# Funcionamiento de ArqGAN

**Resumen**

ArqGAN es una solución que se basa en el uso de redes neuronales, particularmente en Redes Generativas Antagónicas (GAN). El proyecto se centra en el estudio de la reconstrucción de imágenes.

El propósito de este modelo es el desarrollo de una red que, tomando como referencia la imagen de unas ruinas, sea capaz de generar no solo su reconstrucción, sino la reconstrucción de ruinas similares mediante la generalización del aprendizaje.

El conjunto de datos debe ser coherente a nivel arquitectónico (ya que este está basado en simulaciones semi realistas) y completo en cuanto al número de posibles supuestos que representan.

Estas imágenes de entrada son imágenes RGB con formato PNG, con dimensiones de 1024x768 píxeles y calidad media. Se han tomado un total de 300 imágenes por cada templo, para sumar un total de 12000 imágenes. Si a estas le sumamos también las imágenes con colores identificativos, es decir, imágenes donde cada elemento de la estructura tiene un color diferente que hace más fácil su identificación (se representan las columnas en rojo y el techo en verde, por ejemplo) obtendremos el doble, ya que son otras 300 por templo, es decir, 24000 imágenes en total.

**Validación**

Las métricas objetivas utilizadas para la validación del modelo es el Test de clasificación de dos muestras (C2ST) donde el objetivo de esta prueba es determinar si dos muestras proceden de la misma distribución de probabilidad o, de igual manera, comprobar si dos distribuciones e. g. P y Q son iguales.

Se evalúa el generador en función del rendimiento de un nuevo discriminador, para ello se crea un nuevo conjunto de validación dividido en dos subconjuntos: validación-entrenamiento y validación-validación. El subconjunto de validación-entrenamiento se emplea para entrenar al nuevo discriminador cuyo objetivo es diferenciar entre imágenes reales y generadas.

Este nuevo discriminador se pone a prueba con el subconjunto de validación- validación para obtener la métrica objetivo; intuitivamente, si , la precisión del nuevo discriminador debería rondar el nivel de probabilidad (50% en el caso de un clasificador binario).

Se usará la validación cruzada, la validación cruzada es un método de validación que busca estimar la capacidad de generalización de un modelo (estadístico, neuronal, etc.) y asegurar que los resultados obtenidos por este no se deben a la división entre los conjuntos de entrenamiento y validación. En el proyecto de ArqGAN este conjunto de datos está formado por el 25% de las imágenes de cada templo.

Esta validación es cruzada de k iteraciones, que busca estimar la capacidad de generalización de un modelo. La validación cruzada de k iteraciones se emplea para obtener una medida precisa del rendimiento del modelo y subsanar los problemas inherentes a la división entrenamiento-validación clásica. El conjunto de entrenamiento se divide en k subconjuntos; se elige uno de estos como subconjunto de validación y los otros k − 1 subconjuntos se emplean como conjunto de entrenamiento. Este proceso se repite k veces, en este caso k es igual a 5.

**Redes Generativas Antagónicas**

Las Redes Generativas Antagónicas o GAN (Generative Adversial networks) buscan transformar variables de un espacio latente a la distribución de datos objetivo. Para entender la distribución del generador sobre los datos , se define prioridad de entrada sobre sobre unas variables de ruido , luego se crea un mapeo en el espacio de datos, donde G es una función diferenciable representada por un perceptrón multicapa con parámetros . Además, se define otro perceptrón multicapa que se llamará cuya salida es un único escalar. representa la probabilidad de que provenga de los datos y no de . Se entrena D para maximizar las probabilidades de asignar la etiqueta correcta a las muestras de entrenamiento y a las de G. Simultáneamente se entrena G para minimizar .

En otras palabras, y forman parte de la siguiente fórmula minimax :

Por ejemplo y para mejorar su compresión, dado un modelo que clasifica imágenes de perros y gatos es sencillo evaluar el resultado; basta con presentar una imagen al modelo, obtener su salida, compararla con la esperada (e.g. 0 para la imagen de un perro y 1 para la de un gato) y obtener el error. En el caso de una red que genere imágenes nuevas de perros y gatos es necesario evaluar que lo generado sea un perro o un gato. Es ahí donde entra en juego la red antagónica.

Para recapitular, el objetivo del entrenamiento es obtener un generador que produzca muestras lo suficientemente convincentes para engañar a . Para esto, ambas redes se entrenan de manera simultánea, se entrena para maximizar la probabilidad de asignar el valor correcto tanto a muestras procedentes del conjunto de entrenamiento como a aquellas generadas por , mientras que busca minimizar este valor.

**Arquitectura del discriminador**

El discriminador de los modelos desarrollados en este proyecto está basado en el discriminador Markoviano. Este asigna un grado de validez único a diferentes regiones de la imagen. Al igual que un filtro convolucional recorre la imagen extrayendo características locales de la entrada, el discriminador Markoviano convoluciona sobre esta, produciendo como salida una matriz con valores entre 0 y 1 en función del grado de validez de las diferentes regiones de la imagen.

La estructura del discriminador es similar a la porción de encoding del generador por un buen motivo: ambas buscan reducir las imágenes a sus características esenciales, si bien el discriminador tiene como fin realizar una inferencia basada en estas características.

Diagrama

Descripción generada automáticamente

**Reconstrucción por pasos**

**Diagrama

Descripción generada automáticamente**La reconstrucción por pasos busca combinar el proceso de reconstrucción directa, que es el proceso por el cual se genera la imagen del templo sin ayuda y reconstrucción asistida, que es el proceso por el cual se reconstruye usando además de la entrada normal, una imagen a color de esta. Durante el entrenamiento, este aprenderá, a partir de la imagen de unas ruinas, a generar y reconstruir una versión segmentada de este, que será empleada posteriormente por un modelo de reconstrucción asistida para generar la imagen final. Esta aproximación parte de la hipótesis de que será más sencillo reconstruir una imagen de las ruinas con colores más simples, sin sombras y con sus elementos claramente delimitados.

El modelo de reconstrucción de redes se compone, por tanto, de tres GAN:

* Una primera red que genere la versión segmentada de las ruinas, es decir, que genere la imagen con colores identificativos de la entrada ().
* Una segunda red que reconstruya esta versión segmentada (.
* Una tercera red que reconstruya la imagen original ayudándose de la reconstrucción segmentada. ()

Donde es la GAN que reconstruye y la que genera las imágenes a color que etiquetan las partes del templo.

Imagen que contiene Dibujo de ingeniería

Descripción generada automáticamenteEn la siguiente imagen veremos un ejemplo de entradas y salidas a la red con los distintos modelos:

Por otro lado, se puede apreciar como las reconstrucciones generadas por el modelo de reconstrucción asistida (tercera columna de la figura 3) son mucho más fieles al objetivo; las líneas rectas se mantienen y pueden observarse detalles. Cabe destacar que estas reconstrucciones están lejos de ser perfectas; la mayoría de ellas son identificables gracias a diferencias de color similares a las que se presentan en la reconstrucción directa.

**Funcionamiento del código – Paso por paso**

Para comprobar el funcionamiento del modelo y ejecutarlo hay que ejecutar el archivo *test.py* identificando en el comando el modelo guardado a utilizar y las muestras. Como ejemplo tenemos:



Luego para la visualización de los datos usamos tensorboard de la siguiente manera:



Para acceder a los datos hay que acceder a[*http://localhost:6006*](http://localhost:6006)*.* En el siguiente ejemplo vemos la reconstrucción:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Funcionamiento del código - Asistida**

Para comprobar el funcionamiento del modelo y ejecutarlo hay que ejecutar el archivo *test.py* identificando en el comando el modelo guardado a utilizar y las muestras. Como ejemplo tenemos:

*python -m test.py --model\_path saved\_models/512x512.pix2pix.color\_assisted.t1456789.h5 --samples\_path dataset\temples\_ruins\temple\_0\_ruins\_0 --samples\_path2 dataset\colors\_temples\colors\_temple\_0*

Luego para la visualización de los datos usamos tensorboard de la siguiente manera:



Para acceder a los datos hay que acceder a[*http://localhost:6006*](http://localhost:6006)*.* En el siguiente ejemplo vemos la reconstrucción:

Interfaz de usuario gráfica

Descripción generada automáticamente