# CONDITIONAL GENERATIVE ADVERSARIAL NETS IN PAINTING ARTWORKS

**Aguirre Daniel, Avella Juan Camilo, Ceballos Juan David**
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
Bogotá

**ABSTRACT**

Conditional GANs [1] fueron introducidas como una extensión de GANs [2] para entrenar modelos generativos condicionados en etiquetas. En este trabajo hacemos una implementación de GAN y cGAN enfocada en el área del arte. Primero, creamos un conjunto de imágenes(pinturas) etiquetadas con su respectivo movimiento artístico. Posteriormente, mostramos la implementación del modelo (cGAN) y su capacidad para generar pinturas condicionadas por los movimientos artísticos (e.g., arte abstracto, arte geométrico, arte islámico, etc.) Como extensión del trabajo, se realizó un GAN sobre un conjunto de datos de arte en internet para contrastar resultados. El código está disponible en https://github.com/Donvito-911/MLT-project

## 1. Introducción

Los modelos generativos fueron formalizados como un juego de suma cero entre dos jugadores (generador y discriminador) [1]. El generador consiste en transformar un vector latente a una imagen con distribución similar a los datos observados. El discriminador simplemente es un clasificador que se encarga de analizar si una imagen de entrada es real o falsa. En [2] se extiende la idea de GANs a redes adversarias generativas condicionadas (cGANs). Esto quiere decir que el modelo generativo, además de crear una nueva imagen, es capaz de manipular el tipo de imagen que quiere crear dado un parametro.

La inspiración de este trabajo vino de la exploración de herramientas como Stable Diffusion y DALLE que, si bien no son las mismas arquitecturas (combinación de modelos de difusión y transformers), también tienen como uso la generación de tipo de imágenes específicas. Por este motivo, en este trabajo se exploró la capacidad de las cGAN en un contexto experimental. En la sección de metodología se dará más detalles, pero en términos generales, en este trabajo se realizó una recolección de datos, 3 arquitecturas de redes neuronales: una para hacer el modelo pre-entrenado(para realizar transfer-learning), otra para utilizar y extender el modelo preentrenado y otra arquitectura para entrenamiento desde cero. Posteriormente se presentan los resultados encontrados por el generador y se realiza la discusión y conclusión del trabajo. Como extensión adicional al trabajo, entrenamos un modelo simple GAN(pytorch) con un dataset de arte para poder observar como con una mayor muestra de pinturas se logran hacer nuevas piezas.

## 2. Trabajos relacionados

Las cGAN fueron anunciadas por primera vez en *Conditional Generative Adversarial Nets* [2], donde se trabajó con el conjunto de datos MNIST y se llegó a un modelo generador de digitos del 0 al 9, dependiendo de la entrada o condición ingresada. El principal cambio de paradigma de las cGAN con respecto a las GAN consistió en condicionar tanto al generador como al discriminador con información adicional, en este caso el label de los digitos en cuestión. Más adelante, la implementación de BigGAN [3] mostró el potencial real de las GANs condicionadas por clases (estado del arte de cGAN). Este modelo exploró en profundidad con una nueva arquitectura sobre el conjunto de datos ImageNET en resoluciones de 256x256 y 512x512, obteniéndose imagenes en alta resolución y con excelente resultados, de acuerdo a las métricas del FID (Fréchet Inception Distance) y el IS (Inception Socore) Ahora, para este trabajo fue

imposible hacer un entrenamiento de tal magnitud por recursos computacionales, sin embargo, utilizamos la idea de algunas capas (ResBlockUP) para nuestra arquitectura del generador.

Se encontraron bastantes trabajos en GANs condicionadas explorando la capacidad de estas redes con bastantes recursos computacionales e.g. [3, 4, 2]). Sin embargo, al menos desde nuestro conocimiento, no se ha explorado mucho modelos pre-entrenados con cGAN para **nuevas clases**. En [5] se explora una aproximación eficiente para el transfer-learning de las redes adversarias generativas utilizando propagación de conocimiento. En este se trabaja la transferencia desde el componente de Batch Normalization de antiguas clases hacia nuevas clases aprovechando las features antiguas. Si bien en este trabajo no adoptamos esa técnica, queda como extensión para exploración futura.

## 3. Metodología

### 3.1. Datasets

**CIFAR10:** Este conjunto de datos famoso contiene 10 clases y 6000 imágenes por cada clase. En su formato original las imágenes vienen en una escala de 32x32 pero se utilizó una reimplementación optimizada de estas imágenes en formato 64x64 [6]. CIFAR10 fue utilizado para hacer el pre-entrenamiento del cGAN que se utilizaría como modelo pre-entrenado.

**ArtUs:** Este conjunto de datos fue recolectado por nosotros. Utilizamos 3 clases de arte (abstracto-John Pollock-, geométrico e islámico). El motivo de escoger estos 3 tipos de arte fue por la capacidad de computo. Creemos que estos tipo de arte son más fáciles para aprender patrones en la red neuronal en un menor número de iteraciones que, por ejemplo, arte hiperrealista o cualquier tipo de arte que tenga mayor exploración de figuras, técnicas, etc. Cada clase contiene 100 imágenes, es decir, el total de datos de este dataset es de 300 imágenes. Por lo que estaremos trabajando con un conjunto de datos reducido.

**Abstract-art:** Este dataset fue descargado de Kaggle, donde encontramos 2782 imagenes de arte abstracto sin etiquetas que podremos usar para entrenar nuestro modelo GAN con una muestra de imagenes mas grande y igualmente que en CGAN con este tipo de imagenes se facilitara la creación de imagenes al ser de arte abstracto.

Figura 1: Muestra de imágenes ArtUs

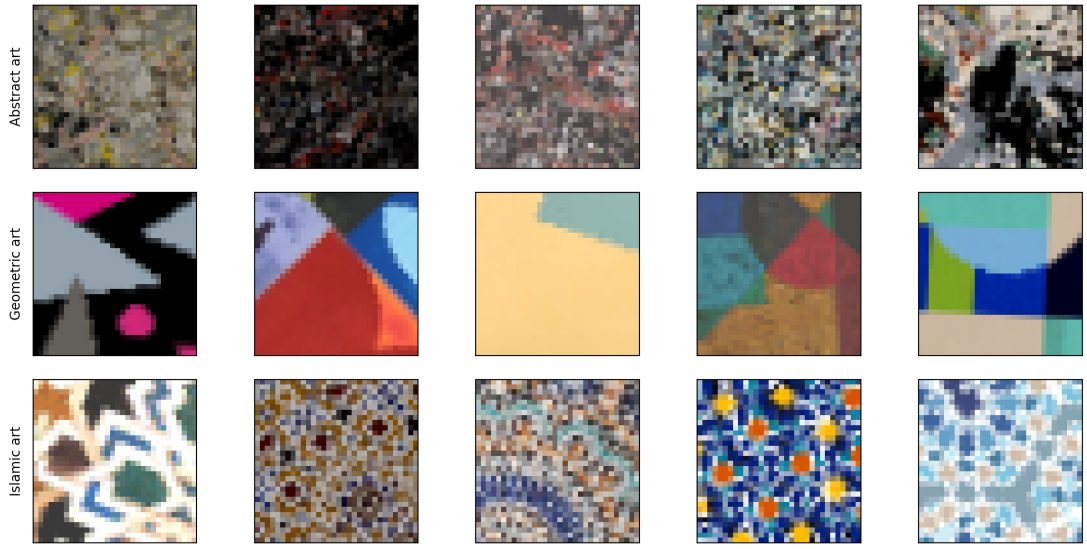
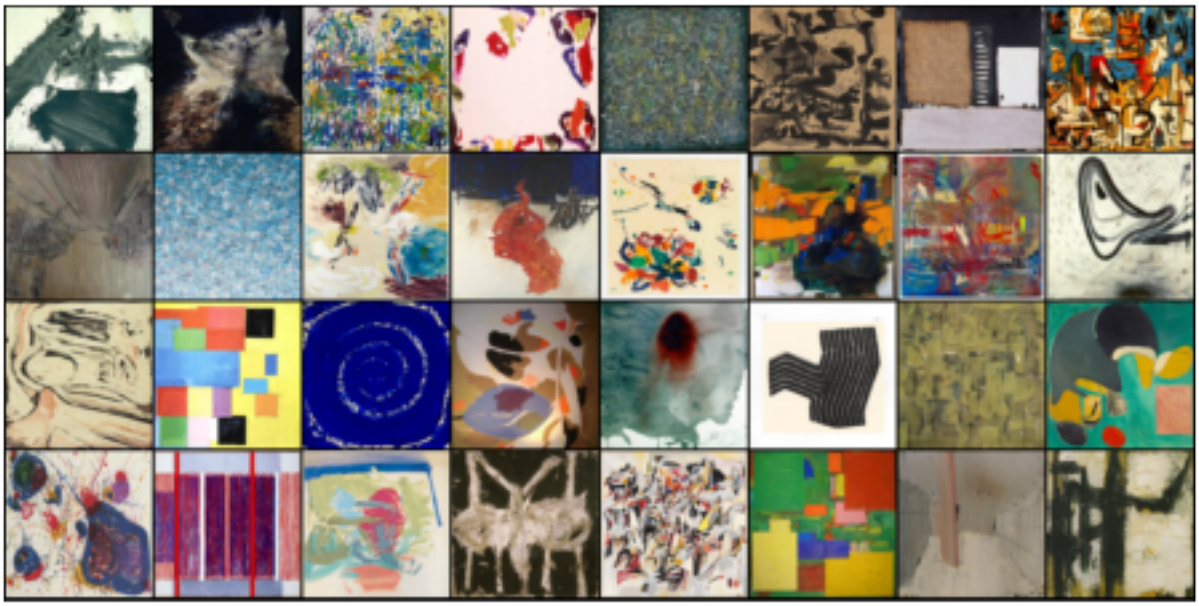


Figura 2: Muestra de imagenes Abstract-art



Para ArtUs haremos aumento de datos porque, dado que es arte, podemos crear nuevas pinturas que tendrán la misma clase (la técnica en la pintura no cambiará) con pequeños cambios en color, zoom, rotación, etc. Para esto, por cada imagen generamos 50 imágenes más. Es decir, para cada clase se tendrán 5000 imágenes por cada clase. Para un total de 15000 imágenes en ArtUs. Por otro lado, para abstract art tambien se hara un aumento de datos para que tengamos una base de 10000 imagenes y entrenar de mejor manera el GAN y para este caso haremos cambios de rotación y zoom

Como se observa en la Figura 1, en el proceso de aumento de datos se utilizó la herramienta de recortar. Esto implica que no se trabajarán con obras de arte completas. Esto podría representar un problema si el tipo de arte fuese hiperrealista, o de rostros. Sin embargo, como los 3 tipos de arte son de figuras, no se perjudicará tanto el diseño. El motivo de hacer un recorte de la imagen fue por los resultados experimentales que obtuvimos. Como las imágenes son tan grandes (>500 pixeles de ancho y altura) al reducirlas se convierten casi en ruido aleatorio (se pierden las formas). Esto se abordará en Discusión. Para el resto de capas de aumento de datos, se utilizó rotación de imágenes y aumento y reducción aleatoria de contraste y brillo. Al trabajar con arte de figuras, estas capas no representan algún peligro para la capacidad generativa del modelo.

El conjunto de datos ArtUs se almacena en HDFS (en el github lo encuentran como data_art.h5).

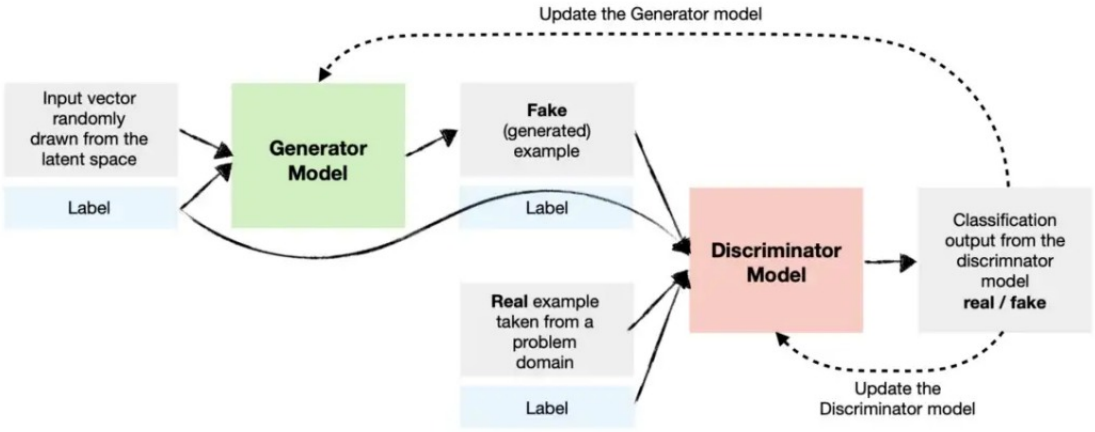
Para las GAN condicionadas, primero se entrena *subBigGAN* con el conjunto de datos de CIFAR10. Esto con el propósito de hacer transfer-learning del generador resultante al generador de *cArtGAN* sobre las capas de más bajo nivel. Posteriormente, se realiza el entrenamiento de *cArtGAN*

### 3.2. Arquitecturas:

Dentro de la Figura 3 se muestra la arquitectura general de una cGAN, donde como Input se tiene un vector de ruido aleatorio a partir de un espacio de características latentes y los label utilizados. Posteriormente se tiene el modelo generador condicionado por el label deseado y en el siguiente paso esta el modelo discriminador, también condicionado por el label con que se esta trabajando y que determinará si la imagen generada es falsa o no. El paso final consiste en la actualización de los parámetros y/o pesos del modelo discriminador y generador, repitiéndose el proceso hasta generar resultados satisfactorios.

***CGAN:*** Se trabajarán 3 tipos de arquitecturas para el CGAN:

Figura 3: Arquitectura de cArtGAN




**subBigGAN:** Esta arquitectura contiene componentes explorados de BigGAN [3] y se utilizará exclusivamente para pre-entrenar el generador con los datos de CIFAR10.

**cArtGAN:** Es una arquitectura condicional en el generador y discriminador a través de una entrada adicional (clase). Se puede observar en la Figura 4

**cTransferArtGAN:** Una arquitectura similar a cArtGAN, con la diferencia que se hizo transferencia de aprendizajes sobre las capas de bajo nivel del generador de 'subBigGAN' hacia el generador de 'cArtGAN'

Para comprobar la calidad de las arquitecturas, se realizó un entrenamiento con los datos de dígitos de MNIST sobre cArtGAN y se compararon con los resultados de [2, 4] , dando resultados similares.

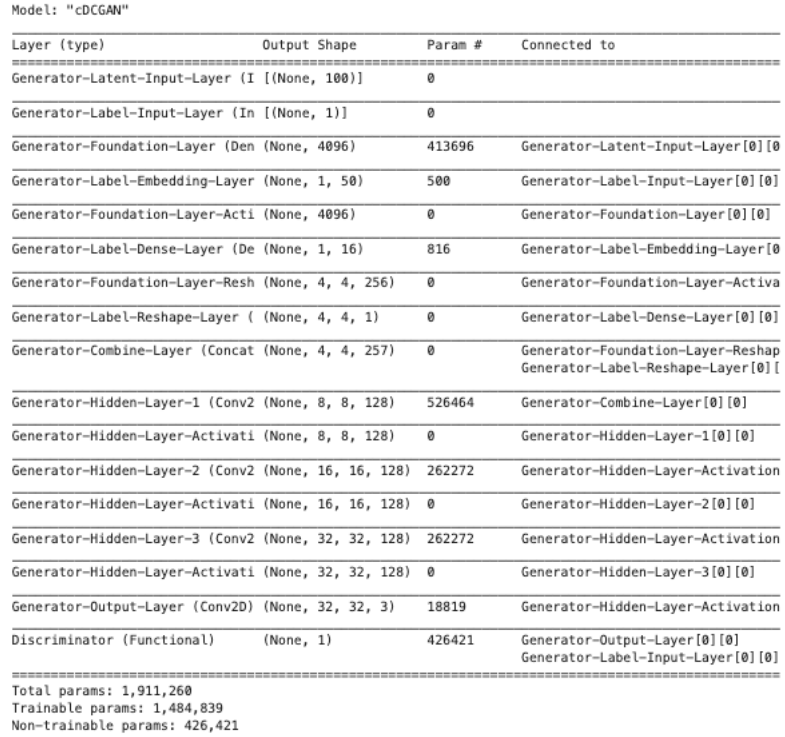
Figura 4: Arquitectura de discriminación cArtGAN

Model: "Discriminator"

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| Discriminator-Label-Input-Layer | [(None, 1)] | 0 | |
| Discriminator-Label-Embedding-L | (None, 1, 50) | 500 | Discriminator-Label-Input-Layer[0 |
| Discriminator-Label-Dense-Layer | (None, 1, 1024) | 52224 | Discriminator-Label-Embedding-Lay |
| Discriminator-Image-Input-Layer | [(None, 32, 32, 3)] | 0 | |
| Discriminator-Label-Reshape-Lay | (None, 32, 32, 1) | 0 | Discriminator-Label-Dense-Layer[0 |
| Discriminator-Combine-Layer (Co | (None, 32, 32, 4) | 0 | Discriminator-Image-Input-Layer[0<br>Discriminator-Label-Reshape-Layer |
| Discriminator-Hidden-Layer-1 (C | (None, 32, 32, 64) | 2368 | Discriminator-Combine-Layer[0][0] |
| Discriminator-Hidden-Layer-Acti | (None, 32, 32, 64) | 0 | Discriminator-Hidden-Layer-1[0][0 |
| Discriminator-Hidden-Layer-2 (C | (None, 16, 16, 128) | 73856 | Discriminator-Hidden-Layer-Activa |
| Discriminator-Hidden-Layer-Acti | (None, 16, 16, 128) | 0 | Discriminator-Hidden-Layer-2[0][0 |
| Discriminator-Hidden-Layer-3 (C | (None, 8, 8, 256) | 295168 | Discriminator-Hidden-Layer-Activa |
| Discriminator-Hidden-Layer-Acti | (None, 8, 8, 256) | 0 | Discriminator-Hidden-Layer-3[0][0 |
| Discriminator-MaxPool-Layer-2 ( | (None, 3, 3, 256) | 0 | Discriminator-Hidden-Layer-Activa |
| Discriminator-Flatten-Layer (Fl | (None, 2304) | 0 | Discriminator-MaxPool-Layer-2[0][ |
| Discriminator-Flatten-Layer-Dro | (None, 2304) | 0 | Discriminator-Flatten-Layer[0][0] |
| Discriminator-Output-Layer (Den | (None, 1) | 2305 | Discriminator-Flatten-Layer-Dropo |

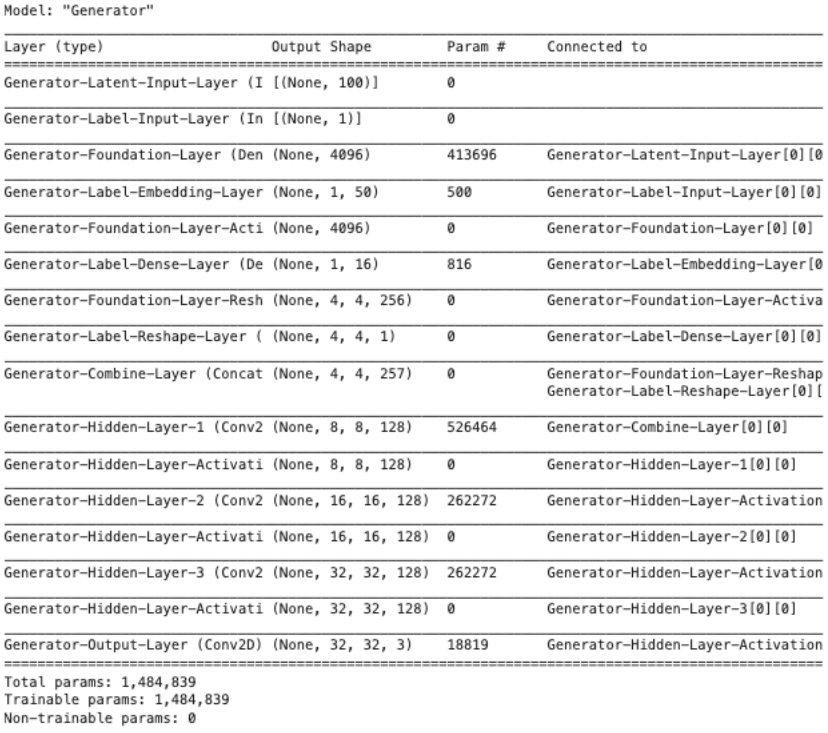
Total params: 426,421
Trainable params: 426,421
Non-trainable params: 0

Figura 5: Arquitectura cArtGAN



```
Model: "cDCGAN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Generator-Latent-Input-Layer (I [(None, 100)]          0
_________________________________________________________________
Generator-Label-Input-Layer (In [(None, 1)]            0
_________________________________________________________________
Generator-Foundation-Layer (Den (None, 4096)           413696
_________________________________________________________________
Generator-Label-Embedding-Layer (None, 1, 50)          500
_________________________________________________________________
Generator-Foundation-Layer-Acti (None, 4096)           0
_________________________________________________________________
Generator-Label-Dense-Layer (De (None, 1, 16)          816
_________________________________________________________________
Generator-Foundation-Layer-Resh (None, 4, 4, 256)      0
_________________________________________________________________
Generator-Label-Reshape-Layer ( (None, 4, 4, 1)        0
_________________________________________________________________
Generator-Combine-Layer (Concat (None, 4, 4, 257)      0
_________________________________________________________________
Generator-Hidden-Layer-1 (Conv2 (None, 8, 8, 128)      526464
_________________________________________________________________
Generator-Hidden-Layer-Activati (None, 8, 8, 128)      0
_________________________________________________________________
Generator-Hidden-Layer-2 (Conv2 (None, 16, 16, 128)    262272
_________________________________________________________________
Generator-Hidden-Layer-Activati (None, 16, 16, 128)    0
_________________________________________________________________
Generator-Hidden-Layer-3 (Conv2 (None, 32, 32, 128)    262272
_________________________________________________________________
Generator-Hidden-Layer-Activati (None, 32, 32, 128)    0
_________________________________________________________________
Generator-Output-Layer (Conv2D) (None, 32, 32, 3)      18819
_________________________________________________________________
Discriminator (Functional)   (None, 1)                 426421
=================================================================
Total params: 1,911,260
Trainable params: 1,484,839
Non-trainable params: 426,421
```

Figura 6: Arquitectura Generador cArtGAN



```
Model: "Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Generator-Latent-Input-Layer (I [(None, 100)]          0
_________________________________________________________________
Generator-Label-Input-Layer (In [(None, 1)]            0
_________________________________________________________________
Generator-Foundation-Layer (Den (None, 4096)           413696
_________________________________________________________________
Generator-Label-Embedding-Layer (None, 1, 50)          500
_________________________________________________________________
Generator-Foundation-Layer-Acti (None, 4096)           0
_________________________________________________________________
Generator-Label-Dense-Layer (De (None, 1, 16)          816
_________________________________________________________________
Generator-Foundation-Layer-Resh (None, 4, 4, 256)      0
_________________________________________________________________
Generator-Label-Reshape-Layer ( (None, 4, 4, 1)        0
_________________________________________________________________
Generator-Combine-Layer (Concat (None, 4, 4, 257)      0
_________________________________________________________________
Generator-Hidden-Layer-1 (Conv2 (None, 8, 8, 128)      526464
_________________________________________________________________
Generator-Hidden-Layer-Activati (None, 8, 8, 128)      0
_________________________________________________________________
Generator-Hidden-Layer-2 (Conv2 (None, 16, 16, 128)    262272
_________________________________________________________________
Generator-Hidden-Layer-Activati (None, 16, 16, 128)    0
_________________________________________________________________
Generator-Hidden-Layer-3 (Conv2 (None, 32, 32, 128)    262272
_________________________________________________________________
Generator-Hidden-Layer-Activati (None, 32, 32, 128)    0
_________________________________________________________________
Generator-Output-Layer (Conv2D) (None, 32, 32, 3)      18819
=================================================================
Total params: 1,484,839
Trainable params: 1,484,839
Non-trainable params: 0
```

***GAN***:

**Discriminador:** Esta arquitectura se encargará de calcular la probabilidad de que un dato sea real (de nuestro dataset) y no creado por el generador, lo que nos permitira ir iterando para que el discriminador genere mejores imagenes.

Figura 7: Arquitectura Discriminador GAN

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 64, 64]           3,072
       BatchNorm2d-2           [-1, 64, 64, 64]             128
         LeakyReLU-3           [-1, 64, 64, 64]               0
            Conv2d-4          [-1, 128, 32, 32]         131,072
       BatchNorm2d-5          [-1, 128, 32, 32]             256
         LeakyReLU-6          [-1, 128, 32, 32]               0
            Conv2d-7          [-1, 256, 16, 16]         524,288
       BatchNorm2d-8          [-1, 256, 16, 16]             512
         LeakyReLU-9          [-1, 256, 16, 16]               0
           Conv2d-10            [-1, 512, 8, 8]       2,097,152
      BatchNorm2d-11            [-1, 512, 8, 8]           1,024
        LeakyReLU-12            [-1, 512, 8, 8]               0
           Conv2d-13           [-1, 1024, 4, 4]       8,388,608
      BatchNorm2d-14           [-1, 1024, 4, 4]           2,048
        LeakyReLU-15           [-1, 1024, 4, 4]               0
           Conv2d-16              [-1, 1, 1, 1]          16,384
          Flatten-17                    [-1, 1]               0
          Sigmoid-18                    [-1, 1]               0
================================================================
Total params: 11,164,544
Trainable params: 11,164,544
Non-trainable params: 0
```

**Generador:** El objetivo de esta arquitectura es la de generar las imagenes al capturar la distribución del dataset de entrenamiento y generar datos que sigan esa distribución.

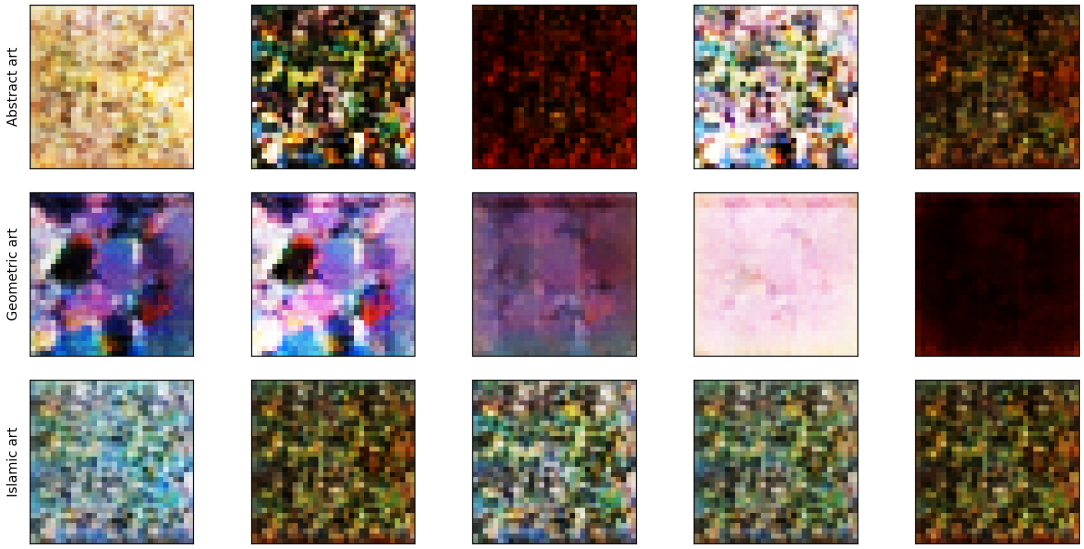
Figura 8: Arquitectura Generador GAN

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1           [-1, 1024, 4, 4]       2,097,152
       BatchNorm2d-2           [-1, 1024, 4, 4]           2,048
              ReLU-3           [-1, 1024, 4, 4]               0
   ConvTranspose2d-4            [-1, 512, 8, 8]       8,388,608
       BatchNorm2d-5            [-1, 512, 8, 8]           1,024
              ReLU-6            [-1, 512, 8, 8]               0
   ConvTranspose2d-7          [-1, 256, 16, 16]       2,097,152
       BatchNorm2d-8          [-1, 256, 16, 16]             512
              ReLU-9          [-1, 256, 16, 16]               0
  ConvTranspose2d-10          [-1, 128, 32, 32]         524,288
      BatchNorm2d-11          [-1, 128, 32, 32]             256
             ReLU-12          [-1, 128, 32, 32]               0
  ConvTranspose2d-13           [-1, 64, 64, 64]         131,072
      BatchNorm2d-14           [-1, 64, 64, 64]             128
             ReLU-15           [-1, 64, 64, 64]               0
  ConvTranspose2d-16          [-1, 3, 128, 128]           3,072
             Tanh-17          [-1, 3, 128, 128]               0
================================================================
Total params: 13,245,312
Trainable params: 13,245,312
Non-trainable params: 0
```

## 4. Evaluación

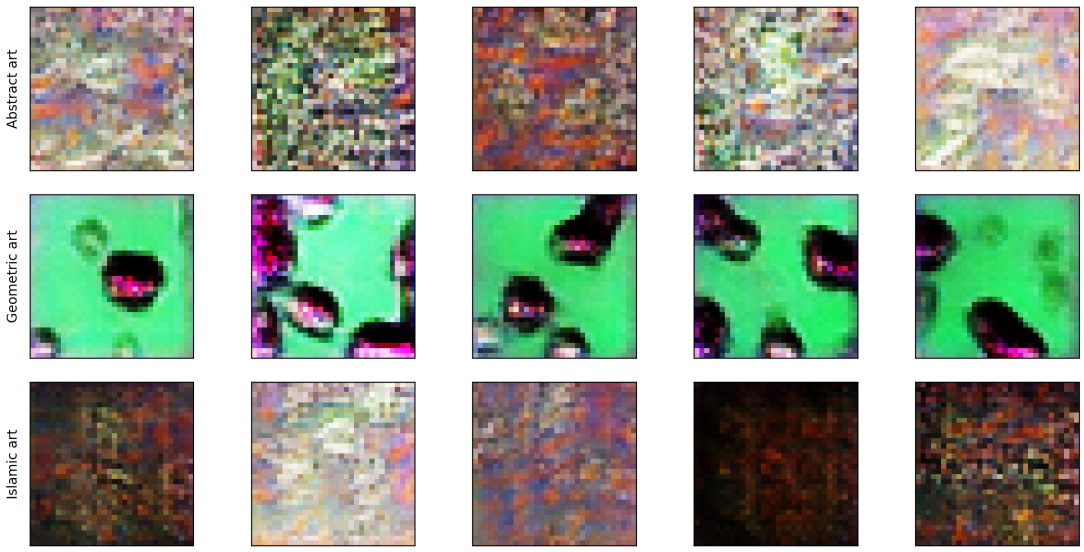
Los resultados en la Figura 9 hacen referencia al modelo entrenado desde 0 con las 15000 imágenes y 100 épocas. Si se comparan estos resultados con la imagen de muestra en la Figura 1 se observan diferencias notables en los resultados esperados para arte geométrico, pero una buena similitud en cuanto a arte abstracto. Para el arte islámico la similitud no es mucha, siendo los resultados más parecidos a arte abstracto que a islámico.

Figura 9: Resultados del generador de 'cArtGAN'



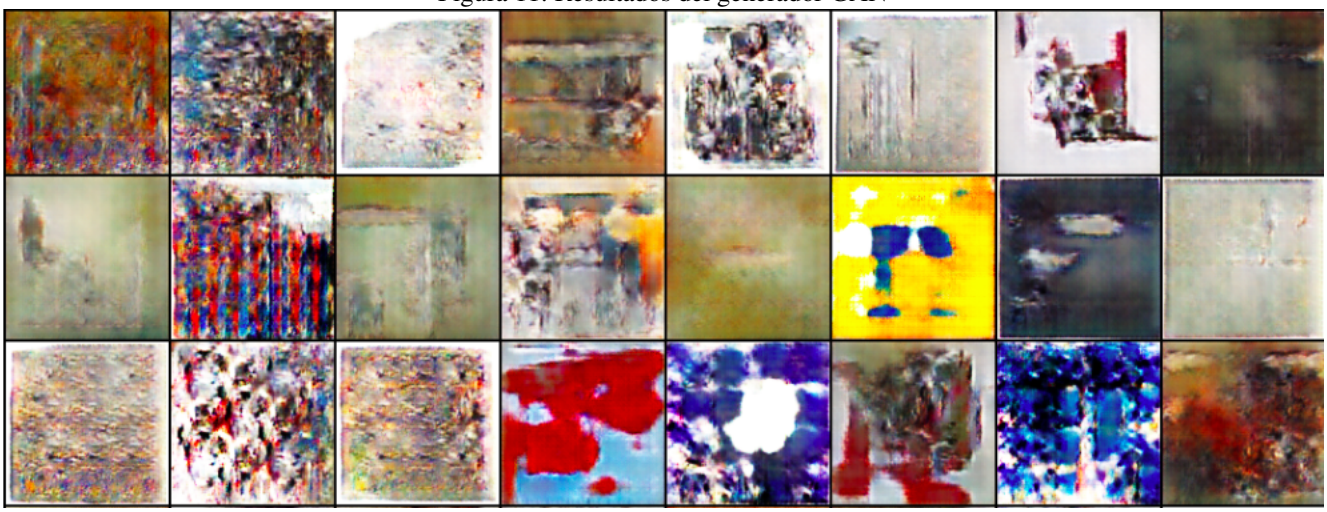

Ahora, los resultados en la Figura 10 hacen referencia al modelo entrenado con transfer-learning del dataset de CIFAR-10 usando 100 épocas. Para este caso los resultados tanto en arte abstracto como en arte islámico son muy similares entre sí y a su vez similares a los observados en la Figura 5. Para el caso de arte geométrico los resultados aunque se diferencian de los obtenidos en arte islámico y abstracto, aún se distancian mucho de los esperados y observados en la Figura 1.

Figura 10: Resultados del generador de 'cTransferArtGAN'




Ahora, en la Figura 11 se muestra un conjunto de 24 imagenes generadas a partir de la GAN con el conjunto de entrenamiento de Abstract-art usando 200 épocas. Para este caso es claro y se pueden identificar dichas imagenes como de arte abstracto.

Figura 11: Resultados del generador GAN



## 5. Discusión

Si bien se observan algunas diferencias entre los 3 tipos de arte escogidos, podemos apreciar que no se logran resultados satisfactorios. Uno de los grandes retos para este trabajo fue lograr resultados diferenciales entre los tipos de arte (islámico, geométrico, abstracto) bajo recursos limitados, lográndose probar solo hasta 100 épocas de entrenamiento con una duración de entrenamiento de hasta 6 horas para el pre entrenamiento de CIFAR10. Fuera de los recursos, uno de los mayores problemas encontrados fue en la construcción del conjunto de datos de arte, pues para el conjunto de ArtUs, los datos se descargaron usando web scraping a partir de diferentes sitios web.

El problema del conjunto de imágenes de ArtUs, como se evidencia en la Figura 1, radica en falta de pinturas completas. La gran mayoría de imágenes en internet tienen un tamaño >500 pixeles de ancho y alto. Esto, como consecuencia, hace que al reducir una imagen se pierdan todos los patrones de la pintura y se comporte como ruido. Por esta razón, como lo mencionamos en metodología, tuvimos que hacer recortes de las pinturas. Esto conllevó a que se cortarán también formas y patrones que eran relevantes para el mismo tipo de arte.

## 6. Conclusión y trabajo futuro

Si bien se reconoce el valor que tienen las cGAN en los modelos generativos, lográndose direccionar este proceso de generación de datos a través de la especificación de ciertos labels o características, se encuentran problemas de costo computacional para lograr resultados aceptables, resaltándose el hecho de que para la arquitectura GAN construida de arte abstracto tomó cerca de 35 segundos por cada época de entranamiento, lográndose resultados bastante aceptables, mientras que en el caso más optimista de las cGAN este tiempo fue de cerca de 70 segundos por época con resultados mucho menos satisfactorios.

Se valora también la importancia de tener modelos pre entrenados, pues aunque con resultados no muy buenos, la implementación del CIFAR10 para el pre entrenamiento del cGAN arrojó mejores resultados (especialmente para arte geométrico) que la construcción de esta red desde cero.

Se plantea como trabajo futuro a este documento el entrenamiento de las cGAN con mayores recursos computacionales, teniendo como objetivo replicar el ejercicio con al menos 500 epocas y tener un dataset de imagenes mas grandes y validar como se comporta el generador ante este caso. Así mismo se plantea la evaluación de la calidad de las imagenes generadas a partir de métricas empleadas para las GAN como lo son la FID (distancia de inicio de Fréchet) o el IS (inception score).

## Referencias

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[2] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[4] Ping Yu, Kaitao Song, and Jianfeng Lu. Generating adversarial examples with conditional generative adversarial net. *arXiv preprint arXiv:1903.07282*, 2019.

[5] Mohamad Shahbazi, Zhiwu Huang, Danda Pani Paudel, Ajad Chhatkuli, and Luc Van Gool. Efficient conditional gan transfer with knowledge propagation across classes. *arXiv preprint arXiv:2102.06696*, 2021.

[6] Joao Paulo Schwarz Schuler. Cifar-10 64x64 resized via cai super resolution. `https://www.kaggle.com/datasets/joaopauloschuler/cifar10-64x64-resized-via-cai-super-resolution`, 2020.