visualmente características que se correspondiesen con los distintos estados de un tejido, en la última versión del programa, que hace uso de una DCGAN condicional, se incorporan las etiquetas al entrenamiento de la red de manera que podemos condicionar el modelo para que genere muestras normales o cancerosas.
- Analizar distintos métodos para la evaluación de redes generativas y evaluar las imágenes generadas mediante algoritmos de segmentación, clasificación o clusterización ya existentes (Inception, Resnet) comprobando como afecta a su desempeño la inclusión de las muestras sintéticas.

1.3. Enfoque y metodología

A partir de la definición de los objetivos y del hardware y tiempo disponibles se han desarrollado las siguientes estrategias relativas a tres aspectos clave del proyecto: una relativa a las características de los datos y a la obtención de estos, otra relativa al software y a la arquitectura de la red y una tercera con respecto del proceso de desarrollo.

En cuanto a los datos:

Las imágenes deben de estar correctamente etiquetadas y/o segmentadas. Las etiquetas no son necesarias para entrenar una Red Generativa Antagónica pero estas pueden tener una función didáctica que nos permita identificar los distintos estados de un tejido. Además aunque en su versión original las GANs no hacen uso de las etiquetas, estas pueden ser incorporadas al entrenamiento con otros tipos de Redes Generativas Antagónicas como las cDCGANs.

Es necesario que las imágenes tengan un tamaño lo suficientemente pequeño de modo que puedan ser procesadas por la red de manera efectiva. La mayoría de las imágenes, tanto histológicas como de escáneres (MRI, CT, PET...) están disponibles en forma de imágenes en alta resolución y etiquetadas solo a nivel de paciente.

En el caso concreto de las imágenes histológicas, estas suelen estar disponibles como Whole Slide Image (WSI), es decir una imagen de alta resolución obtenida directamente a partir de lo que se puede ver en el portaobjetos del microscopio (slide). Si se utilizan este tipo de imágenes se necesitan también anotaciones que indiquen mediante coordenadas que zonas de la imagen se corresponden con los tumores y un método que procese esas coordenadas junto con la imagen para extraer parches con la clase correctamente anotada.

El dataset también debe de permitir un cierto juego a la hora de realizar la parte relativa a la edición de imágenes. Esto significa que si el dataset contiene etiquetas para solo dos estados, por ejemplo, sano y enfermo, será más complicado obtener imágenes intermedias o distintas a estos dos estados que si disponemos de un dataset con muchas más características etiquetadas. Otra posibilidad es realizar la edición con atributos fácilmente distinguibles a nivel visual, sin tener en cuanta si la presencia o no de estos afecta al grado histológico. Esto nos permitiría demostrar que el concepto funciona y sería trivial aplicarlo más adelante a otras características que tengan relevancia médica.

Por último, el número de muestras debe de ser elevado. A mayor número de muestras mayores posibilidades de éxito en el entrenamiento de una Red Generativa Antagónica.

En cuanto al software:

El proyecto se ha llevado a cabo usando el *framework* para el desarrollo de redes neuronales Tensorflow. Como en el caso del *dataset*, otras alternativas fueron estudiadas, teniendo en cuenta aspectos como la popularidad de cada *framework*, la existencia de implementaciones de CNNs o GANs disponibles, su facilidad de uso o su rendimiento.

Relativo a estos dos últimos aspectos, se intentó realizar una prueba de concepto usando la implementación de Keras para Tensorflow, ya que ofrece un modelo multi-GPU capaz de repartir automáticamente la carga de trabajo durante el entrenamiento entre las distintas GPUs. Dado que contamos con dos GPUs se intentó implementar este modelo multi-GPU con una Red Generativa Antagónica sin éxito. Tampoco fue posible encontrar en la red ejemplos de GANs con este modelo multi-GPU (si de CNNs), posiblemente por el hecho de ser un modelo compuesto (generador y discriminador).

Descartada la posibilidad, al menos de inicio de usar las dos GPUs, se optó finalmente por una implementación con Tensorflow 1.x (sin usar la API de Keras) ya que, aún a cambio de una mayor complejidad a la hora de definir los modelos y una curva de aprendizaje más acentuada, este nos ofrece una mayor flexibilidad y control sobre los distintos aspectos de la red. Además la gran mayoría de los ejemplos, tutoriales e implementaciones de GANs disponibles online están escritos para la versión sin Keras.

Una vez elegido el software, los enfoques relativos a la **arquitectura de la red** pueden ser también múltiples:

- **Entrenando distintas GANs** por separado para cada una de las clases no se precisan etiquetas para disponer de dos clases de imágenes generadas separadas, pero perdemos la capacidad de generar muestras a partir de operaciones vectoriales entre imágenes de distintas clases. Al disponer de dos espacios latentes separados no podremos usar operaciones aritméticas que involucren a dos clases distintas. Si tuviésemos una clase ”normal” y una ”tumoral” no sería posible incorporar una característica de la clase tumoral a la normal por ejemplo. Estaríamos limitados a operaciones intra-clase. Las ventajas son que el entrenamiento es más rápido y se obtienen imágenes de mayor calidad. Además la evaluación es más sencilla puesto que podemos, por ejemplo, hacer pruebas de rendimiento usando clasificadores supervisados, ya que conocemos la clase real de las imágenes generadas.

- **Entrenando una sola GAN** para todas las clases dispondríamos de un espacio latente compartido en el que podríamos hacer aritmética entre las distintas clases pero no tendríamos certeza sobre la clase de la imagen resultante. Al no hacer uso de etiquetas de clase queda a nuestra discreción la correcta identificación de las imágenes con las que realizar las operaciones aritméticas y la correcta identificación/clasificación del resultado de estas. Las muestras obtenidas con este método son difíciles de evaluar objetivamente ya que no podemos usar clasificadores supervisados como en el caso anterior y por lo tanto, su uso posterior aumentando datos para tareas de clasificación supervisadas es limitado al carecer de un etiquetado objetivo.

- **Entrenando una sola GAN pero condicionando la generación**. Mediante la incorporación de etiquetas durante el entrenamiento podemos obtener una solución de compromiso en la que perdemos el carácter completamente no supervisado de los otros dos métodos pero que a cambio nos ofrece otras ventajas.

Si lo que pretendemos es aumentar el rendimiento de un método de clasificación supervisado, dispondremos ya de un *dataset* etiquetado, por lo que podemos estudiar el potencial de la GAN para, teniendo un número limitado de muestras de entrenamiento, mejorar los resultados de algoritmos de clasificación añadiendo al *dataset* muestras artificiales.

A diferencia del primer método, disponemos de un espacio latente compartido entre todas las clases y a diferencia del segundo método, la generación condicionada nos permite conocer la clase de las muestras generadas y en consecuencia podemos también valorar objetivamente la calidad de las muestras (al menos antes de aplicar la aritmética).

En cuanto al proceso de desarrollo:

El proceso de desarrollo se corresponde de manera más o menos estricta con la realización de las tareas descritas más abajo en este documento.

Se ha seguido un proceso iterativo y en el que las diferentes tareas que llevan a la consecución de un hito se realizaron de manera a veces paralela. Estas iteraciones terminan una vez llegada la fecha estipulada para cada uno de los hitos y las modificaciones de tareas correspondientes a hitos anteriores se han intentado mantener al mínimo. Esto nos ha garantizado no desviarnos demasiado de los objetivos, por ejemplo embarcándonos en modificaciones de las que no consiguiésemos salir con éxito o que nos hiciesen acumular retraso.

Todo el proceso de implementación, modificación del software y creación de distintas versiones está registrado en un repositorio privado en GitHub[12] que forma parte de los entregables.

1.4. Planificación

1.4.1. Tareas

Las tareas se corresponden con aquellas directamente relacionadas con la consecución de los objetivos, más las tareas relativas al análisis inicial del proyecto y a la elaboración de la memoria.

- Análisis inicial:
  1. Estudio del contexto y estado del arte.
  2. Planteamiento de posibles problemáticas a resolver.
  3. Búsqueda y selección de datos.
  4. Pruebas de concepto y elección de software.
- Implementación de la Red Generativa Antagónica:
  1. Desarrollar un módulo para la lectura y preprocesamiento de las imágenes.
  2. Implementar el modelo computacional de la GAN siguiendo la arquitectura descrita en Radford et al[4].
  3. Implementar el módulo de entrenamiento la red.
  4. Desarrollar un módulo que nos permita ver la salida de la red.
- Entrenamiento de la red:
  1. Entrenar la red con el dataset CelebA[13] y refinar los distintos hiperparámetros hasta obtener un resultado satisfactorio.
  2. Entrenar la red con el dataset de muestras histológicas. En este punto puede ser necesario modificar tanto el módulo de procesamiento de datos como la arquitectura de la red y sus parámetros para adecuarlo a nuestras imágenes.
- Desarrollar la funcionalidad necesaria para realizar aritmética de vectores conceptuales:

1. Aplicar distintas operaciones de ejemplo y mostrar los resultados usando el dataset CelebA.
2. Generar nuevas muestras utilizando operaciones con vectores conceptuales a partir de imágenes del dataset histológico.
3. Búsqueda de tensores en el espacio latente a partir de imágenes.

- Evaluar las imágenes generadas:
  1. Aplicar algún método de clasificación supervisada del que se disponga de una implementación.
  2. Realizar un informe comparando los resultados obtenidos con el método de clasificación dependiendo si se incluyen imágenes sintéticas o si no.
- Elaboración de la memoria del proyecto:
  1. Cierre de la memoria.
  2. Creación de una presentación.

1.4.2. Calendario

Id	Task Name	Duración	Comienzo	Fin
1	**Analisis inicial**	**15 días**	**06/02/19**	**20/02/19**
2	Contexto y analisis del estado del arte	3 días	06/02/19	08/02/19
3	Planteamiento de posibles problematicas a resolver	3 días	09/02/19	11/02/19
4	Búsqueda y selección de datos	6 días	12/02/19	17/02/19
5	Pruebas de concepto y elección de software	3 días	18/02/19	20/02/19
6	**PEC 0 - Definición de los contenidos**	**13 días**	**20/02/19**	**04/03/19**
7	**PEC 1 - Plan de trabajo**	**14 días**	**05/03/19**	**18/03/19**
8	**PEC 2 - Desarrollo del trabajo**	**37 días**	**19/03/19**	**24/04/19**
9	**Implementación de la Red Generativa Antagónica**	**20 días**	**19/03/19**	**07/04/19**
10	Desarrollar modulo para el preprocesamiento de las imágenes	4 días	19/03/19	22/03/19
11	Implementar el modelo computacional de la GAN	6 días	23/03/19	28/03/19
12	Implementar el modulo de entrenamiento la red	6 días	29/03/19	03/04/19
13	Desarrollar un modulo que nos permita ver la salida de la red	4 días	04/04/19	07/04/19
14	**Entrenamiento de la red**	**17 días**	**08/04/19**	**24/04/19**
15	Entrenar la red con el dataset CelebA	7 días	08/04/19	14/04/19
16	Entrenar la red con el dataset de muestras histológicas	10 días	15/04/19	24/04/19
17	**PEC 3 - Desarrollo del trabajo**	**26 días**	**25/04/19**	**20/05/19**
18	**Desarrollo de la funcionalidad para realizar aritmética de vectores conceptuales**	**13 días**	**25/04/19**	**07/05/19**
19	Aplicar distintas operaciones de ejemplo con CelebA	5 días	25/04/19	29/04/19
20	Generar nuevas muestras utilizando operaciones con vectores conceptuales a partir de imágenes del dataset histológico	8 días	30/04/19	07/05/19
21	**Evaluación de las imágenes generadas**	**13 días**	**08/05/19**	**20/05/19**
22	Aplicar algún método de clasificación supervisada	6 días	08/05/19	13/05/19
23	Realizar un informe comparando los resultados de aplicar el método de clasificación con y sin imágenes sintéticas	7 días	14/05/19	20/05/19
24	**PEC 4 - Cierre de la memoria**	**16 días**	**21/05/19**	**05/06/19**
25	**PEC 5a - Elaboración de la presentación**	**8 días**	**06/06/19**	**13/06/19**
26	**PEC5b - Defensa pública**	**8 días**	**17/06/19**	**24/06/19**

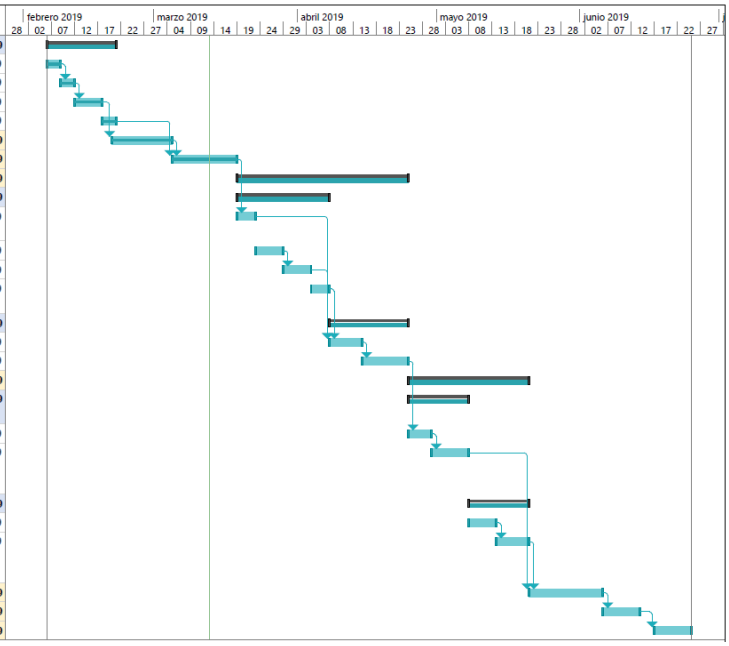


Figura 1.2: Diagrama de Gantt con la planificación del proyecto

1.4.3. Hitos

Los hitos marcan los estados intermedios del proyecto y permiten avanzar en sucesivas etapas de resultados prácticos. De producirse un retraso en la consecución de alguno de los hitos repercutiría negativamente en las siguientes tareas y podría comportar un retraso con respecto a los plazos marcados. Estos son los hitos que se establecieron durante la fase de planificación del proyecto.

Hito	Fecha límite
Disponer de un plan de trabajo	Antes del 18/03
Red correctamente entrenada, capaz de generar imágenes histopatológicas realistas	Antes del 24/04
Ser capaces de generar muestras mediante las operaciones con los vectores conceptuales	Antes del 07/05
Informe sobre la evaluación de las imágenes obtenidas	Antes del 20/05
Cierre de la memoria	Antes del 05/06
Elaboración de la presentación	Antes del 13/06
Defensa	Antes del 24/06

1.4.4. Sumario de productos obtenidos

Producto	Forma de entrega
Código fuente del programa	Zip con la versión final y repositorio de GitHub con el histórico de commits desde el inicio del proyecto y todas las releases intermedias.
Modelos generativos pre-entrenados (CelebA, histo-l0, histo-l1)	Checkpoints del entrenamiento con los distintos datasets
Modelos para la evaluación entrenados mediante Transfer Learning	Archivos .pb (incp_v3_Xk_lX_XXX.pb)
Ejemplos de los distintos tipos de muestras generadas	Imágenes, mosaicos y animaciones
Informes y muestras de la aplicación de operaciones en el espacio latente	Informes interactivos Jupyter
Scripts para el entrenamiento de los modelos usados en la evaluación de los resultados	Incluidos con el proyecto principal (/evaluation)
Informes sobre el rendimiento	Informes interactivos Jupyter
Scripts usados para el parcheo de WSI y creación del dataset.	Zip y repositorio de github
Memoria	Fuente .tex, pdf y documento word
Presentación	Documentos .ppt y .pdf

1.5. Entorno de trabajo. Hardware y software

En este apartado se describe de manera general el hardware y el sofware utilizado justificando en la medida de lo posible tales elecciones. Esta sección no pretende ser una guía de instalación, se incluirá un anexo en esta memoria con el resto de detalles.

Para la realización de este proyecto se ha utilizado un PC con la siguiente configuración de hardware:

- 2X AMD RX580 8GB GDDR5
- Intel Core i7 6700K (QuadCore, 4GHz, 8Mb)
- RAM 32GB
- HDD SSD 512GB

En cuanto al software:

- Ubuntu 18.04 como sistema operativo.
- Entorno virtual con Python 3.6.8 gestionado con conda y pip.
- Tensorflow 1.12 a través del paquete tensorflow-rocm para tarjetas AMD.
- Resto de paquetes utilizados para el desarrollo de la aplicación como numpy, matplotlib, PIL, etc...
- Visual Studio Code como entorno de desarrollo.
- Git para la gestión de versiones y GitHub para alojar el repositorio remoto.
- Jupyter notebooks para la creación de informes.

Las elecciones del sistema operativo y de Tensorflow como framework de desarrollo están determinadas por el tipo de hardware disponible (GPUs AMD). Conda se eligió sobre otros gestores de entornos ya que representa una opción más completa, y que además cumple funciones de gestión de paquetes y es compatible con pip. Para mostrar los resultados se prefirió el uso de informes interactivos ya que permiten una mayor flexibilidad a la hora de mostrar imágenes, animaciones o gráficas que los ficheros python tradicionales. En los anexos se puede encontrar una guía detallada sobre como instalar el entorno.

1.6. Descripción del resto de los capítulos de la memoria

- **Deep Learning. Análisis y generación de imágenes**

  En este capítulo describiremos en que consiste el Deep Learning y cual es su impacto tanto en el análisis de imágenes como en la generación, desde un punto de vista general y desde el de la imagen médica. Se analizaran también las arquitecturas usadas en este trabajo por orden de complejidad y veremos como cada modelo se construye sobre la base del anterior.

  - Análisis de imagen médica
  - Redes Neuronales Convolucionales (CNNs)
  - Redes Generativas Antagónicas (GANs)
  - Redes Generativas Antagónicas Convolucionales Profundas (DCGANs)
  - GANs Condicionales (cGANs y cDCGANs)

- **Diseño del programa**

  Este capítulo tratará sobre el proceso de diseño y desarrollo de la versión final del programa que se presenta en esta memoria. Se incluye un ejemplo de uso, desde la creación del dataset, uso del programa y motorización hasta la obtención de tensores con los que realizar operaciones aritméticas y la realización de las mismas.

  - Ejemplo de uso.
  - Vista y diseño general.
  - Procesado de datos.
  - Modelo.
  - Bucle principal de entrenamiento.
  - Salida de datos.
  - Scripts auxiliares.

- **Experimentos y resultados**

  En este apartado se analizan las distintas pruebas y experimentos realizados antes de obtener la versión final. También se comentan algunos consejos y lecciones aprendidas a la hora de implementar y entrenar con éxito una Red Generativa Antagónica.

- **Evaluación de los resultados finales**

  Capítulo en el que se abordan las características deseables en un buen método para evaluar GANs, la presentación de algunos de los métodos que se usan habitualmente y se dan detalles sobre la manera en que se realizó la evaluación en nuestro caso y los resultados de la misma.

- **Conclusiones**

  Conclusiones extraídas durante la realización del proyecto.

  - Aspectos a mejorar
  - Trabajos futuros

- **Anexos**

  - Guía de instalación del entorno
  - Reproducción del notebook de operaciones en el espacio latente
  - Reproducción del notebook de evaluación

2 | Caso de uso: cáncer de mama e imágenes histológicas

La idea con la que se inició el proyecto era entrenar la red sobre un *dataset* de imágenes de cáncer de mama con anotaciones para las distintas características que se usan para determinar el grado histológico, de manera que pudiésemos generar imágenes con distintas características y grado aplicando operaciones aritméticas sobre los vectores latentes correspondientes.

Desafortunadamente no fue posible encontrar un *dataset* en el que estas características estuviesen anotadas y además cumpliese el resto de requisitos necesarios para entrenar una Red Generativa Antagónica (número de muestras, tamaño de las imágenes, etc...). Finalmente se optó, como ya hemos indicado en la introducción, por usar un *dataset* creado a partir de imágenes WSI con anotaciones que indican únicamente que partes de la imagen completa son normales y cuales tumorales.

La siguiente figura es simplemente ilustrativa y la característica con la que se opera no implica un cambio en la gradación del cáncer, pero nos sirve para entender el concepto e imaginar que podría ser igualmente aplicado a otros datasets similares al que se pensó inicialmente.

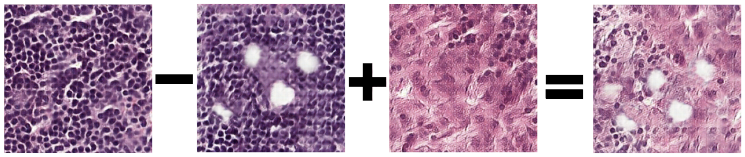


Figura 2.1: Muestra negativa sin lumen - muestra negativa con lumen + muestra positiva sin lumen (ejemplo editado manualmente)

Uno de los objetivos del proyecto es, por tanto, mostrar a nivel técnico que se pueden obtener buenos resultados editando imágenes mediante la aplicación de aritmética de vectores sobre muestras histológicas, aunque las operaciones se realicen con otras características distintas a las usadas para determinar el **grado de Nottingham**. A pesar de que pospongamos la aplicación sobre un *dataset* con las características necesarias para un trabajo futuro, describiremos en este capítulo que significa el **grado histológico** y porque es importante.

El grado de un tumor describe la apariencia bajo el microscopio del tejido y de las células tumorales. A partir de su aspecto los patólogos son capaces de obtener indicadores de como puede ser la evolución de ese tumor. Estos indicadores varían dependiendo del tipo de cáncer pero todos ellos determinan de alguna manera como se va a desarrollar en cuanto a la velocidad y la agresividad a la hora de expandirse.

Para algunos cánceres existen escalas específicas, como sucede en el caso del cáncer de mama con el grado de Nottingham o en el del cáncer de próstata con el de Gleason. Si no se indica que sistema se está usando, el grado suele referirse a la siguiente escala general:

- Grado 1 : Bien diferenciado
- Grado 2 : Bastante diferenciado
- Grado 3 : Poco diferenciado
- Grado 4 : No diferenciado

La gravedad del tumor aumenta con el grado. El término ”no diferenciado” aunque parece contraintuitivo en realidad nos indica que la morfología y distribución de las células es muy diferente de lo que sería si se tratasen de células sanas.

Los tumores bien diferenciados crecen y se expanden de manera más lenta. Cuanto más pobremente diferenciados, más anormales y mayor es el grado y la gravedad del tumor.

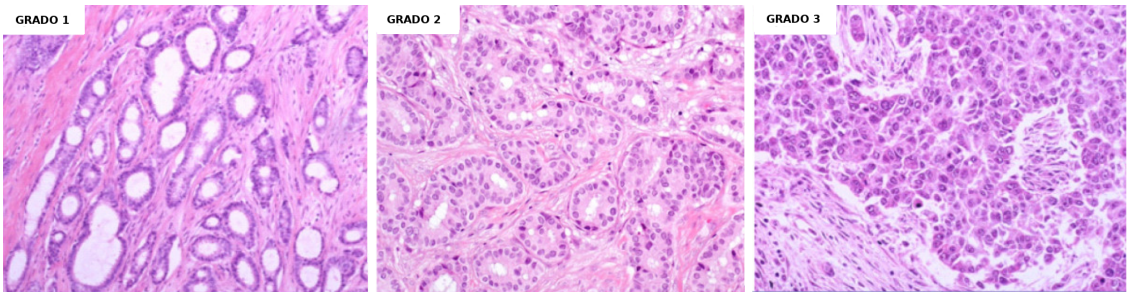


Figura 2.2: Distintos grados histólogicos según el sistema Ellston y Ellis

En el caso particular del cáncer de mama, como mencionábamos, disponemos de una escala específica. Esto es así debido a que algunos cánceres, por sus características específicas, resultan mejor diagnosticados aplicando una escala de gradación propia que tenga en cuenta esas particularidades.

La escala más úsada para el cáncer de mama es el llamado grado histológico de Nottingham, o de Elston y Ellis[10] que modificaron algunos parámetros de la escala más usada hasta ese momento (llamada de escala de Scarff-Bloom-Richardson) y se calcula de manera que se asignan distintas puntuaciones por la presencia o no de una serie de características visuales. Esas puntuaciones luego se suman, siendo más grave el cáncer cuanto mayor sea el resultado de la suma. Las características en las que se basa esta escala son:

- Presencia de formaciones tubulares o túbulos: Puntuación de 1 a 3 indicando la cantidad de tejido que está dispuesta en forma tubular correspondientes a conductos mamarios normales.
- Pleomorfismo nuclear: Puntuación de 1 a 3 que indica que cantidad de células presentan un tamaño anormal, cuantas tienen un tamaño excesivamente grande o irregular, o si se presenta mucha variación entre la morfología de unas células y otras.
- Conteo de mitosis: Puntuación de 1 a 3 que indica cuantas células están en mitosis. Una puntuación alta sería indicativa de que las células tumorales se están dividiendo y expandiendo a mayor velocidad. Se mide delimitando una zona con actividad mitótica y contando el número de mitosis por unidad de área.

Con estas tres puntuaciones se obtiene el grado histológico:

- Grado 1 : Bien diferenciado - Puntuación total entre 3 y 5
- Grado 2 : Algo diferenciado - Puntuación total entre 6 y 7
- Grado 3 : Muy poco diferenciado - Puntuación total entre 8 y 9

La importancia del grado del tumor a la hora de elegir un tratamiento adecuado depende de cada cáncer. En el caso del cáncer de mama es una parte fundamental del diagnóstico, siendo esta una de las razones por la cual es una de las escalas más conocidas.

Por último, no debe confundirse el grado del tumor con el estadio o etapa del cáncer. Este último describe cuanto se ha extendido desde su localización original, el tamaño del tumor o como afecta a otros órganos o tejidos adyacentes, etc. Algunos métodos de evaluación del estadio también tienen en cuenta, en mayor o menor proporción, el grado del tumor pero son conceptos distintos.

En el siguiente capítulo empezaremos a describir de que manera el Deep Learning toma parte en el análisis de imágenes como las que se usan para determinar el grado de Nottingham, desde los sistemas de clasificación supervisados a los sistemas generadores como las GANs.

3 | Deep Learning. Análisis y generación de imágenes

Con la publicación en 2006 de una serie de trabajos por parte de un grupo de investigadores del instituto CIFAR se abre una época de interés renovado en las redes ”feedforward” (o prealimentadas) profundas. Estos trabajos demostraban que era posible superar algunas de las antiguas limitaciones a la hora de entrenar Redes Neuronales Profundas e inauguraron un periodo en el que este tipo de redes comenzaron a acumular éxitos prácticos sobre todo en reconocimiento de voz y en visión artificial y reconocimiento de imágenes[14].

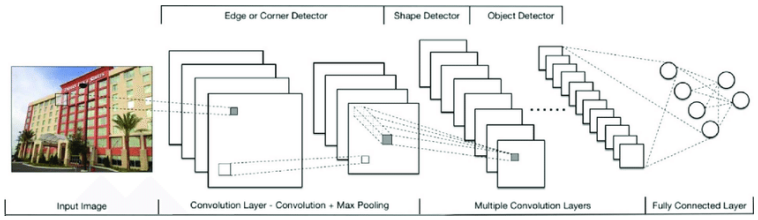


Figura 3.1: Las CNNs aprenden jerarquías de objetos cada vez más complejos según se profundiza en la red. *Fuente de la imagen:[15]*

Entre todos los tipos de redes ”feedforward” profundas se observó que un tipo en concreto poseía propiedades que hacían que fuesen mucho más estables y fáciles de entrenar y con las cuales se obtenían resultados mejores que con redes totalmente conectadas, que eran hasta ese momento la arquitectura más habitual de los sistemas de Deep Learning. Estas redes son las Redes Neuronales Convolucionales y propiciaron una década de innovación en el análisis digital de imágenes y visión por computador además de constituir la base sobre la que se fundamentan, en parte, modelos generadores como las DCGANs.

Las CNNs son capaces de explotar datos etiquetados, es decir, se entrenan con datos de los cuales conocemos la categoría para luego hacer inferencia sobre nuevos datos no etiquetados. Si queremos hacer uso de datos no etiquetados tenemos que recurrir a sistemas de aprendizaje no supervisado. Las GANs[16] nacen con el objetivo de acercar el rendimiento de los sistemas no supervisados al de los supervisados, y se basan en la idea de que si un sistema es capaz de crear algo que parezca real es porque internamente ha entendido los conceptos que hacen que algo sea realista, o dicho de otra manera construye una representación interna del concepto que intenta simular. Esta representación interna puede luego ser usada para otras tareas como clasificación o segmentación.

En este capítulo veremos que papel tienen los sistemas supervisados como las CNNs y los no supervisados como las GANs en el ámbito de la imagen médica y posteriormente se describirán desde el punto de vista técnico los fundamentos y el funcionamiento de cada uno de ellos.

3.1. Análisis de imagen médica

Los usos que los sistemas de Deep Learning encuentran en las disciplinas relacionadas con el análisis de imágenes medicas y biológicas no han dejado de crecer en la última década y no parece que vayan a hacerlo próximamente. En este estudio de la Radboud University[17] podemos comprobar la explosión en cuanto al número publicaciones que ha supuesto el Deep Learning y especialmente las Redes Convolucionales (representan más del 75 % de todos los *papers* de Deep Learning en este ámbito), con un crecimiento exponencial en cuanto al número de publicaciones entre 2012 y 2015.

La mayoría de tareas que se llevan a cabo son de segmentación, detección o clasificación. En la figura 3.2 se puede observar también la gran variedad de áreas de aplicación así como los distintos tipos de imagen a las que se aplican estas tareas.

Por supuesto el éxito del Deep Learning en imagen médica no se limita al número de publicaciones, sino que que además este tipo de sistemas han mostrado muy buenos resultados. En Kunal Nagpal et al[1] se describe un sistema de Deep Learning que mejora la precisión media de una cohorte de 29 patólogos a la hora de establecer la puntuación de Gleason para cáncer de próstata. Otros estudios muestran resultados de rendimiento que superan o igualan al de los humanos, por ejemplo en diagnóstico de cáncer de piel[18] o detección de retinopatía diabética[19].

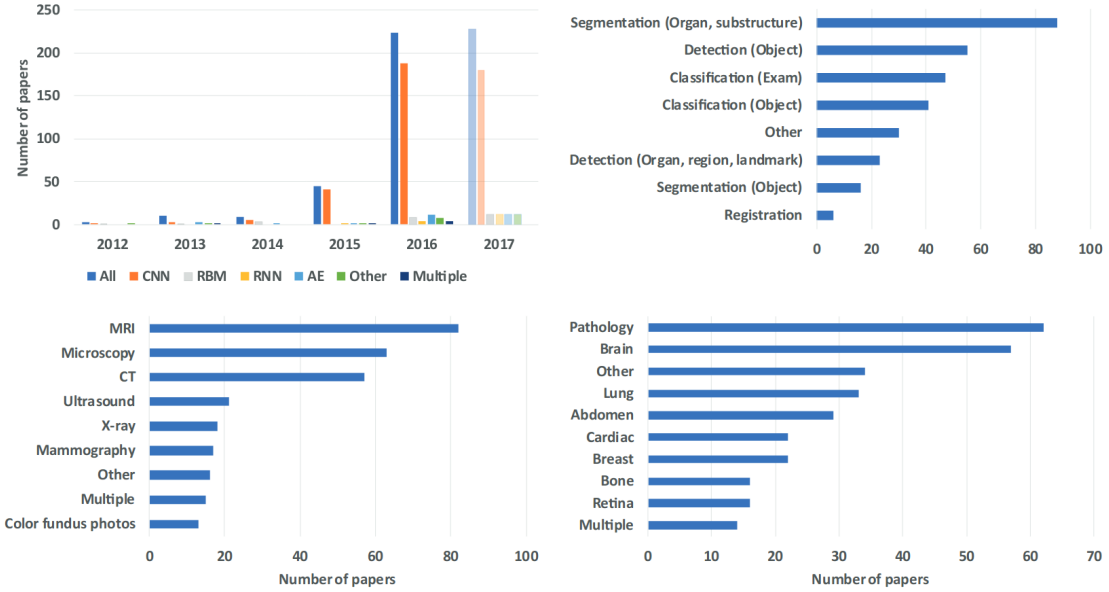


Figura 3.2: *Papers* en el ámbito de la imagen medica que hacen uso del Deep Learning (2012-2017)[17]

Los resultados no se limitan a tareas en las que mejorar la precisión de los patólogos sino también a otras en las que se complementa el trabajo de estos aportando más objetividad y eficiencia. Por ejemplo detectando mitosis en imágenes histológicas de cáncer de mama[20], que como vimos era una de las características usadas para determinar el grado histológico. Otro ejemplo puede ser este trabajo de G. Litjens et al.[21] donde describen un sistema que permite descartar entre el 30 % y el 40 % de las WSI que habitualmente analiza un patólogo sin intervención humana. Como podemos imaginar la automatización de estas tareas además de estandarizar los procedimientos y añadir objetividad a ellos, permite a los patólogos reducir enormemente la carga de trabajo a la que están sometidos.

Queda claro por tanto que el Deep Learning y las CNNs han abierto un mundo de posibilidades en el ámbito del análisis de imágenes médicas, sin embargo, hasta ahora nos hemos referido solo a métodos de **aprendizaje supervisado** para los que necesitamos grandes cantidades de datos etiquetados. Como ya se comentó en la introducción esto puede suponer un problema, ya que la escasez de imágenes etiquetadas en ámbitos médicos o biológicos se acentúa con respecto a ámbitos más generalistas, debido a que el etiquetado debe de ser realizado por expertos.

La importancia de poder contar con métodos **no supervisados** en el campo del análisis de imagen médica queda por tanto justificada y dentro de esta categoría las Redes Generativas Antagónicas han probado poseer características que las hacen atractivas en este ámbito. En la introducción se comentó que estas pueden ser usadas en tareas tales como eliminación de ruido, reconstrucción, segmentación, detección o clasificación[22] pero además se muestran muy prometedoras a la hora de ofrecer una solución a la escasez crónica de imágenes médicas etiquetadas.

Las Redes Generativas Antagónicas originales o "vanilla GANs"tal como fueron ideadas inicialmente por Ian Goodfellow[16], están compuestas por capas totalmente conectadas ("fully connected"). En 2015 Radford et al[4] demostraron que, una nueva arquitectura basada en la combinación de Redes Neuronales Convolucionales y Redes Generativas Antagónicas, además de ser mas fácil de entrenar que la original, poseía propiedades interesantes, como la posibilidad de obtener nuevas imágenes mediante la realización de operaciones aritméticas conceptuales. En esta característica se fundamenta gran parte de este trabajo, pero antes de entrar a detallar de que manera lo hace y como se ha desarrollado vamos a definir los distintos tipos de redes mencionadas hasta el momento.

3.2. Redes Neuronales Convolucionales (CNNs)

Contexto

La idea básica detrás de las CNN, como su nombre indica son las convoluciones. Convolución en matemáticas es un término que define la operación de integrar el producto de dos funciones al ser desplazadas una sobre la otra (y una de ellas volteada). Aunque en el ámbito de las redes neuronales el término más correcto sería el de convolución discreta, el término convolución se usa para describir como, mediante la combinación de una matriz de entrada y otra que se desplaza sobre ella, se obtiene, aplicando una serie de operaciones en cada una de las posiciones posibles, una nueva matriz.

En el ámbito de la inteligencia artificial, las convoluciones fueron usadas por primera vez en 1980 en el neocognitron y sus S-cell o células simples[23][24]. Ya en la actual era del Deep Learning las convoluciones volvieron sobre la mesa con el trabajo de Lecun et al[25], donde demostraba que era posible entrenar redes convolucionales mediante propagación hacia atrás de gradientes, obteniendo resultados excepcionales en reconocimiento de dígitos con su red Lenet. Aunque durante esa época (1988-2010) la mayor parte del progreso fue lento y apartado de los focos se obtuvieron algunos éxitos prácticos (por ejemplo un sistema de reconocimiento de dígitos para ingresos usado en muchos cajeros automáticos[26]) y se sentaron las bases para la explosión de aplicaciones del Deep Learning en la ultima década.

Convoluciones

Formalmente la convolución discreta se define mediante la siguiente fórmula:

$$[f \circledast g](i) \sum_{a} f(a)g(i-a)$$

La integral presente en la convolución continua se substituye por un sumatorio a lo largo de cada una de las posiciones discretas que puede ocupar el kernel o filtro al ser desplazado. En el caso de la primera fórmula ese desplazamiento es a lo largo de un vector unidimensional y en la fórmula que vemos a continuación, la más habitual al trabajar con CNNs, el desplazamiento es a lo largo de una entrada bidimensional, como una imagen.

$$feature\ map = entrada \otimes kernel = \sum_{y=0}^{columnas} (\sum_{x=0}^{filas} entrada\Big(x-a, y-b)kernel(x,y)\Big)$$

El *kernel* (habitualmente 3x3 o 5x5) se desplaza sobre la matriz de entrada, de manera que en cada una de las posiciones se calcula el producto escalar entre la matriz de entrada y el *kernel*, para obtener el resultado del elemento correspondiente en la nueva matriz (llamada *feature map*). Dependiendo de cuanto se desplace el *kernel* en cada paso (*stride*) y de como se desplaza cuando alcanza los bordes de la matriz de entrada obtendremos matrices de salida de diferentes tamaños. Esto se puede entender mas fácilmente de manera gráfica:

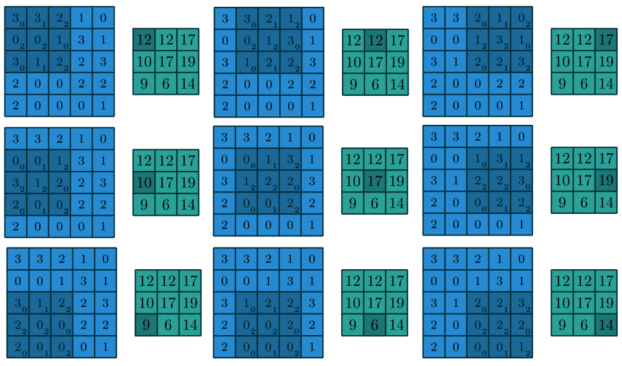


Figura 3.3: Cálculo de los valores de salida de una convolución discreta[27]

Los valores del *kernel* (en la esquina de cada cuadro en azul oscuro) se multiplican con los elementos de la matriz de entrada (números en el centro de cada cuadro). Estos valores se suman para obtener el valor de la matriz de salida (en verde).

Los *kernels* pueden ser vistos como filtros, que mediante el entrenamiento van adaptando sus pesos para detectar distintas características de la imagen. Esta idea es similar a los *kernels* estáticos que se aplican en edición de imagen para obtener un efecto, como detección de bordes o desenfoque gaussiano, solo que en el caso de las CNNs los filtros se inicializan al azar y es mediante el entrenamiento que estos obtienen su configuración óptima.

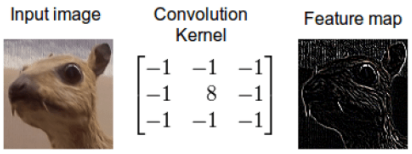


Figura 3.4: Ejemplo de filtro[28]

Relevancia

La idea clave detrás del buen comportamiento de las redes convolucionales al ser usadas en reconocimiento de imágenes o de voz es que las convoluciones preservan mejor la información topológica y la noción de orden de las entradas. Esto es importante porque al contrario que con otro tipo de datos, en el caso de las imágenes además del valor del pixel, es crucial saber que píxeles son vecinos o su posición relativa dentro de la imagen.

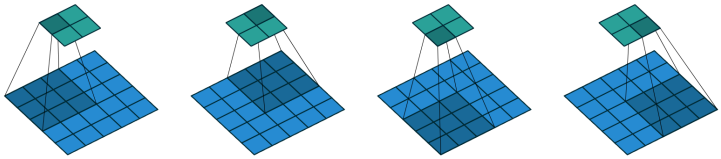


Figura 3.5: Convolución de paso 2 con un kernel 3x3 sobre una matriz 5x5[27]

El hecho de compartir los pesos, al deslizar el filtro a lo largo de la matriz de entrada, implica que estos detecten una misma característica en distintas partes de la imagen, o lo que es lo mismo, que sean invariantes al desplazamiento. Un filtro que detecte bordes horizontales o núcleos celulares lo hará independientemente de la posición relativa de estos dentro de la imagen. Cuando se trata de imágenes es común que aparezcan patrones repetidos con lo que tiene sentido que los pesos se compartan si se trata de reconocer el mismo patrón.

Otra característica importante es la reducción drástica de conexiones. El número de parámetros/pesos al usar capas completamente conectadas (”fully-connected”) se dispara al añadir más capas o aumentar el tamaño de imagen, aumentando demasiado el tiempo de entrenamiento. Sin embargo con capas convolucionales, debido a que normalmente el *kernel* es mucho más pequeño que la imagen de entrada, obtenemos una reducción muy importante de conexiones. Este hecho hizo posible que muchos casos de uso fuesen viables debido al aumento en el rendimiento.

Convoluciones Transpuestas

Las convoluciones transpuestas o convoluciones realizadas con paso fraccional (*fractional stride*), también mal llamadas deconvoluciones, son operaciones que pueden ser vistas como el proceso opuesto a la convolución discreta, de manera que al aplicarla obtenemos un espacio dimensional igual al de la matriz sobre la que se aplicó inicialmente la convolución y en el que además los pesos se conectan manteniendo los mismos patrones de conexión que en la convolución[27]

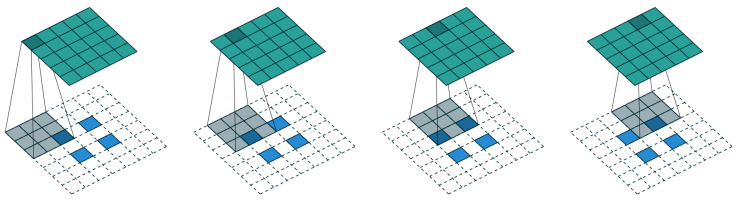


Figura 3.6: Una conv. transp. con kernel 3x3 sobre una matriz 5x5 y paso 2 es equivalente a una conv con paso unitario, kernel 3x3 y matriz de entrada 2x2 rellenada con ceros[27]

Es importante diferenciar la deconvolución de las convoluciones con paso fraccional o transpuestas. Con las primeras se obtienen de nuevo los valores existentes antes de aplicar la convolución, por lo que es estrictamente la función inversa, mientras que con las segundas se obtiene un tamaño de salida igual al de la imagen original, pero no se obtienen exactamente los mismos valores numéricos. Es decir se recupera la forma de la matriz y la manera en que se disponen sus conexiones con el *kernel* pero no los valores de estas.

La manera mas fácil de entender el concepto de convolución transpuesta es imaginarnos que equivale a realizar una convolución estándar de manera que el *kernel* se desplace a velocidad menor que 1. Esto se consigue insertando ceros entre los elementos de la matriz de entrada, como se puede observar en la figura 3.6

Las convoluciones transpuestas tienen gran importancia en las DCGANs ya que se usan para hacer *upscaling* en cada capa del generador de modo que en la última capa de este se obtengan imágenes con el tamaño adecuado para alimentar al discriminador.

3.3. Redes Generativas Antagónicas(GANs)

Las Redes Generativas Antagónicas o GANs son un tipo de red neuronal no supervisada propuesta por Ian J. Goodfellow en 2014[16] capaz de estimar la distribución subyacente de los datos con los que se entrena.

Mediante un entrenamiento entre dos agentes que compiten entre si la red aprende propiedades clave que caracterizan a una distribución a imitar. La red crea una representación interna que le permite generar nuevas muestras que parezcan convincentemente extraídas de la distribución original. El objetivo de la red no es por lo tanto minimizar el error sino generar muestras creíbles, ya que como veremos el error de uno de los agentes es siempre relativo a la capacidad del contrario. El error se puede interpretar como un indicador de la dirección de mejora para ambos jugadores pero no como una medida absoluta del rendimiento del sistema.

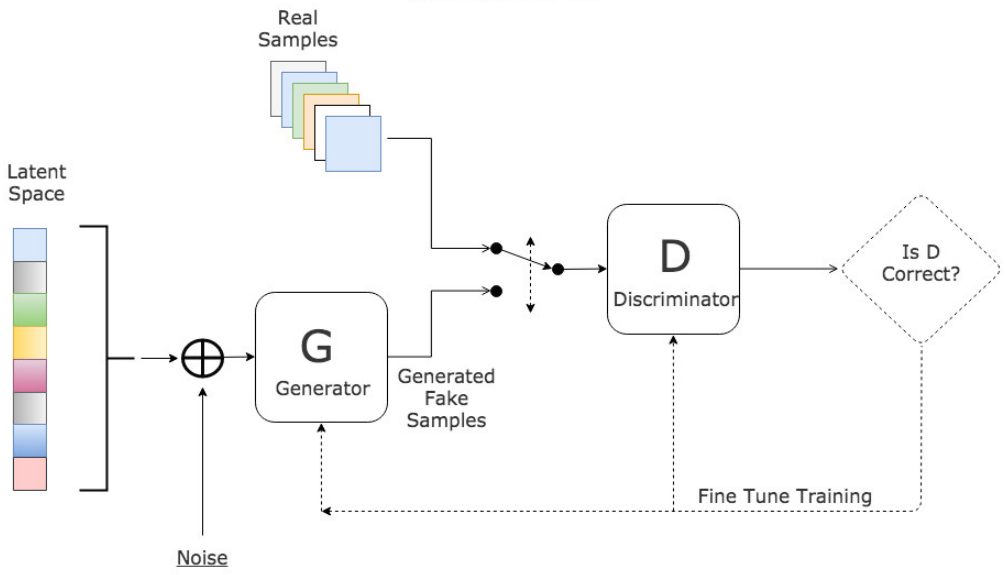


Figura 3.7: Red Generativa Antagónica. (GAN) *Fuente:[29]*

El entrenamiento se define como un proceso antagónico en el que se entrenan simultáneamente un modelo generativo que intenta representar la distribución de los datos y un modelo discriminativo que estima la probabilidad de que la muestra generada por el modelo adversario provenga de la distribución real y no del modelo generador. Este tipo de escenario se conoce como un juego minimax de dos jugadores en el que el objetivo del discriminador es maximizar la probabilidad de identificar correctamente si los datos provienen de la distribución real y el objetivo del generado es minimizar esa probabilidad aportando muestras cada vez más reales.

La función de error combinada de un sistema de este tipo se corresponde con la siguiente ecuación:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[1 - \log G(z)] \tag{3.1}$$

La única solución posible en este caso es que el generador G sea capaz de reproducir exactamente la distribución real de los datos, es decir que su error sea 0, y que la probabilidad de que el discriminador D sepa de donde procede la muestra sea igual a 1/2, o lo que es lo mismo, que no sea capaz de diferenciar las dos distribuciones. Llegar a esta solución óptima es, sin embargo, complicado en la práctica, ya que entre otras razones las GANs sufren una serie de problemas que dificultan su entrenamiento.

El primero de ellos es el denominado **"mode collapse"** en el cual la GAN "colapsa modos", en el sentido de que las distintas clases (modos) presentes en los datos reales no se ven reflejadas en las muestras falsas, generándose solo imágenes de uno de los modos. Se puede definir como la situación en la cual incluso puntos lejanos en el vector de ruido z son mapeados al mismo punto del espacio de muestras del generador. De manera intuitiva puede entenderse como una treta mediante la cual el generador es capaz de engañar continuamente al discriminador con el mismo tipo de imagen o con imágenes muy similares pero mediante la cual no aprende nada. Una de las razones por la que es posible llegar a esta situación es la manera en la cual está definido el error, ya que en este no se premia de ninguna manera la variedad muestral, ni se penaliza la ausencia de esta.

Otro problema es la **no-convergencia del modelo**. En la práctica la convergencia no está garantizada (además de poder ser extremadamente costosa) por lo que es difícil saber en que punto debemos detener el entrenamiento. La optimización a turnos del generador y discriminador provoca oscilaciones en el error. Cuando el discriminador toma ventaja el generador se queda atrás y viceversa. Estas oscilaciones afectan a la calidad de las muestras generadas y añaden incertidumbre al proceso de entrenamiento ya que no tenemos manera de conocer si después de cada oscilación el error de uno de los agentes volverá a bajar o si divergirá indefinidamente a partir de ese momento.

Por último un problema común también a otro tipo de redes profundas es el **desvanecimiento del gradiente**. Este problema se da cuando el gradiente con el que se entrena el generador disminúe hasta un punto en el cual el aprendizaje se detiene o es muy lento. Esto sucede cuando, con el entrenamiento avanzado, el discriminador se acerca a su punto óptimo y descarta con demasiada certeza todas las muestras que el generador le envía de modo que este último no recibe el suficiente feedback en forma de gradiente que le indique en que dirección tiene que actuar.

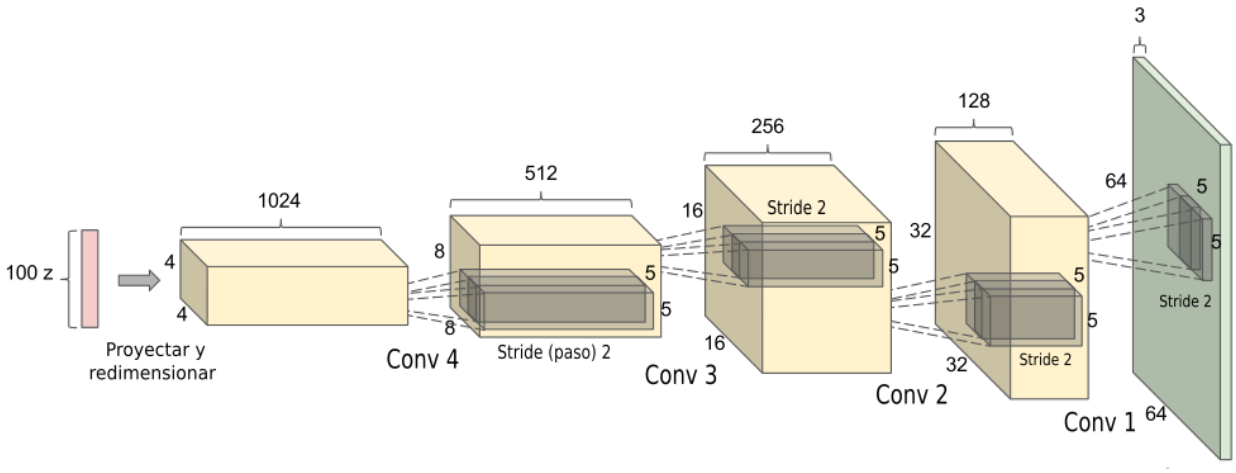


Figura 3.8: Vector de ruido y capas convolucionales transpuestas del generador de una DCGAN[4]

3.4. Redes Generativas Antagónicas Convolucionales Profundas (DCGANs)

Sobre la base de las Redes Generativas Antagónicas originales se idearon distintas versiones mejor adaptadas al entrenamiento con imágenes usando capas convolucionales en distintas combinaciones. La mayoría de esos métodos aún sufrían de problemas similares a los de las GANs originales durante el entrenamiento.

Radford et al[4] en 2015 proponen un nuevo tipo de red neuronal que también combina el entrenamiento antagónico con las redes convolucionales y que mejoraban a los sistemas existentes hasta la época (LAPGAN, PGGAN o BigGAN...). Esas mejoras consisten bien en una mayor estabilidad en el entrenamiento, en la capacidad de generar imágenes de calidad aún siendo más simples en su estructura o en distintas propiedades de su espacio latente, que comentaremos más adelante, y que la hacen muy interesante. En ese trabajo se identificaron una serie de puntos que hacían posible un sistema de ese tipo:

- Eliminar las capas ocultas totalmente conectadas.
- Reemplazar las capas de *pooling* (*max pooling* o *average pooling*) con capas convolucionales con *stride* para el discriminador o con convoluciones con *stride* fraccional para el generador
- Utilizar *BatchNorm* tanto en el generador como en el discriminador excepto en la capa de salida del generador y la de entrada del discriminador.
- Uso de la función de activación ReLU en todas las capas del generador excepto en la última donde se usa la tangente hiperbólica (TanH)
- Uso de la función de activación LeakyReLU en todas las capas del discriminador.

Como podemos observar en las dos figuras de este apartado todas las capas han sido reemplazadas por capas convolucionales. El volumen de cada capa indica el número de filtros y las cifras a los lados de cada capa indican el número de filas y columnas de la matriz de entrada. El *kernel* está representado en el interior de cada capa, en este caso, siempre de tamaño 5x5 y *stride* 2. Con un *stride* = 2 el tamaño de la entrada se duplica en cada capa, o se reduce a la mitad si hablamos del discriminador. Hay que recordar que en el caso del discriminador se trata de convoluciones transpuestas.

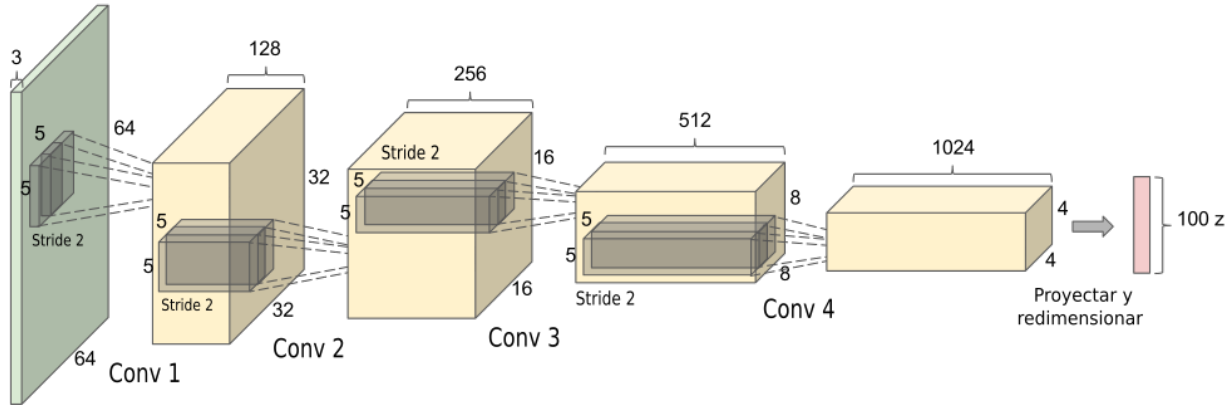


Figura 3.9: Parte discriminativa de una DCGAN y su salida[4]

Además de estos cambios también se proponen varias modificaciones de los valores de ciertos hiperparámetros: 0.2 para la pendiente de la LeakyReLU y 0.0002 y 0.5 para la tasa de aprendizaje y el momento del optimizador Adam respectivamente.

Las redes definidas con esta arquitectura poseen algunas propiedades interesantes:

- Permiten **recorrer el espacio latente**. Mediante el muestreo de puntos contiguos podemos avanzar en las diferentes direcciones del espacio latente y visualizar los cambios que se producen en función de las direcciones que tomemos.
- Es posible **visualizar las activaciones del discriminador**, de modo que sepamos que filtro o que capa detecta que característica u objeto. Permite reducir el componente ”caja negra” de la red de modo que entendamos mejor su funcionamiento.
- Mediante la realización de **operaciones aritméticas con vectores conceptuales** es posible operar con conceptos aprendidos por la red de una manera similar a como lo haríamos los humanos. Por ejemplo, mientras que en el caso de una suma en el espacio de salida se provoca una superposición de píxeles que resta sentido a la imagen, con una suma en el espacio latente los conceptos presentes en las imágenes se suman de manera que la imagen que se obtiene resulta coherente tanto visualmente como semanticamente.

3.5. GANs Condicionales (cGANs y cDCGANs)

Las Redes Generativas Antagónicas condicionales son una versión supervisada de las GANs originales donde un atributo o varios de la imagen son utilizados durante el entrenamiento y la generación de modo que proporcionen un nivel de control adicional sobre la imagen generada. Esta idea se puede aplicar tanto a vanilla GANs con capas totalmente conectadas como a GANs convolucionales como las DCGANs. En el caso de estas últimas, obtenemos una DCGAN condicional o cDCGAN que tiene una arquitectura prácticamente idéntica a la DCGAN con la excepción de las capas de entrada, como podemos ver en la figura 3.11.

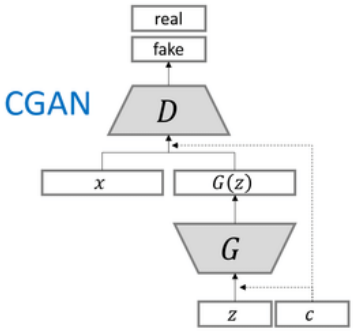


Figura 3.10: GAN condicional[30]

En la capa de entrada se concatena el vector de características (normalmente en forma de vector one hot encoded) a la imagen real, en el caso del discriminador, o al vector z en el caso del generador. Cuando se trata del discriminador la concatenación se realiza en la primera capa convolucional y no directamente en la entrada.

Entre los efectos, tanto positivos como negativos, que conlleva la introducción de etiquetas al entrenamiento podemos destacar:

- Pérdida del carácter no supervisado de las GANs.
- Mayor rápidez de convergencia y mayor calidad de imagen. Ayudando a la red a darse cuenta de que se trata de dos clases diferenciadas esta emplea menos tiempo para identificar las características clave de cada una sin pasar por una fase en la que intente generar muestras donde se mezclan atributos de las dos clases.
- Posibilidad de condicionar la generación. Mediante la introducción, en la entrada del generador, de un vector que indique las características deseadas, condicionaremos la generación de modo que la imagen que se obtenga contendrá esos atributos.
- Al recorrer el espacio latente se encontraran saltos más abruptos si intentamos pasar de una clase a otra que en el caso de una DCGAN incondicional. Cuanto mayor sea el número de características menos abruptos serán estos salto ya que seremos capaces de hacer una ”interpolación” del vector de etiquetas modificando características de manera secuencial. Por otro lado si solo se disponen de 2 o 3 clases los cambios serán mas bruscos ya esas 2 o 3 características controlaran muchos aspectos de la imagen.

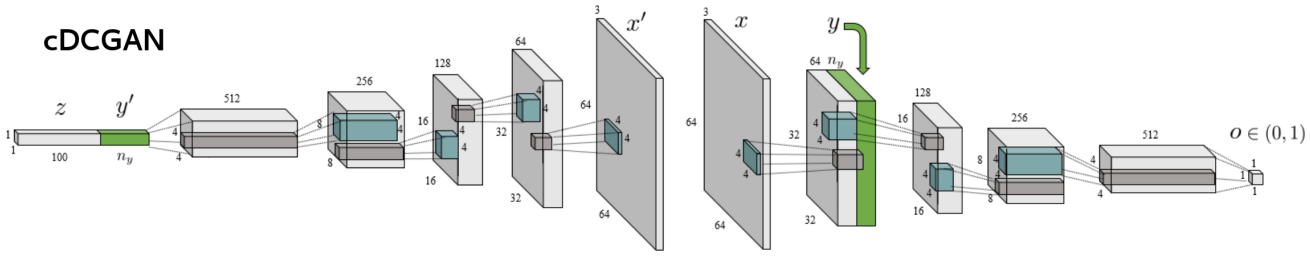


Figura 3.11: Esquema completo de un cDCGAN. En verde, los vectores de etiquetas se concatenan a las entradas tanto en el generador como en el discriminador[31]

4 | Diseño del programa

4.1. Ejemplo de uso

Obtención y preparación de los datos

Para crear el *dataset* descargamos 22 imágenes WSI en formato OpenSlide a través del portal del concurso de patología CAMELYON[32] que organiza la Radboud University Medical Center. Siete de las imágenes corresponden al reto de 2016 y quince imágenes al reto del año siguiente, CAMELYON17, que como novedad organizaba las *slides* a nivel de paciente. Todas las imágenes vienen acompañadas de un fichero de coordenadas indicando las zonas metastatizadas. Un criterio para seleccionar las slides puede ser escoger aquellas que contienen mayores áreas de tejido metastatizado para así poder obtener más muestras positivas. Esto se puede llevar a cabo eligiendo las WSIs cuyo fichero de coordenadas tenga un peso mayor y comprobando posteriormente de manera visual con una aplicación como ASAP[33] que efectivamente contienen áreas grandes de tejido canceroso.

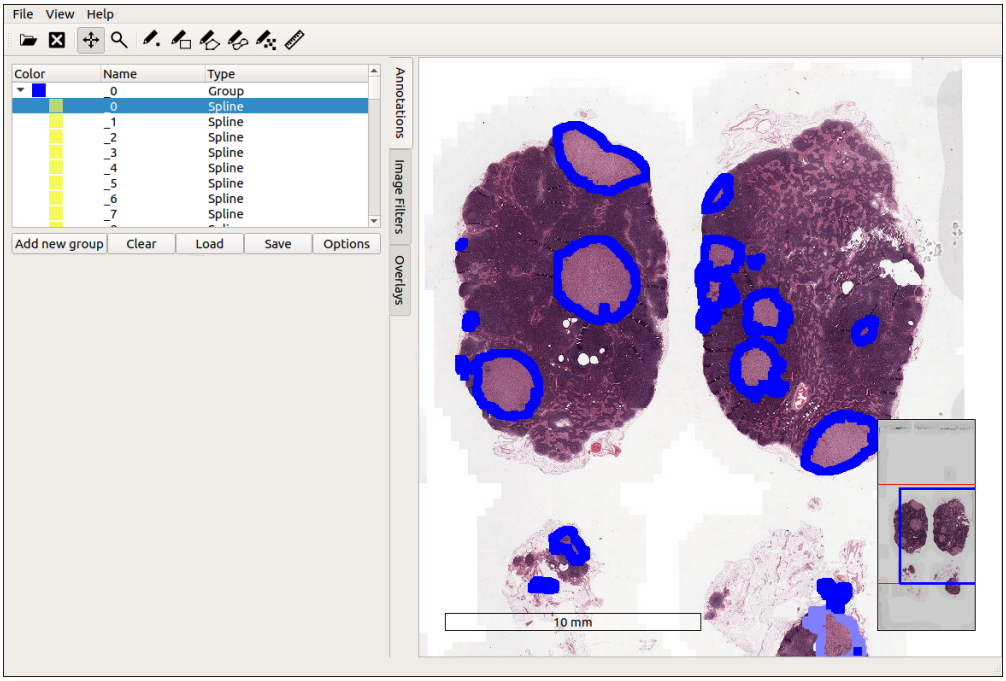


Figura 4.1: Visualización de una *slide* en ASAP. En azul, etiquetadas mediante un fichero de coordenadas, las zonas cancerosas

Mediante el programa WSI-analysis[34] y algunos *scripts* creados a medida(extract_main.py[35]) se obtienen parches de tamaño 256x256 en distintos niveles de zoom (0,1,3). El programa es capaz de leer los ficheros de coordenadas y determinar si un parche abarca una zona metastatizada y ordenarlos según sean parches positivos o negativos. Después del parcheo obtenemos las siguientes cifras:

nivel de zoom	parches negativos	parches positivos
3	6264	2373
1	116278	22122
0	337165	62062

Con el objetivo de obtener un *dataset* balanceado en cuanto al número de muestras de cada clase se crean mediante extracción aleatoria los siguientes *datasets*:

dataset	parches negativos	parches positivos
lvl 3	2300	2300
lvl 1	22000	22000
lvl 0	60000	60000

Como características reseñables del *dataset* podemos enumerar las siguientes:

- Solo 2 clases etiquetadas, lo que implica por un lado mayor facilidad para la GAN a la hora de intentar generar las muestras pero mayor dificultad a la hora de aplicar operaciones aritméticas al disponer de menos piezas con las que hacer combinaciones.
- Parches de 20 pacientes distintos, algunos de ellos con tinciones muy distintas, de distinto brillo, contraste o calidad de imagen. Aunque estas variedades no estén etiquetadas, en la práctica la GAN tiene que aprender a generar 20 clases de imágenes distintas lo que añade dificultad al proceso. Como ejemplo de esa dificultad, con algunos de los tipos que tienen menos representación se necesitaron muchas más epochs y horas de entrenamiento para obtener muestras de la misma calidad que con clases más numerosas.

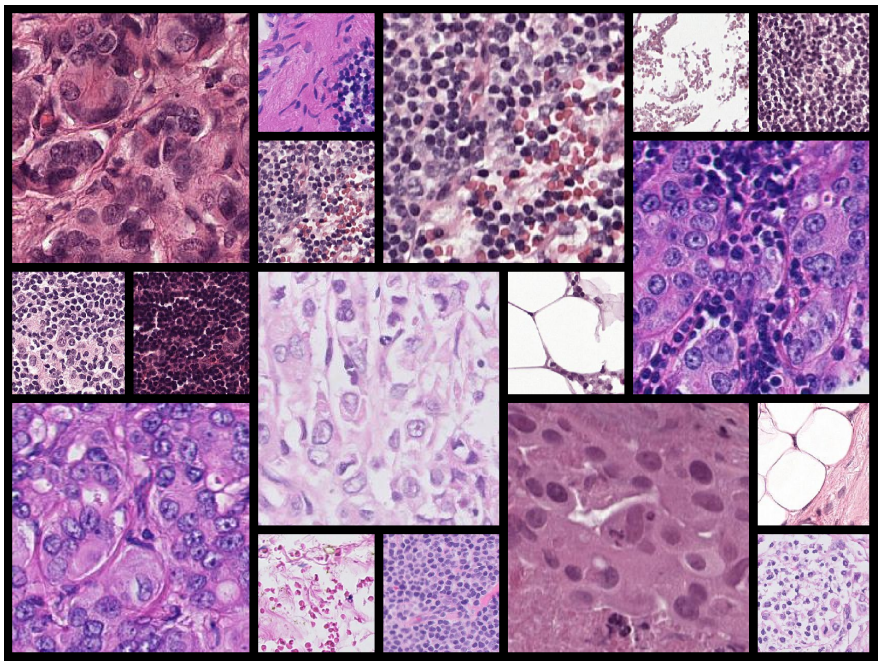


Figura 4.2: Parches obtenidos de diferentes pacientes con diferentes tinciones

- Imágenes de 256x256. Aunque la mayoría de implementaciones, incluida la original del *paper* de Radford et al. trabajan con tamaños de imagen de 64x64 o a lo sumo 128x128 encontramos algunos ejemplos que nos mostraban que podían obtenerse muestras de calidad con tamaños más grandes. Aún a costa de un tiempo de entrenamiento bastante superior decidimos hacerlo con ese tamaño de imagen porque era importante que se diferenciaran ciertas características visuales para la aplicación de la aritmética de vectores conceptuales, las cuales son difíciles de apreciar en imágenes mas pequeñas.

Configuración e inicio del entrenamiento

Para gestionar la configuración de los entrenamientos se añadió un fichero *json* que incluye distintos conjuntos de configuraciones y un campo ”configuracion-activa” que nos permite indicar cual de las configuraciones predefinidas queremos usar. De esta manera podemos crear distintas configuraciones según el dataset que usemos en lugar de modificar una y otra vez la misma. Un ejemplo de configuración puede verse en la figura 4.1.

Figura 4.3: Ejemplo de configuración del programa

Los mayoría de los parámetros que se ven en la configuración pueden cambiarse sin mucho riesgo, sobre todo, el número de *epochs*, el tamaño de las imágenes y los intervalos de guardado y del mostrado de información por consola. El tamaño del *batch* también se puede modificar en función de nuestro *hardware*, aunque no es recomendable usar *batches* demasiado pequeños. Se pueden hacer pruebas también tanto con el tamaño del vector de ruido y el número de filtros. Si nuestras imágenes son pequeñas puede ser suficiente con 100 o 128 para el vector de ruido y 64 o incluso 32 para los filtros (puede ser recomendable por rendimiento usar potencias de 2 en estos parámetros).

También es posible probar distintos valores para las tasas de aprendizaje y para el parámetro beta del optimizador Adam pero posiblemente con los valores por defecto se obtenga ya el mejor resultado. Además de los valores que encontramos en la configuración existen una multitud de parámetros extra que podemos modificar, aunque a no ser que tengamos algún motivo claro, lo más recomendable es utilizar la configuración por defecto para todos ellos:

- Tamaño del *kernel* en cada una de las capas
- Tamaño del *stride* o paso al cual se desplaza el *kernel* a lo largo de la matriz de entrada.
- Parámetro Epsilon y momento de las capas BatchNorm
- Parámetro Alpha o pendiente de LeakyReLU
- Parámetro que regula el suavizado de etiquetas (*label smoothing*)
- Cantidad de ruido que introducimos al discriminador junto con las imágenes y *decay rate* del mismo.
- Tipo de distribución de donde muestrear el ruido que introducimos a la red y parámetros de la misma
- ...

Una vez establecida la configuración podemos ejecutar el programa. Se han diseñado tres modos de funcionamiento, entrenamiento, generación de muestras y búsqueda en el espacio latente a partir de una imagen. Para empezar el entrenamiento se debe lanzar el script principal (histology_main.py) sin indicar ningún parámetro adicional.

Carga de checkpoints

Como se puede ver en la figura 4.1 podemos especificar mediante la propiedad ”save_model_every” cada cuantos pasos queremos guardar un *checkpoint* del modelo. Además de estos *checkpoints* se guardará también uno por cada hora de entrenamiento. Para que no nos quedemos pronto sin espacio en el disco se mantendrán solo los últimos 20 *checkpoints* en cada momento.

La carga del último *checkpoint* se realiza de manera automática al arrancar el programa, de manera que si queremos continuar un entrenamiento solo debemos relanzar el *script* principal sin parámetros. Si por alguna razón queremos cargar un *checkpoint* distinto del más actual, debemos modificar el fichero ”checkpoints” que se encuentra en la ruta indicada en la propiedad ”chkpts_path” de nuestro fichero de configuración. Esta opción es muy útil en caso de que dejemos el programa entrenando de manera desatendida, ya que en el caso de que la red haya comenzado a divergir, podemos cargar un *checkpoint* anterior que se corresponda con un punto en el que la convergencia era mayor o en el que observemos que las imágenes eran de mayor calidad.

Monitorización

Un aspecto muy importante a la hora de completar con éxito un entrenamiento con Redes Generativas Antagónicas es su monitorización. En la fase inicial del proyecto la monitorización se limitaba a imprimir las pérdidas del discriminador y generador y a la generación de n muestras cada cierto número pasos. Más adelante se añadió la posibilidad de monitorizar tanto el error como las imágenes generadas a través de Tensorboard.

Tensorboard nos proporciona la ventaja, a la hora de monitorizar el error, de mostrarnos los valores en una gráfica y de manera mucho más continua si así lo deseamos. Como habíamos comentado al hablar de problemas de las DCGAN y como también se puede observar en el gráfico, tanto el error del discriminador como el del generador oscilan mucho a la largo del entrenamiento, de ahí que para intentar averiguar en que punto la red deja de converger, observar los valores impresos en la consola no será igual de efectivo que ver un gráfico que muestre la evolución del error, así como sus puntos máximos y mínimos.

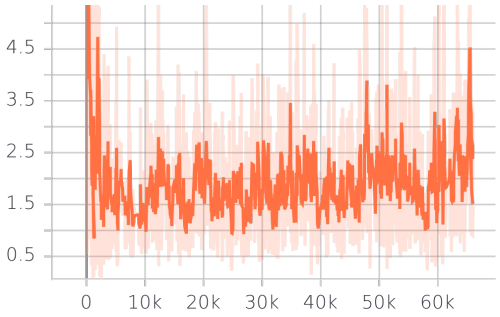


Figura 4.4: Pérdida del generador tras 66k pasos

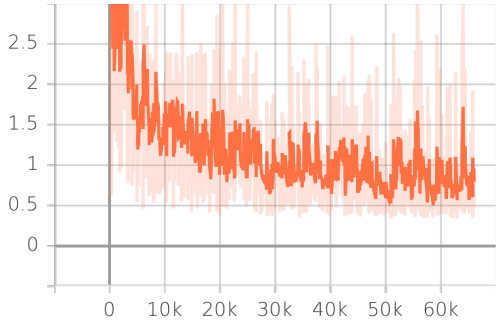


Figura 4.5: Discriminador - 84h de entrenamiento

Generación de muestras

Una de las pruebas que realizaremos para poder evaluar el rendimiento del modelo será entrenar un clasificador con imágenes artificiales. Para esto es necesario que generemos un dataset completo con presencia de ambas clases. Una vez tengamos el modelo entrenado podemos usarlo para generar muestras o datasets artificiales. Para ello debemos ejecutar el programa con los siguientes parámetros.

```
python histology_main.py --generate 30000 --labels n --out_prefix neg
python histology_main.py --generate 30000 --labels p --out_prefix pos
```

Con --generate indicamos el numero de muestras que queramos generar, si no indicamos nada más será un número aleatorio de muestras por cada clase hasta llegar al número total indicado, si indicamos --labels y n o p generaremos etiquetas negativas o positivas. El parámetro opcional --out_prefix sirve para añadir una cadena de texto al nombre de las muestras que se generen.

Edición de las imágenes aplicando operaciones aritméticas en el espacio latente:

- Mediante inversión de la clase

Si mantenemos fijo el tensor z que genera una imagen dada pero cambiamos el vector ”y” de etiquetas podemos obtener variaciones de la imagen con distintas características. En nuestro caso al solo haber dos clases, solo podemos alternar entre positivo y negativo pero si hubiésemos entrenado la cDCGAN con el dataset CelebA y sus etiquetas por ejemplo podríamos activar o desactivar características como añadir gafas, tipos de pelo, mujer, hombre, color de piel, etc... manteniendo el resto de rasgos de la persona.

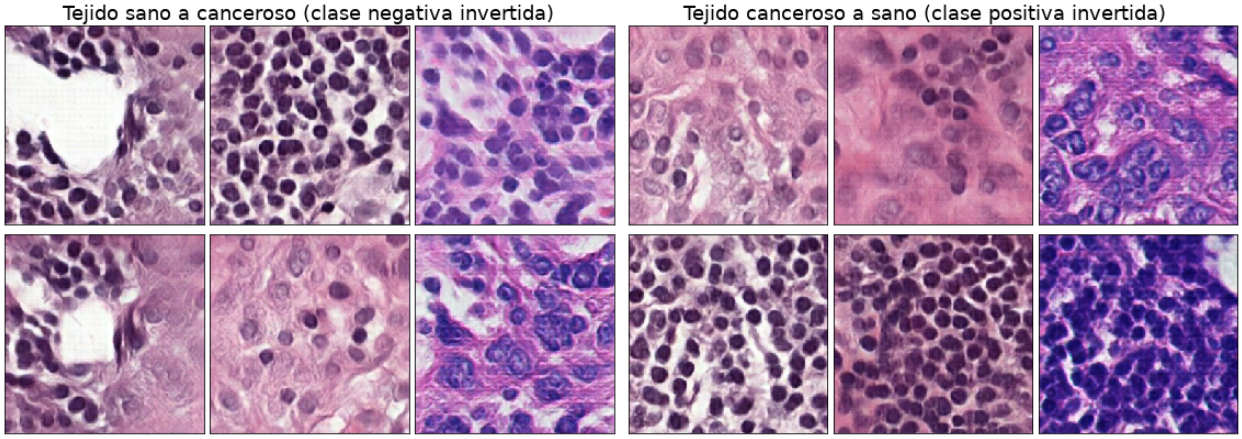


Figura 4.6: Si se superponen las imágenes de la fila superior e inferior podemos observar que comparten muchas características a excepción de las que determinan si son positivas o negativas. En los entregables se adjunta una animación donde esto se aprecia de manera mucho más clara

Como podemos ver en este ejemplo con imágenes histológicas la imagen conserva características entre las dos clases. En este caso el cambio es muy abrupto porque una sola etiqueta marca la diferencia entre muestras sanas y cancerosas, lo que provoca que un solo parámetro controle todas las características no comunes y que estas se activen o desactivan en bloque.

La manera de realizar este tipo operaciones se puede ver por completo en el notebook Jupyter correspondiente, pero básicamente se tratar recuperar un tensor que hayamos guardado previamente con la opción generate por ejemplo, ver de que clase es y construir un vector en codificación one hot con la clase inversa:

```
cancer_2 = np.loadtxt(FULL_OUTPUT_PATH + 'histo__0_4_pos.txt')
cancer_2 = np.expand_dims(img,0)
z_ = tf.placeholder(tf.float32, [1, Z_NOISE_DIM])
#Como es una imagen positiva, vamos a cambiarla a negativa.
labels = [0]
labels = np.eye(len(labels)+1)[labels]
samples = session.run(dcgan.sampler(z_,input_g_y,1), feed_dict={z_: cancer_2,input_g_y:labels})
```

- Mediante generación de interpolaciones

A la hora de generar interpolaciones se obtienen resultados bastante distintos dependiendo de si se utiliza la versión con DCGAN o la versión final con cDCGAN. Con la DCGAN, sin importar que dos muestras escojamos tendremos un espacio latente continuo que recorrer y la interpolación será siempre suave a poco que la red esté entrenada. En el caso de la cDCGAN, al disponer solo de dos clases en nuestro *dataset*, se produce un salto demasiado abrupto al cambiar el vector que codifica la clase, por lo tanto no se obtienen transiciones suaves entre cualquier muestra, solo si se escogen muestras que pertenezcan a la misma clase.

Al igual que sucede con la inversión de clases este problema se ve mitigado si disponemos de más clases, ya que podemos hacer ”interpolaciones binarias” del vector de clases. Por ejemplo con caras humanas, podríamos cambiar el pelo, las gafas, la sonrisa, etc (no a la vez) de manera que la transición fuese mas suave.

En el notebook que forma parte de los entregables (no en el anexo, ya que los pdf no admiten animaciones) se pueden ver dos ejemplos de cada uno de los casos que acabamos de comentar. La manera de generar estas interpolaciones es crear un vector que contenga en cada uno de sus posiciones el tensor ”a” multiplicado por un escalar que va decreciendo en cada paso y el tensor ”b” por uno que crece en una cantidad inversamente proporcional. De manera simple se puede describir como tener un poco menos del tensor ”a” y un poco más del tensor ”b” en cada paso:

```
z = np.empty(shape=(steps, Z_NOISE_DIM))
for i, alpha in enumerate(np.linspace(start=0.0, stop=1.0, num=steps)):
z[i] = (1-alpha) * tensor_a + alpha * tensor_b
z_ = tf.placeholder(tf.float32, [steps, Z_NOISE_DIM])
g_labels = tf.placeholder(tf.float32, [None, Y_DIM])
samples = sess.run(sampler(z_,g_labels, steps), feed_dict={z_: z, g_labels:labels})
```

- Mediante la suma, resta y media aritmética de los tensores

Para realizar más operaciones con los tensores, se han definido varias funciones, como la suma, la resta y la media aritmética. Se trata de operaciones muy básicas con matrices que podemos realizar con numpy.

```
def add_tensors(tensor_a, tensor_b):
return tensor_a + tensor_b

def substract_tensors(tensor_a, tensor_b):
return tensor_a - tensor_b

def average_tensor(tensors):
return np.mean(tensors, axis=0)
```

Para aplicarlas basta con obtener de nuevo algunos tensores y operarlos. En la siguiente porción de código podemos ver como hacemos tres operaciones idénticas entre una muestra con cáncer con lumen visible, otra muestra normal o con

cancer pero siempre sin lumen y una muestra normal también sin lumen. La cuarta operación (en la figura la columna de la derecha) se hace con las medias de los tres vectores en cada caso, la media de los tres primeros - la media de los tres siguientes + la media de los tres últimos. El resultado operando con las medias es sensiblemente mejor que los otros, donde también se aprecia que se añade lumen pero la imagen no tiene tanta coherencia.

```
resultado_1 = cancer_lumen_1 - all_no_lumen_1 + normal_no_lumen_1
resultado_2 = cancer_lumen_2 - all_no_lumen_1 + normal_no_lumen_2
resultado_3 = cancer_lumen_3 - all_no_lumen_1 + normal_no_lumen_3

cancer_lumen = latent_space.average_tensor([cancer_lumen_1,cancer_lumen_2,cancer_lumen_3])
all_no_lumen = latent_space.average_tensor([all_no_lumen_1,all_no_lumen_2,all_no_lumen_3])
normal_no_lumen = latent_space.average_tensor([normal_no_lumen_1,normal_no_lumen_2,normal_no_lumen_3])

resultado =  cancer_lumen  - cancer_no_lumen + normal_no_lumen
```

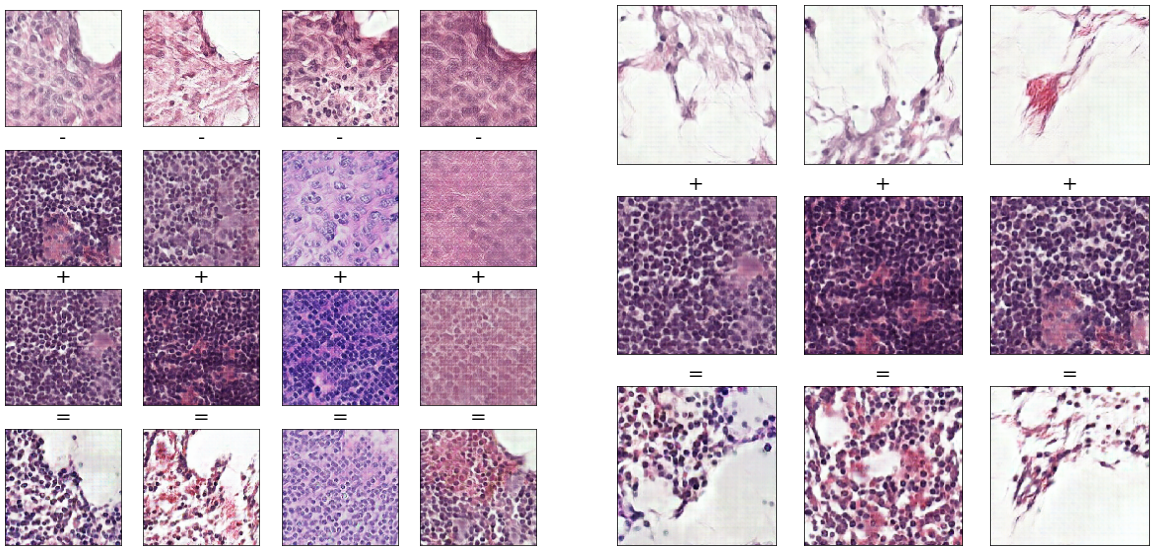
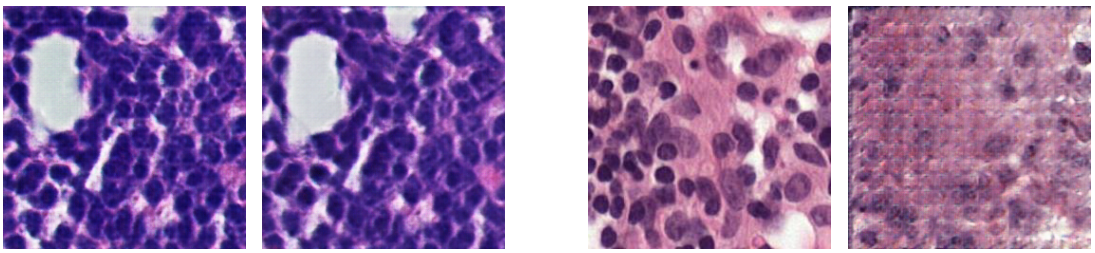


Figura 4.7: Ejemplos de aplicaciones de aritmética de vectores

Búsqueda inversa de tensores

Por último una pequeña demostración de búsqueda en el espacio latente. Se ha implementado mediante la minimización del error cuadrático medio entre las imágenes por descenso de gradiente y se puede utilizar para obtener tensores de imágenes generadas o mapear imágenes reales en el espacio latente para, una vez obtenido su tensor latente, poder cambiar alguno de sus atributos o hacer alguna de las operaciones aritméticas que hemos visto.



(a) Imagen generada **(b)** Imagen encontrada **(c)** Imagen real **(d)** Imagen encontrada

Figura 4.8: Mapeo de una imagen generada 4.8a y de una imagen real 4.8c en el espacio latente

En el caso de las imágenes generadas el mapeo inverso es casi perfecto. No lo es tanto cuando lo intentamos con una imagen real como la siguiente, aunque el resultado es mejorable, tanto si mejoramos el entrenamiento de la GAN como si incrementamos el tiempo de búsqueda en el proceso de mapeo.

4.2. Diseño general

El programa se ha diseñado siguiendo una estructura modular. Cada uno de los módulos cumple una función específica, y algunos de ellos como por ejemplo las dos implementaciones de la GAN son intercambiables.

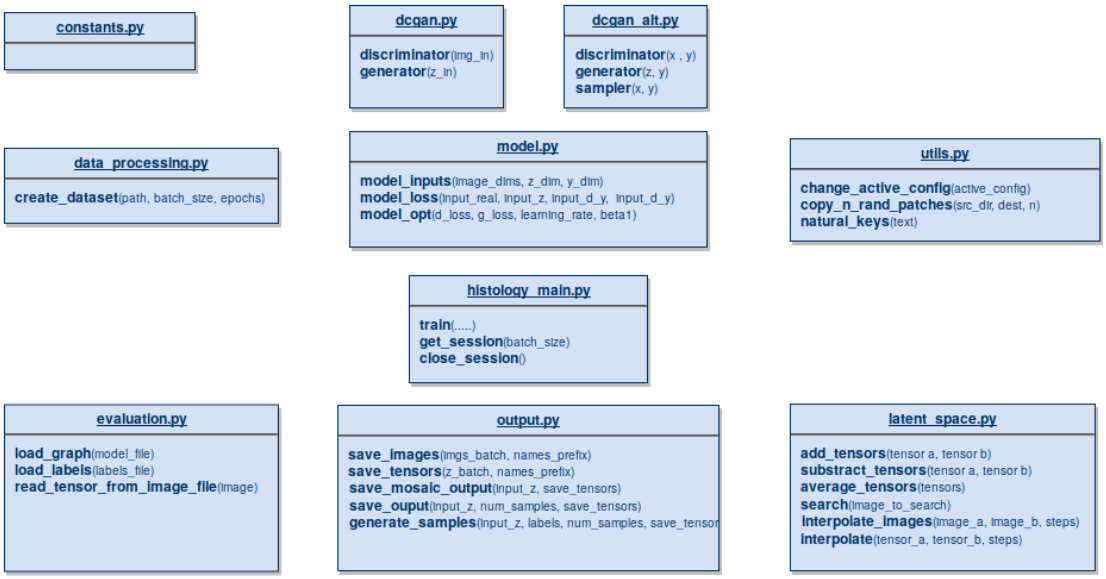


Figura 4.9: Ficheros o módulos que componen nuestro programa

En la figura 4.10 podemos ver como las distintas partes del grafo que crea Tensorflow se corresponden con los módulos de nuestra aplicación. La parte izquierda en verde se corresponde con el tratamiento de los datos y la creación de batches para alimentar al modelo, en el diagrama anterior se correspondería con el módulo data_processing.py. La parte azul comprende la arquitectura de la GAN y se encuentra implementado en los ficheros dcgan o dcgan_alt.py dependiendo de la implementación que usemos. El área en color rojo comprende las partes del modelo donde se calcula el error y los gradientes mediante los optimizadores. Se corresponde a grandes rasgos con el fichero model.py. Finalmente el grafo completo se une en el módulo principal, histology_main.py

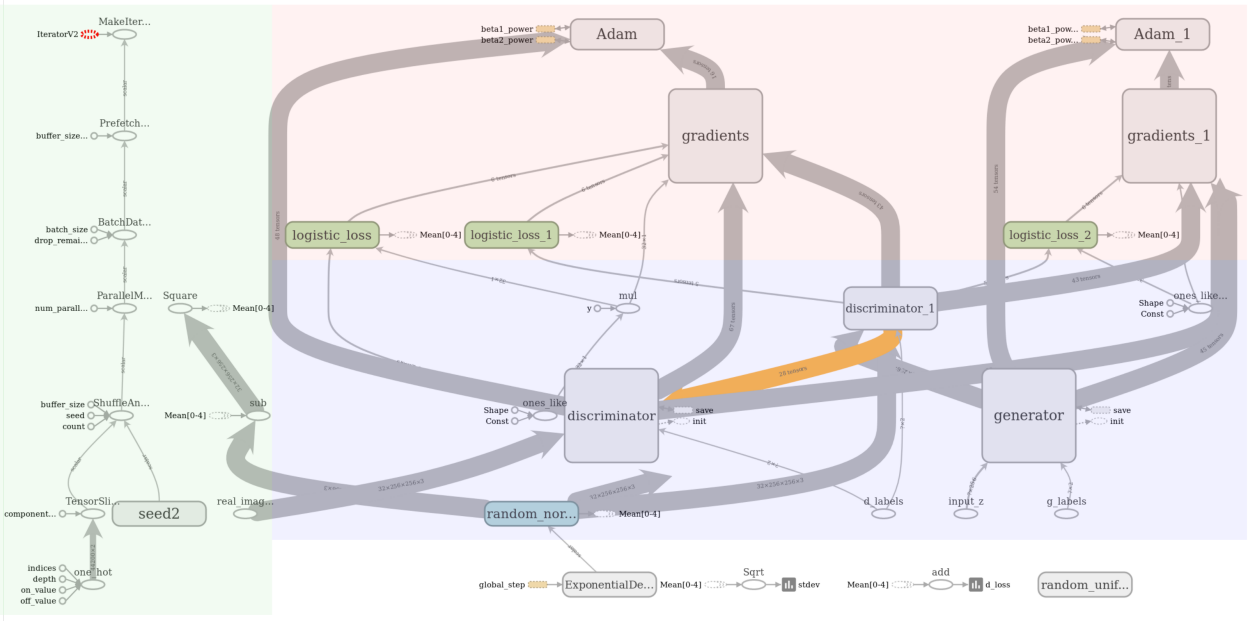
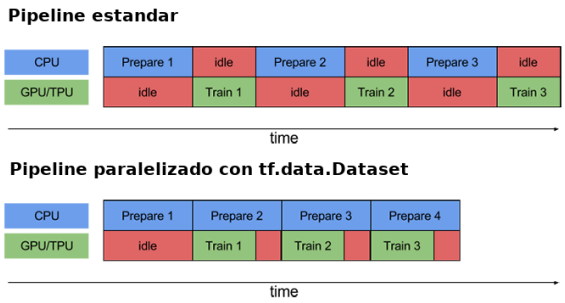


Figura 4.10: Captura del grafo que genera Tensorflow para ejecutar las operaciones programadas

Una decisión importante en cuanto al diseño es el uso o no de un enfoque orientado a objetos. Al principio del proyecto no se previó correctamente la amplitud que terminaría tomando el mismo y se inició la implementación usando un estilo más procedural fruto de implementar de manera iterativa funcionalidades cada vez más complejas a partir de ejemplos y tutoriales sencillos. Esta manera de programar, más adaptada al scripting o a programas mas simple nos obligó a pasar muchos argumentos en las llamadas de las funciones o a usar variables globales haciendo el código menos claro y más propenso a errores. De volver a implementarse se debería de optar por un enfoque orientado a objetos.

4.3. Procesado de datos

Para procesar los datos se eligió el modulo de Tensorflow tf.data.Dataset que nos permite implementar un pipeline de entrada más eficiente que la manera habitual y obvia de hacerlo, que sería usando placeholders y alimentándolos en cada iteración del entrenamiento.



Con la manera tradicional de procesar los datos y alimentar el modelo de manera síncrona la GPU tiene que esperar a que la CPU prepare el siguiente lote de datos antes de proseguir con el entrenamiento.

Con tf.data.Dataset podemos paralelizar el procesado y la carga de datos a distintos niveles:

- Permite transformar los datos mediante la funcion map, en la que indicamos la función a aplicar y el numero de threads. Estas funciones se aplican de forma paralela en distintos conjuntos de datos.

- Podemos indicar el numero de batches que queremos tener preparados de antemano con la función prefetch, de manera que en antes de comenzar cada paso del entrenamiento dispongamos ya de los lotes necesarios.

- Provee funciones combinadas para algunas de las operaciones típicas. Estas funciones combinadas ofrecen un mejor rendimiento ya que sus equivalentes simples se ven afectadas por el orden en el cual se ejecutan. Un ejemplo es indicar shuffle antes de repeat o después, en el primer caso se garantiza el orden a costa del rendimiento y el segundo ofrece un mejor rendimiento. La funcion shuffle_and_repeat combine las dos funciones de manera que no afecta al rendimiento. Otros ejemplos de funciones combinadas serian map_and_batch o padded_batch_and_drop_remainder.

4.4. Modelo

Las arquitecturas de la DCGAN y de la cDCGAN usadas en la última y penúltima versión son muy similares a la descrita en el paper de Radford et al con la salvedad del distinto número de filtros por capa. El generador está compuesto por 4 capas convolucionales, con kernels 5x5 y stride 2x2. A cada capa convolucional le sigue una capa de batch normalization y función de activación ReLU, salvo en la última capa donde no hay BatchNorm y la función de activación es la tangente hiperbólica.

El discrimidador se compone de 4 capas convolucionales transpuestas con kernels 5x5 y stride 2. Del mismo modo a cada capa convolucional transpuesta le sigue otra de BatchNorm salvo a la primera. La función de activación es LeakyRelu para todas las capas con pendiente 0.2. Se usan la mitad de filtros en cada capa con respecto a la versión original ya que no se observaba una mejoría en cuanto a la calidad de las muestras generadas pero si es algo que afecta al rendimiento a la hora de entrenar la red. De manera simplista con el doble de filtros cada iteración toma aproximadamente el doble de tiempo (aunque también la red aprende más).

Otras diferencias son:

- El tamaño del batch, 64 en vez de 128, debido a limitaciones del hardware. De hecho usando el número de filtros que se describe en el paper y no la mitad, solo podemos subir hasta 32 muestras por lote.
- Añadimos ruido junto con las entradas al discriminador, con el objetivo de debilitarlo frente al generador.
- Usamos label smoothing a la hora de calcular el error del discriminador. De esta manera obtenemos un efecto similar al punto anterior, evitamos el overfitting del discriminador haciendo que este menos seguro cuando clasifica una imagen como real o falsa.
- El generador se entrena dos veces por cada vez que lo hace el discriminador.

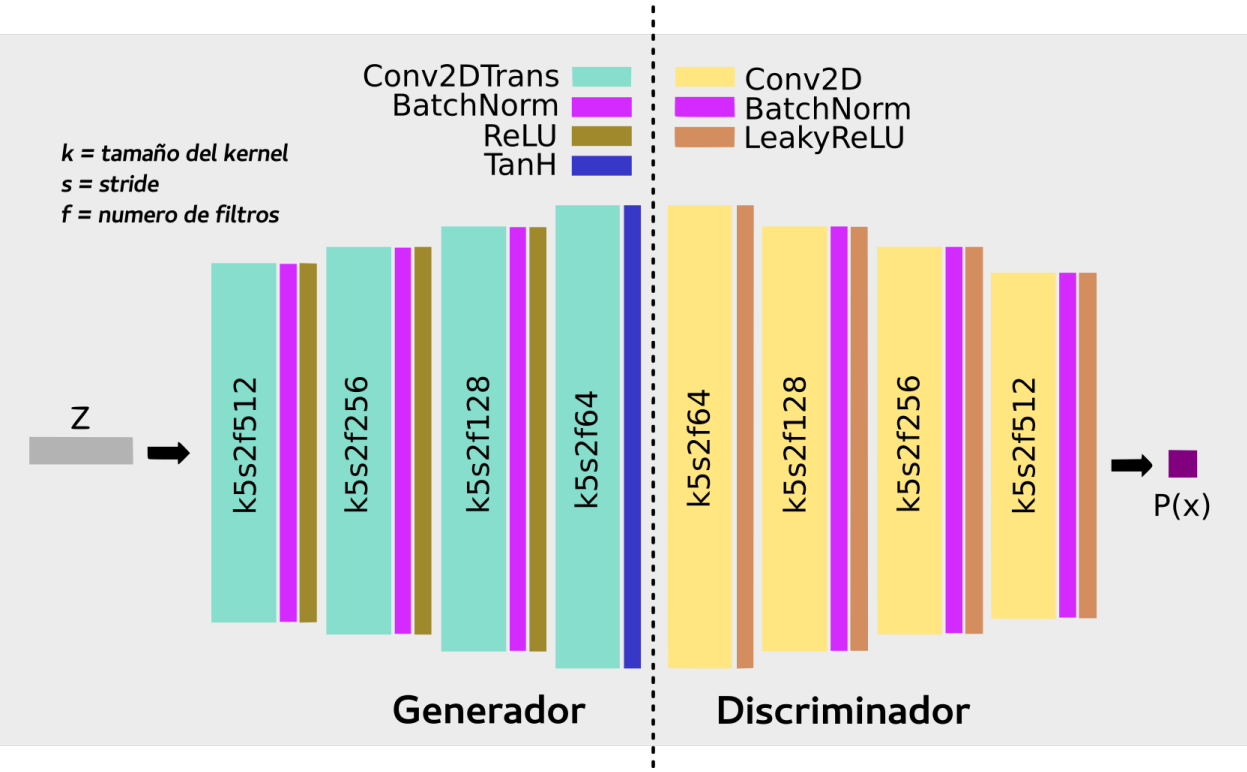


Figura 4.11: Esquema de la arquitectura DCGAN implementada.

Por otro lado, el optimizador es Adam con los parámetros por defecto descritos en Radford et al[4](learning rate de 0.0002 y beta1 de 0.5) y la pérdida del modelo queda implementada de la siguiente manera en tensorflow:

```
def model_loss(input_real, input_z, input_d_y, input_g_y, smooth_factor=0.1, decaying_noise=None):
fake_samples = generator(input_z,input_g_y)

d_model_real, d_logits_real = discriminator(input_real, input_d_y, reuse=False, decaying_noise=decaying_noise)
d_model_fake, d_logits_fake = discriminator(fake_samples, input_d_y, reuse=True, decaying_noise=decaying_noise)

d_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
logits=d_logits_real, labels=tf.ones_like(d_model_real) * (1 - smooth_factor)))
d_loss_fake = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
logits=d_logits_fake, labels=tf.zeros_like(d_model_fake)))
d_loss = d_loss_real + d_loss_fake

g_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
logits=d_logits_fake, labels=tf.ones_like(d_model_fake)))

return d_loss, g_loss
```

4.5. Salida de datos

En este módulo se incluyen diferentes funciones que a partir de los tensores generan imágenes llamando al generador y la vuelcan a disco. Se han definido tres funciones principales; save_output() se usa para guardar imágenes sueltas durante el entrenamiento, save_mosaic() para generar mosaicos de imágenes y generate_samples() que se usa con el parámetro –generate para generar imágenes en masa para la creación de datasets. Las tres funciones, con alguna pequeña variación, consisten básicamente en lo mismo:

```
#Obtenemos un tensor de ruido y se lo pasamos al sampler
example_z = sess.run(z_batch_tensor)
##el sampler no es mas que un generador con los
##pesos congelados, de manera que no se modifiquen al generar muestras
samples = sess.run(sampler(input_z, input_g_y),
feed_dict={input_z: example_z, input_g_y: lb_h})
....
....
imgs = [img[:, :, :] for img in samples]
....
#Posteriormente iteramos sobre las imagenes y las guardamos
output_image = Image.fromarray(transform_image(
image_from_array(image)).astype(np.uint8))
output_image.save(filepath)

#Transform image devuelve el array al rango normal de una
#imagen, ya que como comentamos, durante el entrenamiento
#la normalizamos entre -1,1 para obtener mejores resultados.
def transform_image(image):
# back from -1,1 to 0,1
image = image / 2 + 0.5
# from 0,1 to 0,255
image = image * 255
return image
```

4.6. Experimentos y resultados

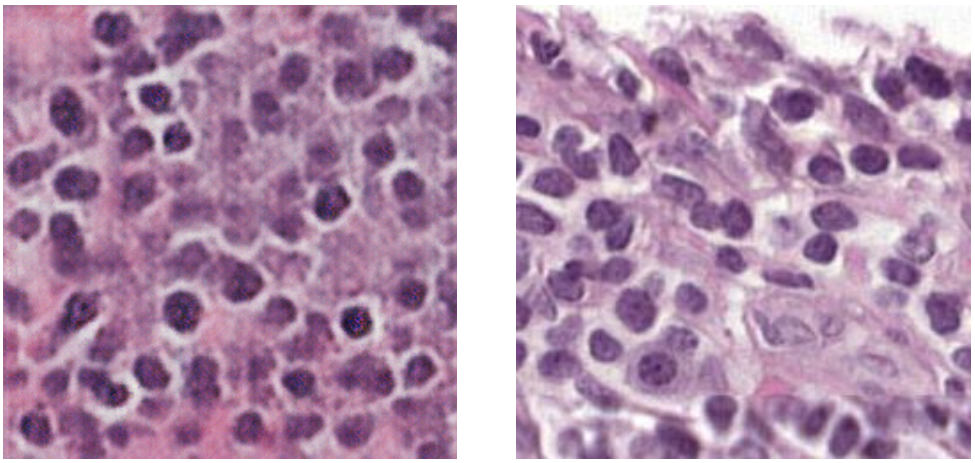
Una vez realizado todo el proceso de desarrollo es posible extraer algunas conclusiones. Estos son algunos de los fallos que se han detectado una vez llegados al final del proyecto:

No usar configuraciones estándar: La mayoría de ejemplos en la red usaban imágenes de 64x64 para entrenar la red y así fue como se implementó inicialmente. Sin embargo a la hora de intentar generar muestras de 128x128 se optó por añadir una capa más, de modo que con una convolución más con un stride = 2 (o conv. transp.) el tamaño resultante fuese 64x2, es decir 128. Aunque **añadir mas capas convolucionales** puede ser una estrategia válida, en este caso fue contraproducente ya que aumentó el tiempo de entrenamiento y lo hizo más inestable. El no usar una configuración estándar produjo una serie de dudas acerca de la validez del modelo y nos indujo a probar todo tipo de ajustes en hiperparámetros que introdujeron retraso en el proyecto.

Problemas al implementar batchnorm en el discriminador: Aunque finalmente tanto en la versión 0.1 como la 0.2 se eliminó debido a los malos resultados, el intento de implementar BatchNorm en el discriminador introdujo bastante retraso en el proyecto. Se ha podido constatar que la mayoría de implementaciones de DCGAN disponibles en la red no implementan correctamente estas capas, posiblemente debido a que inadvertidamente se comparten algunas de las variables entre las capas BatchNorm del discriminador y del generador a la hora de pasar las variables de cada uno de estos al optimizador. En la última versión con DCGAN y en la que se implementa la cDCGAN se volvió a introducir correctamente implementada esta vez con un enfoque completamente distinto en cuanto a la creación de ”scopes” de variables para las capas BatchNorm.

Uso de capas de dropout: Posiblemente como consecuencia de intentar obtener resultados satisfactorios con un dataset muy pequeño (BreCaHad) se añadieron capas de dropout en el discriminador para evitar overfitting a los pocos datos de los que disponíamos. Aunque en ese momento se mostraron efectivas y permitieron alargar más los entrenamiento antes de que divergiesen o entrasen en mode collapse, en la versión final con un modelo más correcto y datasets de mucho mayor tamaño se eliminaron por ser contraproducentes. Fueron eliminadas en la versión 0.3 a la vez que se corrigió el problema con las capas BatchNorm en el discriminador.

Uso de LeakyReLU también en el generador: Posiblemente el que menos consecuencias tiene de todos, pero aun así también fruto de no usar una configuración estándar. En el paper de referencia se indica el uso de ReLU en el generador y LeakyReLU en el discriminador. Este error se corrigió en la versión 0.2.



(a) Imagen generada de tamaño 256x256

(b) Ejemplo de parche real 256x256 obtenido de la imagen WSI

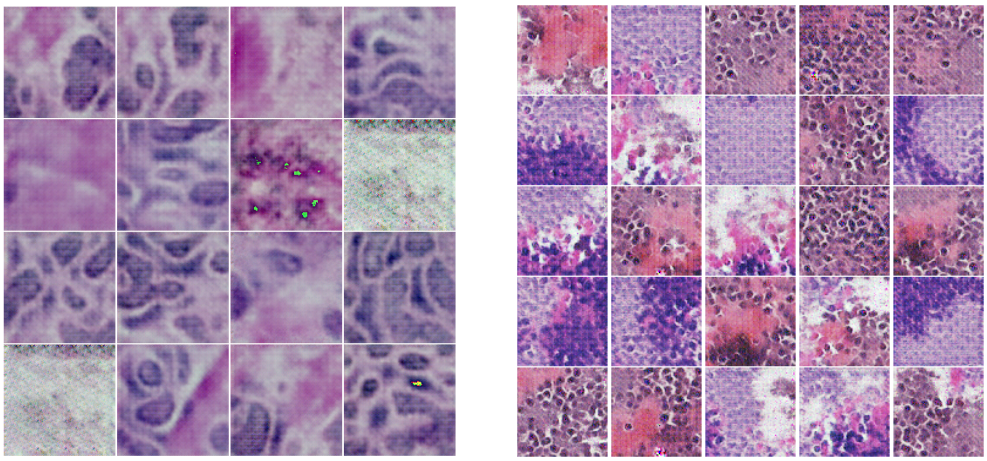
Figura 4.12: Ejemplo de imagen generada con la versión 0.2

Excesiva insistencia con el aumento de datos: Si tenemos un dataset muy pequeño no va a ser suficiente con encadenar un sinfín de operaciones de aumento de datos. Demasiado tiempo empleado en este apartado a la vista de los resultados que se obtuvieron con su aplicación.

Uso de distintas normalizaciones de la imagen de entrada. Inicialmente se realizó una normalización entre 0 y 1 de las imágenes pero se obtuvieron mucho mejores resultados a partir de la normalización de las imágenes de entrada entre -1 y 1 debido a que algunas funciones de activación de la DCGAN como LeakyReLU o la tangente hiperbólica usada en la salida del generador funcionan en ese rango. Corregido en la versión 0.2.

No utilizar la misma distribución para muestrear el ruido. Los vectores de ruido se sampleaban de una distribución normal(0,1) y una vez introducida la normalización entre [-1,1] de las imágenes de entrada se obtienen mejores resultados si mantenemos la coherencia y sampleamos también el ruido de una distribución uniforme en el rango -1,1. Corregido también en la versión 0.2.

Usar distintos ordenes de las capas en la versión inicial (¿dropout antes de batch normalization? ¿ReLU justo después de la convolución o después de la capa de dropout o de batch normalization?).



(a) Ejemplo de muestras de un entrenamiento fallido

(b) Algunas de las imágenes generadas con la version 0.2

Figura 4.13: Ejemplos de los resultados obtenidos con las versiones intermedias

Aparte de los errores mencionados se probaron otras estrategias que no se pueden considerar erróneas pero que una vez alcanzada la forma final de la arquitectura no produjeron mejores resultados o no se consideraron oportunas:

- Se probaron distintos tamaños de imágenes, tanto en la entrada como en la salida de la red. Partiendo de una premisa equivocada, se probaron y se añadieron distintos números de capas convolucionales con la idea de generar imágenes de mayor tamaño.
- Distintas cantidades de filtros en cada capa convolucional.
- Distintos valores para el kernel usado en las convoluciones y el paso al cual este se desplaza.
- Uso de tasas de aprendizaje diferentes para el generador y el discriminador para intentar mantener un equilibrio entre los dos.
- Ajuste en distintas combinaciones del resto de hiperparámetros, como el tamaño del lote, el tamaño del espacio latente, momentos y decay rates para el optimizador y las funciones de activación...
- Además de las normalizaciones que antes comentamos, se aplicaron distintas operaciones de aumento de datos rotando, volteando, o aplicando distintos niveles de contraste o brillo. Se probaron distintas combinaciones de estas.
- Dado que las muestras de las que disponíamos eran de muchos pacientes con tinciones muy distintas, se realizó una normalización de la tinción mediante un programa en python[36] que aunque ofreció buenos resultados visualmente y ayudo a mejorar los resultados de la generación no fueron suficientemente satisfactorios y no se siguieron usando. Posiblemente requiera de un ajuste más fino de los parámetros.

Por último y para completar la sección, puesto que ya se han comentado en detalle en el capítulo que describe la arquitectura final de la red, estas son las modificaciones que pudimos constatar como efectivas en nuestro caso y que se conservaron en la versión final:

- Añadir ruido a las entradas del discriminador.
- Entrenar varias veces el generador en cada iteración.
- Usar suavizado de etiquetas (label smoothing).
- Usar batch normalization en todas las capas, menos en la primera del discriminador y la última del generador.

5 | Evaluación de los resultados finales

La evaluación de las GANs presenta dificultades adicionales a las de otros modelos usados en Deep Learning. El primer escollo es que no es posible utilizar el error como medida absoluta de la bondad del generador o del discriminador ya que la perdida de uno de los agentes depende de la capacidad del contrario. Si los dos están poco entrenados tendremos un error total bajo, de la misma manera que si los dos están igual de bien entrenados. El error nos indica el rendimiento relativo de un agente con respecto a su adversario. Si se cuenta con experiencia entrenando GANs o se ha pasado tiempo entrenando en un proyecto concreto, se pueden detectar ciertas tendencias o patrones en la gráfica del error, pero no existe una regla general y la apariencia de la gráfica dependerá en gran medida de cada caso (arquitectura de la red, dataset usado, etc).

La manera más habitual de medir el rendimiento de las redes antagónicas es de manera indirecta, usando el modelo para tareas subrogadas como clasificación o reducción de ruido[37] y evaluando los resultados obtenidos al realizar estas tareas. Algunos ejemplos de este tipo de evaluación podrían ser los siguientes:

Utilidad en Data Augmentation : La utilidad de las muestras generadas para mejorar el rendimiento de clasificadores supervisados puede ser usada como medida de la calidad de estas muestras. Se podría entrenar un clasificador con un dataset combinado de muestras reales y generadas y compararlo con el mismo clasificador entrenado con solo muestras reales. Esto nos puede dar una idea tanto de la calidad como de la diversidad de las muestras que es capaz de generar nuestro modelo.

En la misma linea, se podría entrenar un clasificador con imágenes generadas en distintas fases, añadiendo en cada una de ellas un número de muestras mayor. La idea detrás de esto es que si al añadir más muestras mejora el rendimiento es que nuestro modelo generador genera diversidad. Por el contrario, si el hecho de añadir imágenes no mejora la precisión podemos concluir que estamos añadiendo datos redundantes y que la diversidad de nuestras imágenes es baja[38].

Este tipo de métodos de evaluación son por supuesto muy dependientes del clasificador que usemos, y para que sean útiles a la hora de comparar GANs será necesario establecer referencias, e indicar claramente con que clasificador y bajo que condiciones se ejecuta la prueba.

Rendimiento al clasificar. Otra medida indirecta para evaluar una Red Generativa Antagónica consiste en utilizar los "feature extractors"de una Red Generativa Antagónica entrenada para clasificar datasets etiquetados y evaluar el rendimiento obtenido. En Radford et al por ejemplo utilizaron las capas convolucionales del discriminador después de ser entrenado con el dataset ImageNet en combinación con un modelo SVM para clasificar imagenes de CIFAR-10

Inception Score. Es posiblemente la medida más popular. Se trata de clasificar las muestras generadas con un clasificador Inception V3 pre-entrenado con el dataset Imagenet. Si el clasificador clasifica nuestras muestras con certeza en las distintas categorías, la puntuación será mayor, indicando que nuestro modelo generador es capaz de crear muestras que pertenecen claramente a una clase.

Como hemos visto hay muchas tareas que pueden ser realizadas por una Red Generativa Antagónica que nos pueden servir como referencia. Una buena medida de la robustez de un método que evalúa el rendimiento de una GAN es que que cumpla todas o algunas de las siguientes características[37]:

1. El método prima a los modelos que generen muestras más diversas. Por lo tanto es capaz detectar el ”mode collapse” o el ”mode drop”.

2. Favorece modelos en los que el espacio latente está mapeado de forma coherente y tiene continuidad espacial de modo que no haya transiciones abruptas a la hora de generar imágenes correspondientes a puntos cercanos en el espacio latente.

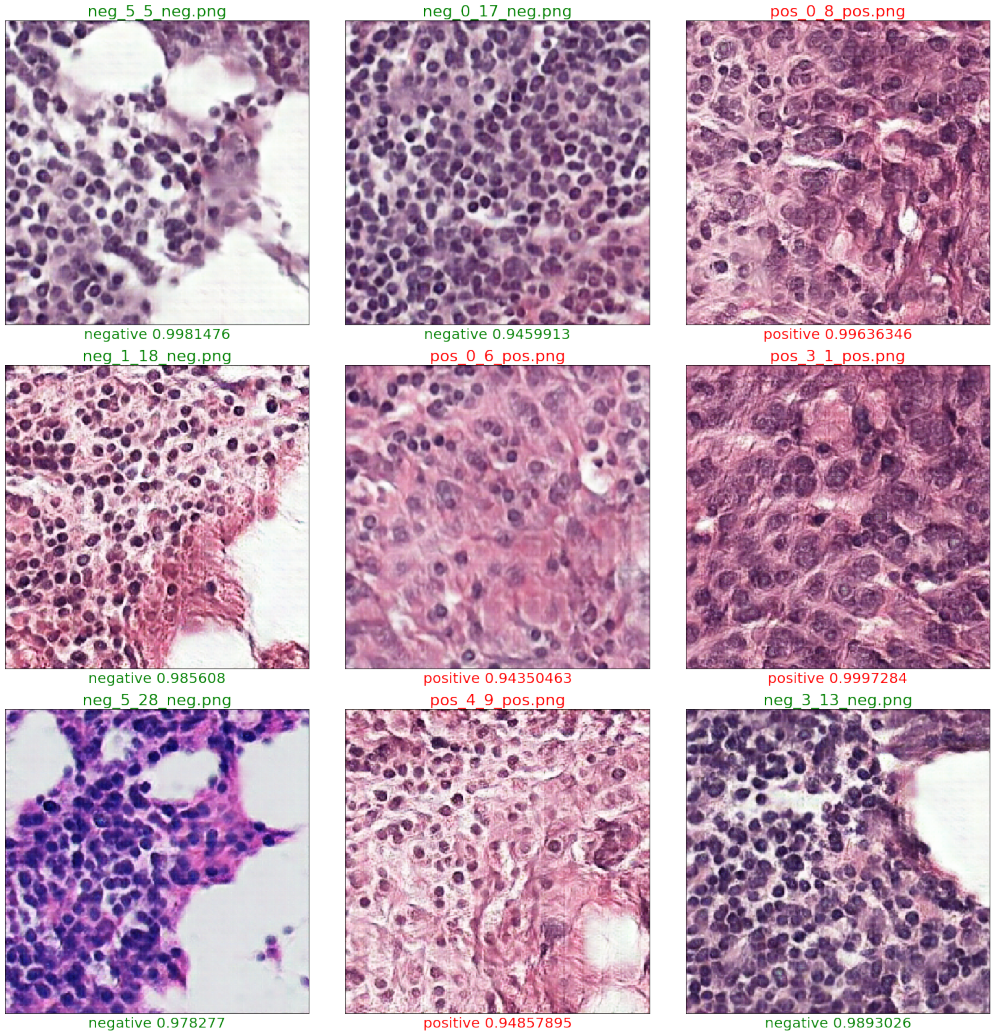


Figura 5.1: Resultados al aplicar a algunas de nuestras muestras un clasificador Inception V3 entrenado con el dataset Imagenet en el que se han reentrenado, con nuestro dataset, las últimas capas mediante transfer learning

3. Que sean capaz de detectar distorsiones y transformaciones. La puntuación que obtenga un modelo no debe variar por el hecho de que las imágenes que genera contengan pequeñas variaciones que no cambien el sentido semántico de las mismas, como una ligera rotación, desplazamientos o cortes.
4. Un buen método de evaluación debe de concordar o ir en la misma línea que las evaluaciones y las percepciones humanas.
5. Un método de evaluación es mejor cuanto más sencillo sea, tanto desde un punto de vista conceptual como computacional.

Esta es la teoría, pero puede ser complicado encontrar un método que cumpla todas estas características. En nuestro caso optamos por la evaluación visual combinada con un método que fuese sencillo y fácil de interpretar y de implementar. En la fase final del proyecto se cambió la arquitectura de la red de DCGAN a cDCGAN lo que nos facilitó una parte de la evaluación ya que pasamos a disponer de la clase a la que pertenece cada muestra.

En nuestro caso no se ha optado por usar el Inception Score porque da la impresión de que esta puntuación está más orientada a clasificadores de objetos en un ámbito mucho más general que la histología (al estar entrenado con Imagenet), de manera que la puntuación numérica que obtendríamos sería injustamente más baja y de difícil comparación con otras puntuaciones referencia de Inception.

El método que implementamos hace uso del clasificador Inception de una manera distinta. Mediante transfer learning[39], se entrena un clasificador Inception V3 sobre dos de nuestros datasets, uno compuesto de imágenes histológicas reales,

```
INFO:tensorflow:2019-06-01 19:31:11.053180: Step 1650: Train accuracy = 86.0%
I0601 19:31:11.053232 140152645003072 retrain.py:1107] 2019-06-01 19:31:11.053180: Step 1650: Train accuracy = 86.0%
INFO:tensorflow:2019-06-01 19:31:11.053358: Step 1650: Cross entropy = 0.323803
I0601 19:31:11.053369 140152645003072 retrain.py:1109] 2019-06-01 19:31:11.053358: Step 1650: Cross entropy = 0.323803
INFO:tensorflow:2019-06-01 19:31:11.110800: Step 1650: Validation accuracy = 83.0% (N=100)
I0601 19:31:11.110852 140152645003072 retrain.py:1128] 2019-06-01 19:31:11.110800: Step 1650: Validation accuracy = 83.0% (N=100)
INFO:tensorflow:2019-06-01 19:31:12.831232: Step 1660: Train accuracy = 88.0%
I0601 19:31:12.831281 140152645003072 retrain.py:1107] 2019-06-01 19:31:12.831232: Step 1660: Train accuracy = 88.0%
INFO:tensorflow:2019-06-01 19:31:12.831401: Step 1660: Cross entropy = 0.266865
I0601 19:31:12.831414 140152645003072 retrain.py:1109] 2019-06-01 19:31:12.831401: Step 1660: Cross entropy = 0.266865
INFO:tensorflow:2019-06-01 19:31:12.886047: Step 1660: Validation accuracy = 80.0% (N=100)
```

Figura 5.2: Un par de pasos del entrenamiento de inception con imágenes generadas

que se evalúa sobre un conjunto de test de imágenes generadas por la GAN, y otro dataset de imágenes artificiales que se testea sobre un dataset de imágenes reales. Lo que se pretende probar es que si las imágenes son realistas, es decir contienen coherentemente características de las imágenes reales, entonces el clasificador obtendrá buena precisión ya que puede aprender usando esas características y usar la representación aprendida para clasificar con precisión imágenes reales. Por otro lado si la versión entrenada con imágenes reales obtiene buena puntuación evaluando imágenes generadas es que estas reproducen los patrones de las imágenes reales. La evaluaciones se han repetido con distinta cantidad de datos y distintos tiempos de entrenamiento y estos son los resultado obtenidos:

Cuadro 5.1: Resultados evaluacion con imagenes nivel 0

Evaluación cDCGAN entrenada con 120k imgs. nivel 0 – 76k pasos (∼90 horas)					
N. de muestras	Iter. evaluación	Entrenamiento (5k iterac.) de Inception V3 con imágenes reales		Entrenamiento (5k iterac.) de Inception V3 con imágenes generadas	
		% evaluando img. reales	% evaluando img. gener.	% evaluando img. gener.	% evaluando img. reales
5000	500	0.85	0.764	0.93	0.796
	5000	0.8638	0.7746	0.9366	0.8032
15000	500	0.882	0.78	0.96	0.82
	5000	0.873	0.7984	0.9436	0.8158

No debemos fijarnos en los valores absolutos ya que el clasificador solo ha sido entrenado durante 5000 iteraciones (30 min aprox.) de ahí que las precisiones no sean excesivamente altas. El objetivo de la evaluación es comparar la diferencia de precisión al entrenar y evaluar con imágenes reales o generadas además de observar como se comporta la precisión al aumentar el numero de muestras.

De los resultados se pueden sacar algunas conclusiones:

- La mejor precisión al entrenar el clasificador con las imágenes generadas que con las reales se puede interpretar como una falta de diversidad en las muestras y/o una estructura más simple de las mismas por lo que al clasificador le es más fácil aprender y capturar esa información.
- Los resultados al clasificar imágenes reales con el modelo entrenado a partir de las generadas son sólo ligeramente inferiores que cuando se usa el clasificador entrenado con imágenes reales, por lo cual se deduce que el rendimiento del modelo generador es aceptable y es capaz de generar imágenes realistas.
- Al aumentar el número de muestras, aumenta la precisión al clasificar muestras reales. Esto es relevante en el caso del entrenamiento con imágenes generadas ya que implica que la red no está generando muestras redundantes. La precisión aumenta al clasificar imágenes reales (aumenta en menor medida al clasificar imágenes generadas) y aproximadamente en la misma proporción con la que lo hace al aumentar el número de muestras en el caso del entrenamiento con las muestras reales.
- Las imágenes con menos zoom son mas fáciles de generar de manera realista que las de nivel 0.

Cuadro 5.2: Resultados de la evaluación con imágenes nivel 1

Evaluación cDCGAN entrenada con 44k imgs. nivel 1 – 66k pasos (∼84 horas)					
N. de muestras	Iter. evaluación	Entrenamiento (5k iterac.) de Inception V3 con imágenes reales		Entrenamiento (5k iterac.) de Inception V3 con imágenes generadas	
		% evaluando img. reales	% evaluando img. gener.	% evaluando img. gener.	% evaluando img. reales
5000	500	0.904	0.856	0.944	0.848
	5000	0.9096	0.8384	0.9424	0.8536
15000	500	0.896	0.798	0.938	0.872
	5000	0.9006	0.821	0.9446	0.8592

Evaluación de las muestras obtenidas mediante aritmética de vectores

Una vez hemos evaluado de manera objetiva la capacidad del modelo para generar muestras realistas de las dos clases realizamos una serie de pruebas basadas en la inspección visual con las que intentaremos valorar el resultado de las operaciones con vectores conceptuales así como la eficacia del entrenamiento a la hora de construir un mapeo coherente del espacio latente. En el primer caso tenemos que valorar si el resultado de las operaciones que realizamos responden a la suma o resta (o media) de conceptos presentes en las imágenes con las que se han realizado dichas operaciones. Por lo que podemos observar en las figuras 5.3a y 5.3b parece que este es el caso.

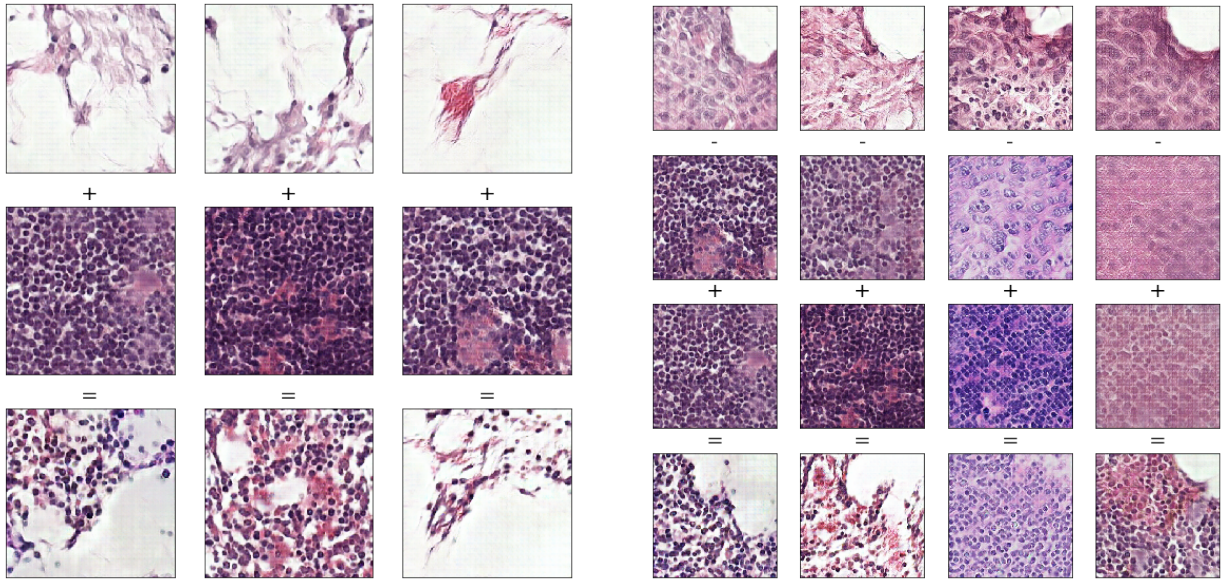


Figura 5.3: Resultado de distintas operaciones aritméticas con vectores conceptuales. En los anexos y los notebooks Jupyter entregados se dan más ejemplos y detalles de este tipo de operaciones

En el segundo caso podemos probar avanzando en distintas direcciones dentro del espacio y comprobar que las muestras obtenidas en cada paso muestran coherencia espacial. Las 4 imágenes de la figura 5.4a son los puntos de inicio y de partida de las interpolaciones y se corresponden con las esquinas de la figura 5.4b. Como podemos observar la imagen

es coherente en cualquiera de las 6 direcciones sobre las que avanzamos, en las que vemos desaparecer y aparecer diferentes atributos de manera que la imagen siempre tiene sentido.

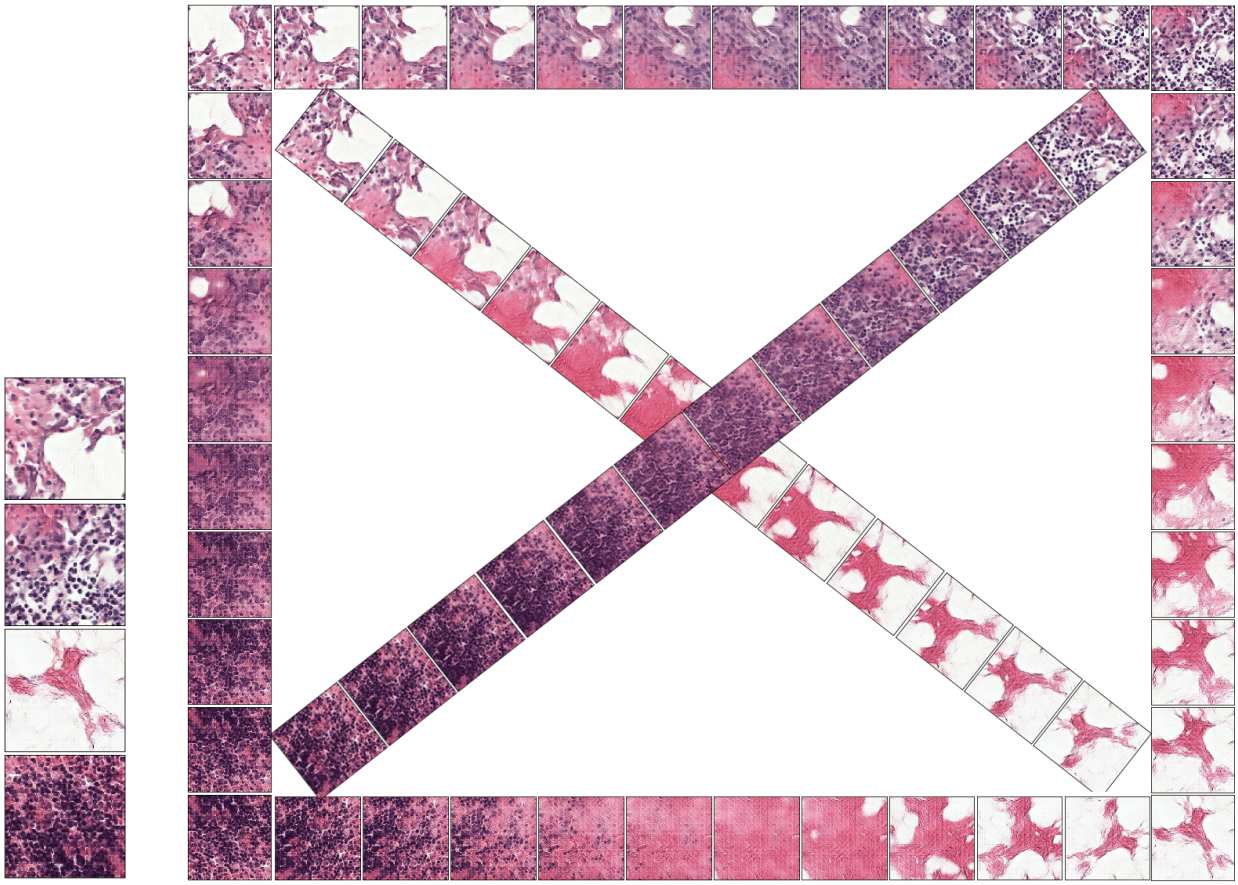


Figura 5.4: Puntos de origen y destino de las interpolaciones (izquierda) e interpolaciones en 6 direcciones distintas (derecha). Las 4 imágenes de la izquierda se corresponden con las esquinas del mosaico de la derecha

6 | Conclusiones

Con la realización de este trabajo hemos mostrado que es posible editar imágenes histológicas mediante aritmética de vectores conceptuales obteniendo resultados prometedores. Sin embargo es necesario seguir investigando con distintas arquitecturas y mejor hardware para llegar mas lejos en el entrenamiento de la red y así obtener mejores resultados tanto al aplicar operaciones aritméticas como en lo que se refiere a la calidad de imagen de las muestras.

También es necesario investigar posibles aplicaciones de las imágenes generadas mediante este método más allá del aumento de datos. Algunas aplicaciones que parecen interesantes podrían ser su uso como material didáctico o otros usos en los que la rigurosidad a nivel médico no sea tan determinante. Como ejemplo de aplicación didáctica se podría diseñar un portal web en el que a partir de dos imágenes elegidas por el usuario se pudiesen visualizar una interpolación entre ellas, permitir al usuario navegar en el espacio latente y visualizarlo en cada punto o aplicar otro tipo de operaciones y mostrar su resultado.

Su uso en *data augmentation* necesita ser explorado de manera más rigurosa mediante el uso de métodos de evaluación que se ajusten a las particularidades de cada tipo de imagen y a esta manera de generar imágenes. Sería interesante extender el análisis también a otro tipo de imágenes distintas a las histopatológicas, que quizás por su naturaleza microscópica muestran comportamientos mas difíciles de representar mediante operaciones aritméticas. Un caso interesante a nivel más macroscópico podría ser su utilización con imágenes de MRI o CT de distintos órganos o con imágenes óseas con las cuales intentar generar imágenes que representen crecimientos, fracturas o otro tipo de transiciones entre distintos estados.

Si continuamos hablamos de imagen médica de manera más general que refiriéndonos a las imágenes histológicas, también podría ser interesante investigar el uso de este tipo de operaciones para anonimizar imágenes de pacientes de cara a su uso público. Si se desea por ejemplo eliminar alguna característica que permita la identificación del paciente antes de usar una imagen médica en una publicación, en lugar de editar las imágenes manualmente, que puede ser un proceso tedioso si se trata de muchas imágenes o si son complicadas de editar porque la característica se encuentra muy intrincada en la imagen, una edición conceptual podría ser útil.

Por otro lado, ya en cuanto a otro tipo de conclusiones relacionadas con el uso y el entrenamiento de las DCGAN, podemos concluir que aunque los resultados son espectaculares a nivel visual, entrenar y utilizar una GAN no es una tarea sencilla y necesita ser explorada desde la experimentación ya que no se trata de una solución que pueda ser instalada y utilizada a la primera. Como hemos comentado a lo largo de toda la memoria, tiene una serie de problemas que necesitan ser abordados (o ya lo están siendo) en futuras arquitecturas de modo que sobre todo su configuración sea más fácil y generalizable y su entrenamiento más estable y menos incierto.

En cuanto al propio proceso de desarrollo, una conclusión a la que hemos llegado es que aunque parezca abrumador implementar la versión del *paper* directamente puede que a veces sea la mejor opción. Muchos de los errores que arrastramos procedían de intentar adaptar o escalar ejemplos sencillos que están disponibles en la red. No es necesario prescindir de estos ejemplos, ya que son herramientas muy valiosas para el aprendizaje, pero se debería de separar esa fase de la implementación real y no reutilizar o adaptar las pruebas de concepto.

Por último, cabe mencionar también la complicada tarea que supone la evaluación de los resultados. Aunque para valorar las imágenes generadas disponemos de una serie de métodos más o menos automatizados (no exentos de sus propias problemáticas), en el caso de la evaluación de las imágenes obtenidas mediante aritmética de vectores conceptuales no parece claro cual debe de ser la manera correcta de evaluarlas, a parte de la inspección visual, que es costosa y de carácter subjetivo.

7 | Anexos

7.1. Guia de instalación del entorno

Esta es una guía para instalar el entorno en una sistema Ubuntu 18.04. Aunque se incluyen detalles de como instalar ciertos componentes para poder usar GPUs AMD la guía es igualmente valida para tarjetas NVIDIA, simplemente debemos saltar la parte de instalación de rocm y tensorflow-rocm e instalar directamente la versión estándar de tensorflow.

El primer paso sera instalar Conda. Conda es un gestor de paquetes, dependencias y entornos virtuales de python. No sustituye a pip (el gestor de paquetes por excelencia) sino que podremos usar ambos para instalar paquetes dentro de los entornos virtuales creados con Conda.

Conda puede ser instalado con la distribucion Anaconda que incluye numerosos paquetes cientificos de python ademas de otras utilidades y programas, o con Miniconda que incluye los componentes mínimos. En este tutorial usaremos miniconda por que ocupa mucho menos y no nos importa instalar los paquetes que necesitemos a mano, pero puedes usar Anaconda si lo prefieres.

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda-latest-Linux-x86_64.sh
./Miniconda-latest-Linux-x86_64.sh

##En caso de que despues de reloguearnos no se reconozca el comando conda
##puede que necesitemos añadir el directorio al PATH
echo "PATH=$PATH:$HOME/miniconda/bin" >> .bashrc
```

Ahora podemos crear nuestro entorno virtual para el proyecto. Podemos especificar la versión de python que necesitemos y esta sera la usada dentro del entorno, independientemente de la versión de python que ya tengamos en nuestro sistema

```
##Con -n especificamos un nombre para el entorno, el que queramos
conda create -n tensorflow python=3.6.5
```

Una vez creado podemos activarlo, desactivarlo o borrarlo:

```
conda activate tensorflow
conda deactivate tensorflow
conda remove -n tensorflow --all
```

Si activamos el entorno todos los paquetes que instalemos estarán solo disponibles dentro de ese entorno y no tendremos conflictos de versiones con otros instalados ni en el sistema base ni en ningún otro entorno creado.

```
conda activate tensorflow
##podemos instalar un paquete con el siguiente comando
```

```
conda install numpy
##tambien podemos installar pip e utilizarlo como gestor de paquetes
##(hay mas paquetes que en conda)
conda install pip
pip install openslide-python
```

El siguiente paso es instalar las herramientas necesarias de la plataforma ROCm. Esta plataforma esta orientada o acelerar tareas computacionales típicamente usadas en machine learning o computación científica. ROCm se base en OpenCL, la alternativa open source del CUDA de Nvidia. Ademas ROCm incluye una serie de librerías de desarrollo y herramientas encima de OpenCL para hacer el codigo escrito para CUDA mas compatible con este y viceversa. Gracias han podido obtener una version de tensorflow que funciona con tarjetas AMD y se esta portando codigo upstream de manera que en algun momento la version oficial de tensorflow no contenga codigo especifico CUDA.

Para instalarlo podemos seguir los pasos indicados en la pagina de ROCm[]:

```
##actualizamos el sistema
sudo apt update
sudo apt dist-upgrade
sudo apt install libnuma-dev
sudo reboot

##añadimos el repositorio de rocm
wget -qO - http://repo.radeon.com/rocm/apt/debian/rocm.gpg.key | sudo apt-key add -
echo 'deb [arch=amd64] http://repo.radeon.com/rocm/apt/debian/ \
xenial main' | sudo tee /etc/apt/sources.list.d/rocm.list

##e instalamos el paquete rocm-dkms que contiene tambien el modulo del kernel
sudo apt update
sudo apt install rocm-dkms

##por ultimo es necesario que nos añadamos al grupo video para poder usar la GPU
sudo usermod -a -G video $LOGNAME
echo 'ADD_EXTRA_GROUPS=1' | sudo tee -a /etc/adduser.conf
echo 'EXTRA_GROUPS=video' | sudo tee -a /etc/adduser.conf
```

Para ver que todo ha ido correctamente podemos ejecutar los siguientes comandos y comprobar que nuestras gráficas están correctamente detectadas.

```
/opt/rocm/bin/rocminfo
/opt/rocm/opencl/bin/x86_64/clinfo
```

```
ruben@dumeiro:~$ /opt/rocm/opencl/bin/x86_64/clinfo
Number of platforms:                               1
  Platform Profile:                                FULL_PROFILE
  Platform Version:                                OpenCL 2.1 AMD-APP (2679.0)
  Platform Name:                                   AMD Accelerated Parallel Processing
  Platform Vendor:                                 Advanced Micro Devices, Inc.
  Platform Extensions:                             cl_khr_icd cl_amd_event_callback


  Platform Name:                                   AMD Accelerated Parallel Processing
Number of devices:                                 2
  Device Type:                                     CL_DEVICE_TYPE_GPU
  Vendor ID:                                       1002h
  Board name:                                      Ellesmere [Radeon RX 470/480/570/570X/580/580X]
  Device Topology:                                 PCI[ B#1, D#0, F#0 ]
  Max compute units:                               36
  Max work items dimensions:                       3
```

Figura 7.1: Si la instalación ha sido correcta deberíamos obtener algo así.

Una vez configurado el entorno base, pasaremos a instalar los paquetes necesarios. En primer lugar tensorflow y tensorboard, para ello ejecutamos las siguientes instrucciones:

```
##algun paquete extra de rocm necesario
sudo apt install rocm-libs miopen-hip cxlactivitylogger
##instalamos tensorflow (rocm) y tensorboard
pip install tensorflow-rocm
pip install tensorboard
#para el resto de paquetes en general podemos usar conda
#y si alguno no esta disponible probamos con pip
conda install jupyter
...
...
conda install pillow
...
conda install pandas
...
...
pip install opencv-python
...
```

Para probar la instalación podemos descargarnos un ejemplo sencillo de tensorflow y ejecutarlo

```
cd ~ && git clone https://github.com/tensorflow/models.git
cd ~/models/tutorials/image/imagenet
python classify_image.py
```

Una vez llegados a este punto deberíamos ser capaces de clonar el repositorio y de reproducir por completo el ejemplo de uso descrito en el capitulo 4 sin mayores problemas que la falta de algún paquete python que procederíamos a instalar con conda o pip.

7.2. Notebook de operaciones en el espacio latente

En esta sección reproduciré de la mejor manera posible solo uno de los tres notebook Jupyter que utilicé para mostrar los resultados. La mejor manera de obsérvalos, tanto este como los otros tres, más el notebook de evaluación, es directamente con los ficheros .ipynb o .html que forman parte también de los entregables. Ahí podremos ver todo mejor formateado y con las animaciones en movimiento y el anexo no quedará tan redundante.

Comenzamos cambiando la configuración activa

```
dataset = "camelyon_17_16_level0"
change_active_config(dataset)
```

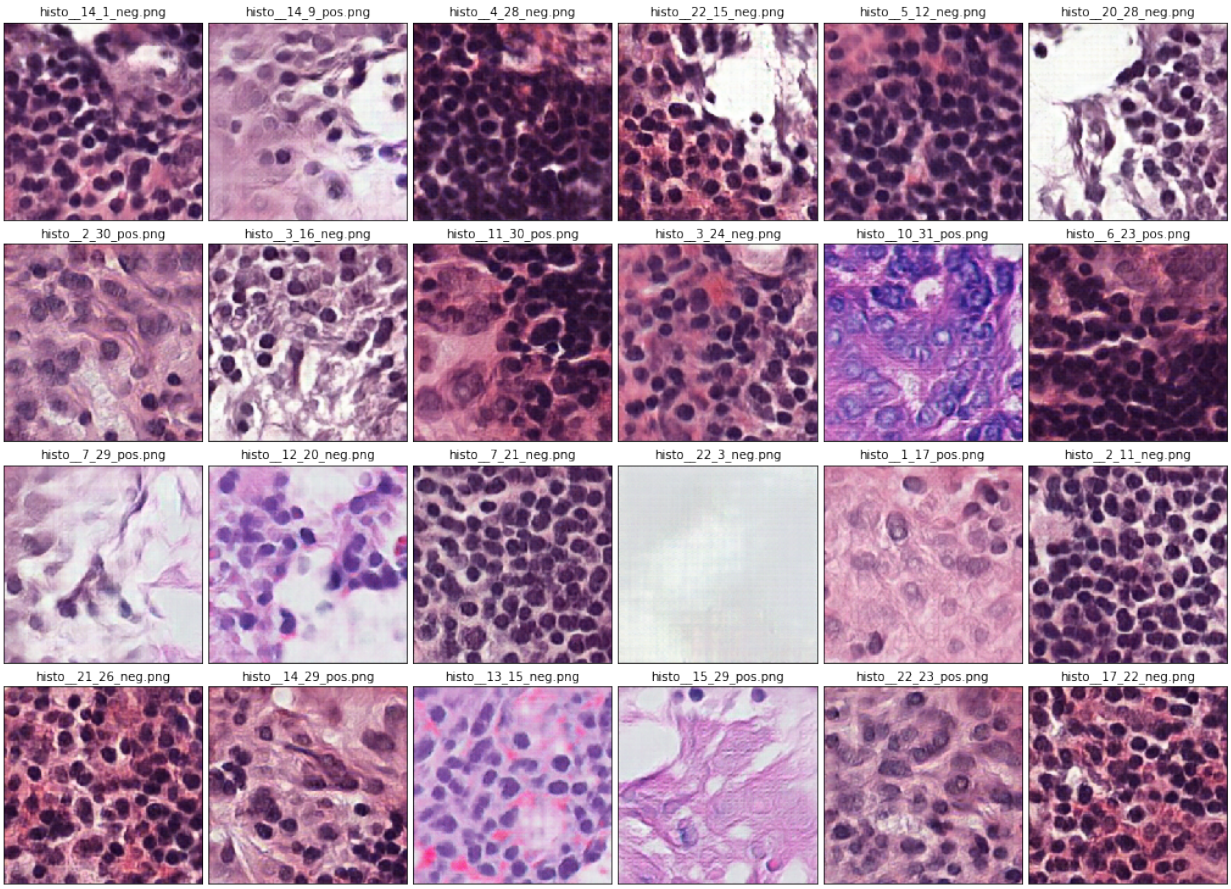
Y utilizamos este función que hemos definido para obtener una sesión de tensorflow y poder usar nuestra arquitectura y el modelo ya entrenado. El numero que se muestra es el numero de iteraciones con las que se guardo el checkpoint

```
session, input_real, input_z, input_g_y= get_session(1)
z_batch_tensor = tf.random.uniform(
(BATCH_SIZE, Z_NOISE_DIM), dtype=tf.float32, minval=-1, maxval=1)
prefix = "histo_"
```

```
INFO:tensorflow:Restoring parameters from /home/ruben/Master/camelyon_17_16_level0/checkpoints/model-76000
76000
```

Para poder realizar la aritmética necesitamos primero generar algunas muestras. Las identificamos con el prefijo _histo de manera que las podamos borrar y mostrar posteriormente

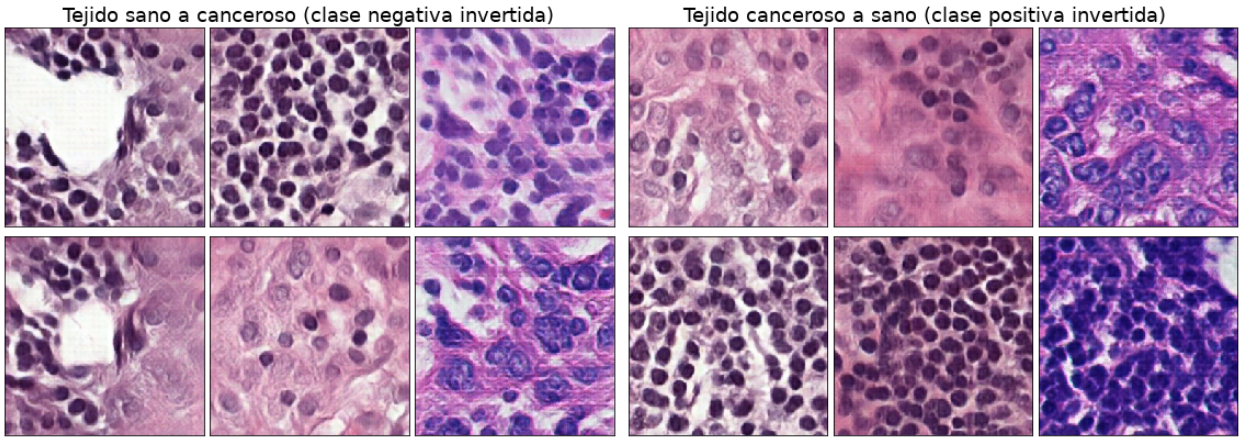
```
#clean up
[os.remove(file) for file in glob(FULL_OUTPUT_PATH+"*"+prefix+"*.*")]
#Generamos unas 3000 imagenes (32*100)
labels = np.random.randint(0,2,BATCH_SIZE)
output.generate_samples(session, z_batch_tensor, input_z, input_g_y, labels, BATCH_SIZE * 100, save_tensor=True, name_prefix=prefix)
all_files = glob(FULL_OUTPUT_PATH+"*"+prefix+"*.png")
files_to_show = random.sample(all_files,24)
plt.figure(figsize=(18, 13))
plt.subplots_adjust(hspace=0.12, wspace=0.03)
for i,img_file in enumerate(files_to_show):
img = plt.imread(img_file)
plt.subplot(4, 6, i+1).set_title(img_file.split("/")[-1]).set_fontsize(10)
plt.imshow(img, aspect="auto")
plt.xticks([])
plt.yticks([])
i=i+1
```



Ahora invertiremos el vector de etiquetas. Cambio de clase de una muestra, condicionando el sistema de manera que usemos el mismo tensor pero con la etiqueta de otra clase.

```
def show(labels):
plt.figure(figsize=(20, 20))
plt.subplots_adjust(hspace=0.2, wspace=0.03)
for i,img in enumerate(tensors):
img = np.expand_dims(img,0)
z_ = tf.placeholder(tf.float32, [1, Z_NOISE_DIM])

samples = session.run(dcgan.sampler(z_,input_g_y,1), feed_dict={z_: img,input_g_y:labels})
img = np.squeeze(samples)
subplt = plt.subplot(1, 6, i+1)
plt.imshow(img / 2 + 0.5)
plt.xticks([])
plt.yticks([])

cancer_1 = np.loadtxt(FULL_OUTPUT_PATH + 'histo__70_2_pos.txt')
cancer_2 = np.loadtxt(FULL_OUTPUT_PATH + 'histo__0_4_pos.txt')
cancer_3 = np.loadtxt(FULL_OUTPUT_PATH + 'histo__0_23_pos.txt')
normal_lumen_1 = np.loadtxt(FULL_OUTPUT_PATH + 'histo__2_13_neg.txt')
normal_lumen_2 = np.loadtxt(FULL_OUTPUT_PATH + 'histo__33_13_neg.txt')
normal_lumen_3 = np.loadtxt(FULL_OUTPUT_PATH + 'histo__70_29_neg.txt')

tensors = [cancer_1,cancer_2,cancer_3,
normal_lumen_1,normal_lumen_2,normal_lumen_3]

labels = [1]
labels = np.eye(len(labels)+1)[labels]
```

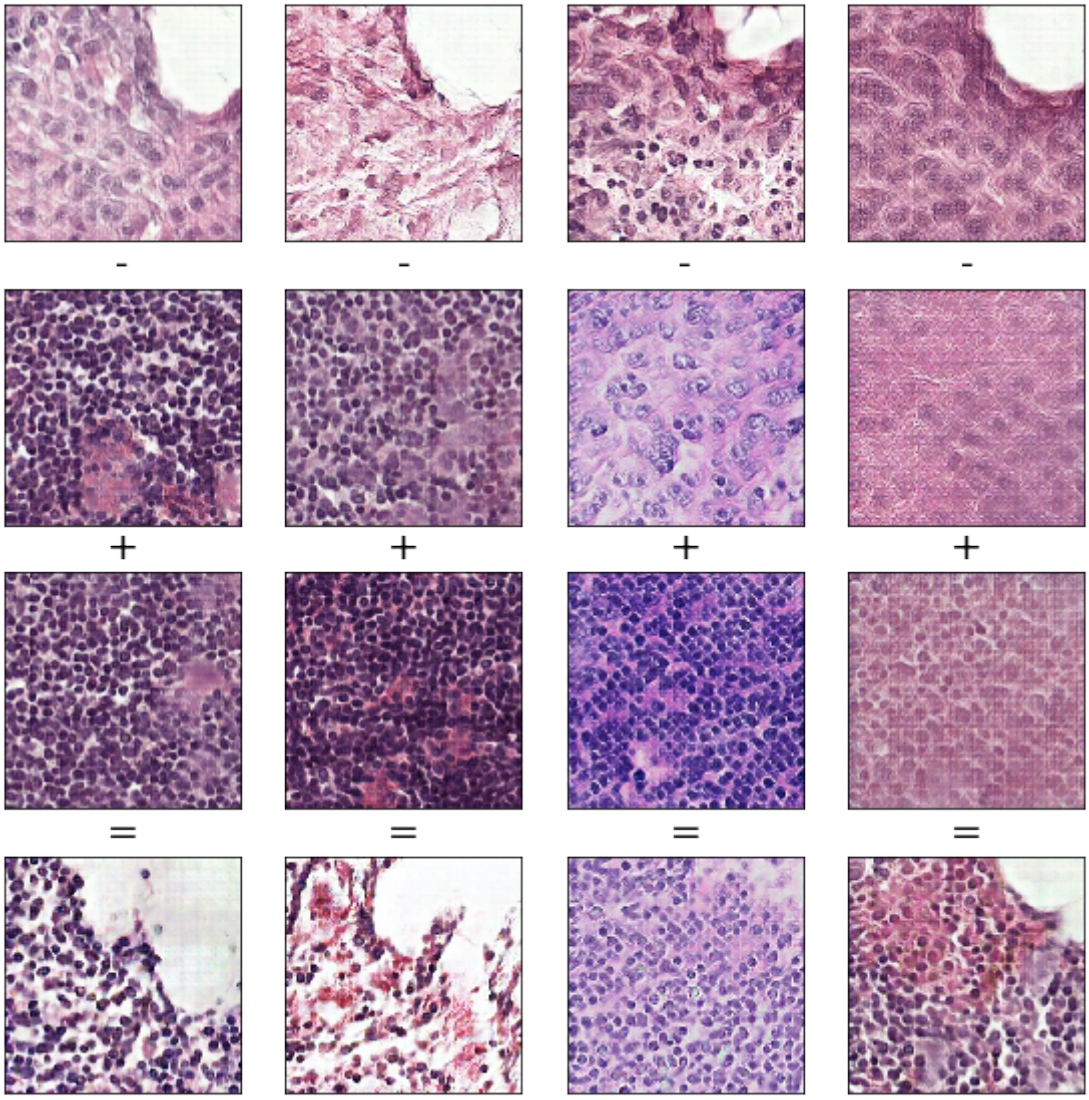
```
show(labels)
labels = [0]
labels = np.eye(len(labels)+1)[labels]
show(labels)
```



Seleccionamos 3 imágenes negativas y en los que se vea el lumen, otras 3 que no sean compactas y donde no se vea, y otras 3 cancerosas también sin lumen. Vamos a realizar la operación 3 - 3 + 3 para obtener una imagen cancerosa con con lumen.

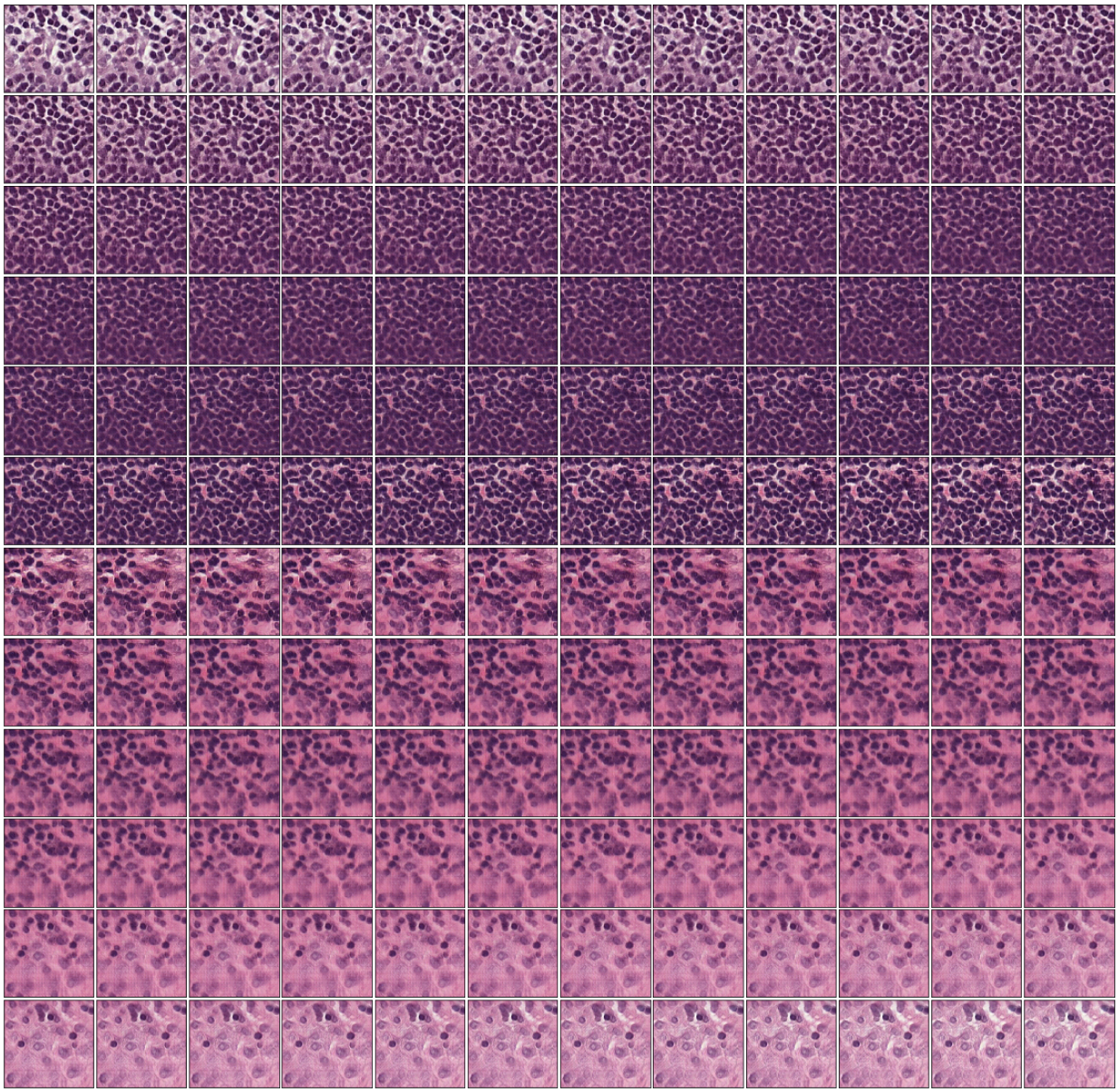
```
resultado_1 = normal_no_tub_1 - normal_tub_1 + cancer_no_tub_1
resultado_2 = normal_no_tub_2 - normal_tub_2 + cancer_no_tub_2
resultado_3 = normal_no_tub_3 - normal_tub_3 + cancer_no_tub_3

normal_tub = latent_space.average_tensor([normal_tub_1,normal_tub_2,normal_tub_3])
normal_no_tub = latent_space.average_tensor([normal_no_tub_1,normal_no_tub_2,normal_no_tub_3])
cancer_no_tub = latent_space.average_tensor([cancer_no_tub_1,cancer_no_tub_2,cancer_no_tub_3])
resultado =  normal_no_tub  - normal_tub + cancer_no_tub_1

resultados = [normal_tub_1,normal_tub_2,normal_tub_3,normal_tub,
normal_no_tub_1,normal_no_tub_2,normal_no_tub_3,normal_no_tub,
cancer_no_tub_1,cancer_no_tub_2,cancer_no_tub_3,cancer_no_tub,
resultado_1,resultado_2,resultado_3,resultado]
plt.figure(figsize=(10, 10))
plt.subplots_adjust(hspace=0.2, wspace=0.03)
labs = [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1]
for i,img in enumerate(resultados):
     img = np.expand_dims(img,0)
     z_ = tf.placeholder(tf.float32, [1, Z_NOISE_DIM])
     labels = np.eye(2)[[labs[i]]]
     samples = session.run(dcgan.sampler(z_,input_g_y,1), feed_dict={z_: img, input_g_y:labels})
     img = np.squeeze(samples)
     subplt = plt.subplot(4, 4, i+1)
     subplt.set_title("" if i<4 else "-" if (i>3 and i<8) else "+" if (i>7 and i<12) else "=").set_fontsize(20)
     plt.imshow(img / 2 + 0.5)
     plt.xticks([])
     plt.yticks([])
```

Otro tipo de operación que podemos realizar es elegir dos tensores y avanzar por la "linea"que los une en el espacio latente, creando así una interpolación suave de imágenes intermedias. El siguiente GIF fue realizado con la versión anterior entregada, consistente en un arquitectura DCGAN, como se puede ver la transición entre un tejido sano y enfermo es suave y coherente paso a paso.



No sucede lo mismo en el caso de la DCGAN condicional. En este caso hemos hechos dos interpolaciones, del punto a al b y del punto b al a, pero cambiando la clase en la segunda interpolación. Empezamos con una imagen cancerosa convertida a negativa pasamos a una negativa (normal), luego ese misma normal convertida en cancerosa y terminamos con la cancerosa.

```
punto_a = np.loadtxt(FULL_OUTPUT_PATH + 'histo__16_2_pos.txt')
punto_b = np.loadtxt(FULL_OUTPUT_PATH + 'histo__41_10_neg.txt')
latent_space.interpolate(session, punto_a, punto_b, [0,0], 72, prefix = "a")
latent_space.interpolate(session, punto_b, punto_a, [1,1], 72, prefix = "b")
all_files = glob(FULL_OUTPUT_PATH+"*interp*")
all_files.sort(key=natural_keys)
plt.figure(figsize=(20, 20))
plt.subplots_adjust(hspace=0.03, wspace=0.03)
for i,img in enumerate(all_files):
    img = plt.imread(img)
    plt.subplot(12, 12, i+1)
    plt.imshow(img, aspect="auto")
    plt.xticks([])
    plt.yticks([])
```



Como podemos observar hay un salto mas o menos abrupto al cambiar la clase. Si mantenemos la interpolación dentro de la misma clase obtenemos una transición continua y suave, y en el caso del siguiente gif, infinita, ya que hacemos la interpolación hacia adelante y luego volvemos al punto de origen

Por último alguna de las pruebas efectuadas con la búsqueda inversa en el espacio latente. En el primer ejemplo vemos que la perdida después de 50000 iteraciones es prácticamente 0 y las dos imágenes son idénticas. En este caso se trata de la mapeo inverso de una imagen que había sido generada previamente por la red. El segundo caso, el resultado después de 100k iteraciones no es tan bueno, al tratarse de una imagen real, y obtenemos 0.08 de error e imágenes ligeramente similares.

Figura 7.2: Resultado de la búsqueda de imágenes en el espacio latente. A la izquierda búsqueda con una imagen generada y a la derecha con una real

Glosario

API
Interfaz de Programación de Aplicaciones. 8, 51

cDCGAN
Ver: Red Generativa Antagónica Convolucional Profunda Condicional. 8, 22, 27, 51

CIFAR
Instituto Canadiense de Investigaciones Avanzadas (Canadian Institute for Advanced Research). 15, 51

CNN
Ver: Red Neuronal Convolucional. 6, 8, 12, 15–18, 51

CT
Tomografía computarizada: procedimiento para obtener imagenes de rayos X del interior del cuerpo desde distintos angulos con las que posteriormente construir imagenes 3D por medio de un computador. 6, 8, 51

DCGAN
Ver: Red Generativa Antagónica Convolucional Profunda. 12, 15, 19, 22, 27, 51

Deep Learning
. 4, 6, 12, 14–17, 37, 51

GAN
Ver: Red Generativa Antagónica. 6–9, 12, 14–16, 19, 20, 22, 24, 29, 30, 37, 51

GPU
Unidad de Procesamiento Gráfico. 8, 51

MRI
Imagen por Resonancia Magnética: procedimiento por el cual se obtienen imágenes del interior del cuerpo desde distintos ángulos con las luego se construyen computacionalmente imagenes en 3D. 6, 8, 51

PET
Tomografía por Emisión de Positrones. 8, 51

Red Generativa Antagónica
. 6–9, 12, 13, 16, 19, 20, 22, 37, 51, 52

Red Generativa Antagónica Convolucional Profunda
. 12, 51, 52

Red Generativa Antagónica Convolucional Profunda Condicional
. 51, 52

Red Neuronal Convolucional
. 6, 15, 16, 51, 52

Whole Slide Image

Digitalizacion del cristal portaobjetos o slide de microscopio mediante un escaner. La imagen obtenida es una replica en alta resolucion del portaobjetos completo. 7, 8, 51

WSI

Ver: Whole Slide Image. 7, 8, 13, 16, 23, 51

Bibliografía

[1] Kunal Nagpal, Davis Foote, Yun Liu, Po-Hsuan, Chen, Ellery Wulczyn, Fraser Tan, Niels Olson, Jenny L. Smith, Arash Mohtashamian, James H. Wren, Greg S. Corrado, Robert MacDonald, Lily H. Peng, Mahul B. Amin, Andrew J. Evans, Ankur R. Sangoi, Craig H. Mermel, Jason D. Hipp, and Martin C. Stumpe. Development and Validation of a Deep Learning Algorithm for Improving Gleason Scoring of Prostate Cancer. *arXiv e-prints*, page arXiv:1811.06497, Nov 2018.

[2] Bo Hu, Ye Tang, Eric Chang, Yubo Fan, Maode Lai, and Yan Xu. Unsupervised learning for cell-level visual representation in histopathology images with generative adversarial networks. *IEEE Journal of Biomedical and Health Informatics*, PP, 11 2017.

[3] Xin Yi, Ekta Walia, and Paul Babyn. Generative adversarial network in medical imaging: A review. 09 2018.

[4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 11 2015.

[5] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in Liver Lesion Classification. *arXiv e-prints*, page arXiv:1803.01229, Mar 2018.

[6] Hojjat Salehinejad, Shahrokh Valaee, Tim Dowdell, Errol Colak, and Joseph Barfett. Generalization of Deep Neural Networks for Chest Pathology Classification in X-Rays Using Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1712.01636, Nov 2017.

[7] Xin Yi and Paul Babyn. Sharpness-aware Low dose CT denoising using conditional generative adversarial network. *arXiv e-prints*, page arXiv:1708.06453, Aug 2017.

[8] Xin Yi, Ekta Walia, and Paul Babyn. [table iv]: Generative adversarial network in medical imaging: A review, 09 2018.

[9] Alper Aksac, Douglas J. Demetrick, Tansel Ozyer, and Reda Alhajj. Brecahad: a dataset for breast cancer histopathological annotation and diagnosis. *BMC Research Notes*, 12, 12 2019.

[10] Ellis I.O. Elston C.W. Pathological prognostic factors in breast cancer. i. the value of histological grade in breast cancer: experience from a large study with long-term follow-up. *Histopathology*, 41:154–61, 1991.

[11] Zheng Ping, Yuchao Xia, Tiansheng Shen, Vishwas Parekh, Gene Siegal, Isam Eltoum, Jianbo He, Dongquan Chen, Minghua Deng, Ruibin Xi, and Dejun Shen. A microscopic landscape of the invasive breast cancer genome. *Scientific Reports*, 6:27545, 06 2016.

[12] Rubén Fernández. Histology DCGAN Repository. https://github.com/benru89/histology_dcgan/.

[13] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[14] Yann LeCun, Y Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[15] Yufeng Ma, Zheng Xiang, Qianzhou Du, and Weiguo Fan. Effects of user-provided photos on hotel review helpfulness: An analytical approach with deep leaning. *International Journal of Hospitality Management*, 71:120–131, 04 2018.

[16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1406.2661, Jun 2014.

[17] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A Survey on Deep Learning in Medical Image Analysis. *arXiv e-prints*, page arXiv:1702.05747, Feb 2017.

[18] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. 542(7639):115–118, 02 2017.

[19] Varun Gulshan, Lily Peng, Marc Coram, Martin C. Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip C. Nelson, Jessica L. Mega, and Dale R. Webster. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA*, 316, 11 2016.

[20] Dan C. Cireşan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In Kensaku Mori, Ichiro Sakuma, Yoshinobu Sato, Christian Barillot, and Nassir Navab, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, pages 411–418, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[21] Geert Litjens, Clara I Sánchez, Nadya Timofeeva, Meyke Hermsen, Iris Nagtegaal, Iringo Kovacs, Christina Hulsbergen-Van De Kaa, Peter Bult, Bram Van Ginneken, and Jeroen Van Der Laak. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific reports*, 6:26286, 2016.

[22] Salome Kazeminia, Christoph Baur, Arjan Kuijper, Bram van Ginneken, Nassir Navab, Shadi Albarqouni, and Anirban Mukhopadhyay. Gans for medical image analysis. *arXiv preprint arXiv:1809.06222*, 2018.

[23] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[24] K Fukushima. Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] V. Sze, Y. Chen, T. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017.

[27] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[28] Tim Dettmers. Understanding convolution in deep learning, Oct 2015.

[29] Al Gharakhanian Al Gharakhanian. Gans: One of the hottest topics in machine learning.

[30] Ajkel Mino and Gerasimos Spanakis. Logan: Generating logos with a generative adversarial neural network conditioned on color. 10 2018.

[31] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M. Álvarez. Invertible conditional gans for image editing. *CoRR*, abs/1611.06355, 2016.

[32] Camelyon 17 challenge. https://camelyon17.grand-challenge.org/Background/.

[33] Computational Pathology Group Radboud University Medical. ASAP (Automated Slide Analysis Platform). https://github.com/computationalpathologygroup/ASAP.

[34] WSI-analysis. https://github.com/tcxxxx/WSI-analysis.

[35] WSI-analysis forked Rubén. https://github.com/benru89/WSI-analysis.

[36] Histopathology-stain-color-normalization. https://github.com/FarhadZanjani/Histopathology-Stain-Color-Normalization.

[37] P Manisha and Sujit Gujar. Generative Adversarial Networks (GANs): What it can generate and What it cannot? *arXiv e-prints*, page arXiv:1804.00140, Mar 2018.

[38] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. How good is my gan? In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 213–229, 2018.

[39] How to Retrain an Image Classifier for New Categories | TensorFlow Hub | TensorFlow. https://www.tensorflow.org/hub/tutorials/image_retraining.