

# **Diseño de una red generativa antagónica para el mejoramiento de la resolución de imágenes.**

**John Gabriel Muñoz Cruz**

Máster en ingeniería computacional y matemática  
Inteligencia artificial

**Samir Kanaan Izquierdo**  
**Carles Ventura Royo**

21 de junio del 2019



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

## FICHA DEL TRABAJO FINAL

|                                    |   |
|------------------------------------|---|
| <b>Título del trabajo:</b>         | <i>Diseño de una red generativa antagónica para el mejoramiento de la resolución de imágenes.</i> |
| <b>Nombre del autor:</b>           | <i>John Muñoz Cruz</i>  |
| <b>Nombre del consultor/a:</b>     | <i>Samir Kanaan Izquierdo</i>   |
| <b>Nombre del PRA:</b>             | <i>Carles Ventura Royo</i>  |
| <b>Fecha de entrega (mm/aaaa):</b> | <i>06/2019</i>  |
| <b>Titulación:</b>                 | <i>Máster en ingeniería computacional y matemática</i>  |
| <b>Área del Trabajo Final:</b>     | <i>Inteligencia artificial</i>  |
| <b>Idioma del trabajo:</b>         | <i>Español</i>  |
| <b>Palabras clave</b>              | <i>Redes convolucionales adversas, mejora de resolución en imágenes.</i>                          |

**Resumen del Trabajo (máximo 250 palabras):** *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El trabajo consiste en diseñar una red generativa antagónica (GAN), la cual aumenta la resolución de una imagen, en donde no se conoce una mejor resolución de esta.

Este desarrollo se aplica en situaciones donde se hace necesario crear información que no existe, en este caso es con imágenes donde su finalidad es aumentar su resolución a partir de una red neuronal pre-entrenada, lo cual es practico en situaciones donde se necesite identificar algo y por la resolución de la imagen no se permita.

Para el desarrollo del trabajo inicialmente se entrena una red convolucional para extraer las características de un dataset, y posteriormente utilizando esta información se diseña una red antagónica que es la unión de una red que genera información y otra que lo discrimina obteniendo así la mejor versión de la imagen.

Como resultado de la investigación es evidente una mejora de las imágenes de baja resolución a alta resolución, la red diseñada permite obtener una mejor versión en un computador sin hardware especializado en cuanto memoria y GPU, sin embargo, es importante considerar el costo computacional para el desarrollo una red de este tipo; ya que si implica un dataset con un gran volumen de imágenes de alta resolución es necesario hacer uso de un clúster.

**Abstract (in English, 250 words or less):**

The work consists of creating an antagonistic generative network (GAN), which increases the resolution of an image, where a better resolution is not known.

This development applies in situations where it is necessary to create information that does not exist, in this In this case, the purpose is to increase the resolution of the images from a pre-trained neural network, which is practical in situations where you need to identify something and for the resolution of the image is not allowed.

For the development of the work a convolutional network is initially trained to extract the characteristics of a dataset, and later using this information an antagonistic network is designed that is the union of a network that generates information and another that discriminates it obtaining the best version of the picture.

As a result of the research it is evident an improvement of low resolution images to high resolution, the designed network allows obtaining a better version in a computer without specialized hardware in memory and GPU, however It is important to consider t the computational cost for developing this kind of network since implies a dataset with a large volume of high resolution images and the need to use a cluster

## Contenido

|  |    |
|--|----|
| 1. Introducción.....   | 2  |
| 1.1 Contexto y justificación del trabajo .....                               | 2  |
| 1.2 Justificación del TFG .....  | 2  |
| 1.3 Objetivos del trabajo.....   | 3  |
| 1.4 Enfoque y método seguido.....  | 4  |
| 1.5 Planificación del trabajo.....   | 5  |
| 1.5 Breve resumen de productos obtenidos .....                               | 7  |
| 1.6 Breve descripción de los otros capítulos de la memoria.....              | 7  |
| 2.Preparación de los datos para la red neuronal.....                         | 8  |
| 2.1 ¿Por qué interpolar?.....  | 8  |
| 2.2 Métricas de evaluación de una imagen .....                               | 9  |
| 2.3 Preparación del dataset.....   | 12 |
| 2.4 Preparación de las imágenes .....  | 14 |
| 3 Diseño de la GAN.....  | 16 |
| 3.1 Que son las redes neuronales y el aprendizaje profundo.....              | 16 |
| 3.2 Que son las GANS .....   | 17 |
| 3.3 Programación de una red neuronal en Keras .....                          | 18 |
| 3.4 Arquitecturas  extractoras de características previamente entrenadas ... | 20 |
| 3.5 Diseño de una red extractora de características .....                    | 24 |
| 3.6 Esquema de la GAN.....   | 30 |
| 3.7 Programación de la GAN.....  | 32 |
| 4. Resultados de la GAN .....  | 38 |
| 4.1 Resultados en general.....   | 38 |
| 4.2 Google colabatory .....  | 41 |
| 5. Conclusiones.....   | 44 |
| 6. Glosario .....  | 45 |
| 7.Bibliografía .....   | 46 |
| 8. Anexos .....  | 48 |
| 8.1 Librería MSE .....   | 48 |
| 8.2 Librería carga_ imágenes.....  | 48 |
| 8.3 Código red extractora de características .....                           | 49 |
| 8.4 Código GAN .....   | 52 |

## Lista de figuras

|                |  |    |
|----------------|--|----|
| Ilustración 1  | Ciclo de machine learning                              | 4  |
| Ilustración 2  | Fechas diagrama de Gantt                               | 6  |
| Ilustración 3  | Cronograma Diagrama de Gantt                           | 6  |
| Ilustración 4  | Características imágenes                               | 8  |
| Ilustración 5  | Imagen con resolución disminuida                       | 9  |
| Ilustración 6  | imagen sin interpolación                               | 9  |
| Ilustración 7  | Ejemplo dataset MINST                                  | 13 |
| Ilustración 8  | Dataset empleados                                      | 13 |
| Ilustración 9  | función carga_imagenes                                 | 14 |
| Ilustración 10 | Red neuronal mas simple                                | 16 |
| Ilustración 11 | Función de activación                                  | 17 |
| Ilustración 12 | estructura de una GAN                                  | 18 |
| Ilustración 13 | Proceso de aprendizaje de un red                       | 19 |
| Ilustración 14 | Resumen Keras VGG16                                    | 22 |
| Ilustración 15 | Características CNN                                    | 24 |
| Ilustración 16 | Convolución en una imagen                              | 25 |
| Ilustración 17 | Función de activación ReLu                             | 25 |
| Ilustración 18 | Resumen SRCNN  | 27 |
| Ilustración 19 | Preparación datos SRCNN                                | 28 |
| Ilustración 20 | Resultados SRCNN                                       | 30 |
| Ilustración 21 | Estructura GAN   | 30 |
| Ilustración 22 | Etapas de entrenamiento GAN                            | 31 |
| Ilustración 23 | Funcion LeayRelu                                       | 33 |
| Ilustración 24 | Etapas de Pooling                                      | 34 |
| Ilustración 25 | Funcion Sigmoid  | 34 |
| Ilustración 26 | Resumen del generador                                  | 35 |
| Ilustración 27 | Resumen del discriminador                              | 35 |
| Ilustración 28 | Celdas notebook de Jupyter                             | 41 |
| Ilustración 29 | Configuración entorno de ejecución Google Colaboratory | 42 |
| Ilustración 30 | Autorización Google Drive                              | 42 |
| Ilustración 31 | Archivos de Google Drive                               | 43 |
| Ilustración 32 | Resultados Google Colaboratory                         | 43 |

# 1. Introducción

## 1.1 Contexto y justificación del trabajo

### 1.1 Descripción general

Mejorar la calidad de una imagen mediante las técnicas actuales es una labor compleja porque requiere añadir información a la imagen que no existe, es complejo porque a pesar de usar modelos matemáticos y la vectorización de imágenes los resultados no siempre son los esperados por la variedad de elementos que pueden existir en las imágenes, como puede ser en las fotos donde varía el color, la intensidad, la rotación entre otras características.

por otra parte, el aprendizaje profundo basado las redes neuronales con capas ocultas está presente en la visión por ordenador hace algunos años [1], en donde una capa reconoce características sencillas como bordes, la siguientes capa elementos como esquinas, el siguiente elemento geométricos y así hasta obtener características complejas de la imagen, en particular existen las CNN en donde los datos de entrada solo son imágenes y su función es la clasificación de estas.

En el 2014 se presentan las redes neuronales antagónicas, las cuales consisten en la unión de 2 redes neuronales que compiten entre ellas mismas para aprender y evolucionar por sí solas, donde una red evalúa los resultados de la otra red mejorando los resultados iteración tras iteración.

Así partir de una imagen en alta resolución se crea una imagen en baja resolución la cual ingresa a una red generativa antagónica GAN que genera una nueva versión en alta resolución. Una vez se tienen las dos versiones de la imagen se evalúan con distintas métricas tratando así que los resultados sean parecidos a la imagen original, diseñando un sistema capaz de mejorar la resolución de una imagen mediante el aprendizaje autónomo.

### 1.2 Justificación del TFG

La inteligencia artificial ha sido un tema de incidencia tecnológica en los últimos años que ha creado una nueva forma de idear soluciones y procesos, ya no es utópico el diseño de artefactos que antes solamente parecían ciencia ficción, como el diseño de robots con el aprendizaje adaptativo [1] donde se espera que un robot con determinadas características realice un aprendizaje similar al de un recién nacido adaptándose a las necesidades del entorno, máquinas que les ganen a los humanos en diversos juegos donde estas máquinas aprenden diversas técnicas y estrategias, coches autónomos, reconocimiento de patrones entre muchas otras aplicaciones. Incluso se ha proyectado [2]

un futuro en donde la IA es un protagonista que marca un cambio en la forma de como el ser humano se comporta actualmente no solo en el sentido tecnológico si no social.

Independientemente del discurso ético de crear artefactos inteligentes que puedan imitar procesos del ser humano son hechos ciertos que la llegada de la IA es inminente y que la tecnología bien aplicada puede solucionar grandes necesidades de la humanidad. Una de las ramas de la inteligencia artificial es el machine learning que utiliza el aprendizaje autónomo para hacer que las maquinas se doten de la capacidad de aprendizaje lo cual es un tema de gran relevancia en particular cuando es usan redes neuronales [1] en lo que se denomina aprendizaje profundo.

El cual consiste en imitar por medio de capas de redes neuronales el cerebro humano y así procesos de aprendizajes lo que es muy relevante cuando se traslada a máquinas en múltiples escenarios que sin lugar a duda es un tema intrigante, pertinente y motivante para estudiarlo porque es muy probable que la humanidad volqué más sus necesidades hacia este tema.

### 1.3 Objetivos del trabajo

#### Objetivo principal

- Diseñar una red generativa antagónica para aumentar la resolución de una imagen a partir de una versión de baja resolución usando Keras

#### Objetivos específicos

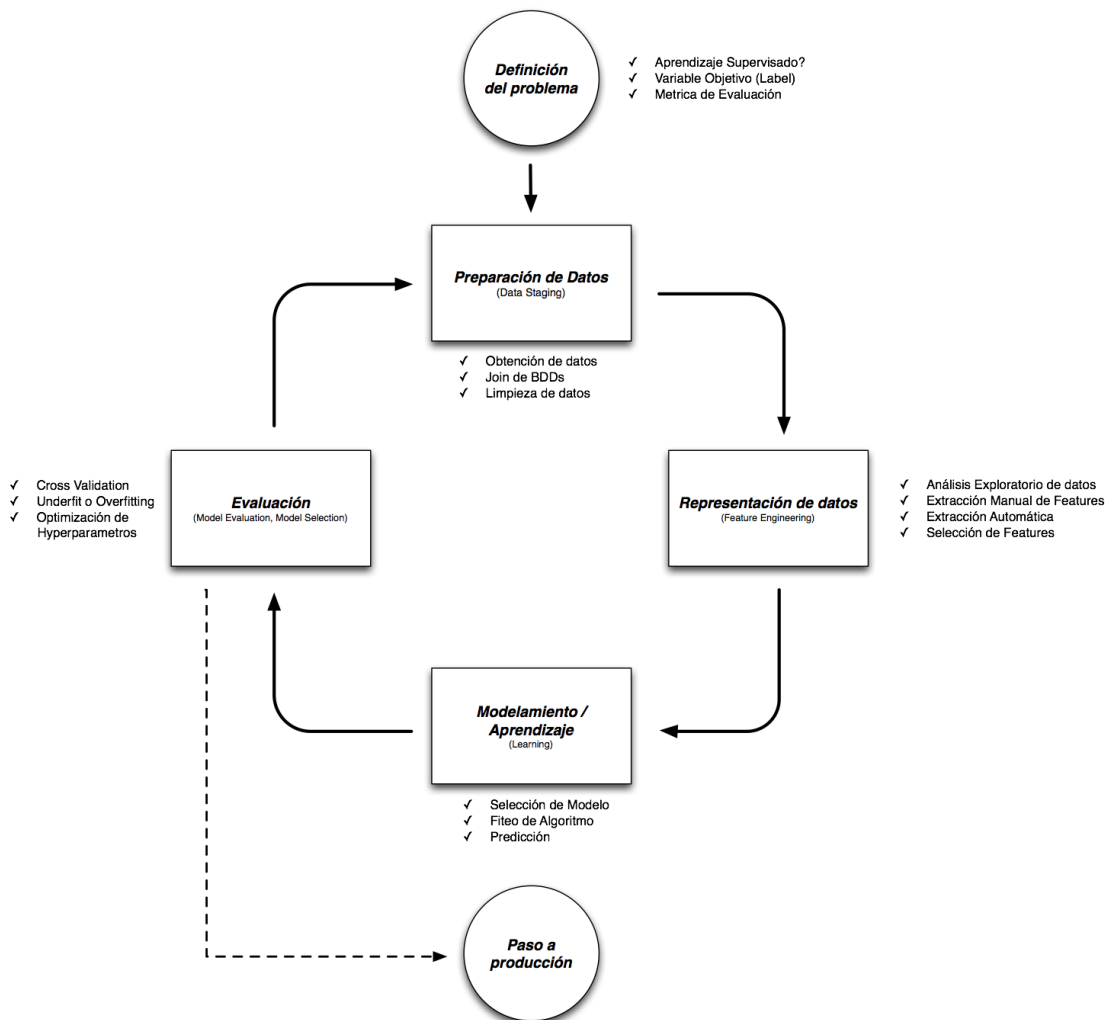
- Recolectar información del estado del arte de las GAN'S, la programación del diseño y entrenamiento en Keras, así como las soluciones existentes para mejorar la calidad de una imagen.
- Obtener un Dataset con imágenes en alta resolución y disminuir la calidad de las imágenes para obtener un conjunto de datos para entrenar la red.
- Diseñar una red generativa antagónica entrenándola mediante un dataset de imágenes, donde a partir de una imagen de baja resolución se produzca una imagen de alta resolución.
- Comprobar la calidad de resultados de las imágenes originales con las imágenes entregadas por la red generativa antagónica.



## 1.4 Enfoque y método seguido

La estrategia consiste en dividir el trabajo en hitos de forma secuencial según las fechas propuestas en el plan docente de la asignatura. Inicialmente se hace un estudio del estado de arte de las redes neuronales y su programación para posteriormente seguir a la fase de diseño.

Se hace un ciclo para el diseño de la red generativa antagónica basándose en un ciclo de machine learning (ilustración 1) que es una metodología [16] que propone ir desde lo más sencillo hasta lo más complejo iteración tras iteración, así que a continuación se hace una breve descripción de cada proceso.



**Ilustración 1 Ciclo de machine learning**

La definición del problema consiste en generar imágenes a partir de una versión de baja resolución, para representar los datos la primera iteración es con pocas imágenes a blanco y negro, el modelamiento se hace usando las interpolaciones más comunes para generar imágenes en alta resolución y se diseñan las respectivas métricas de evaluación para comparar la imagen de alta y baja resolución.

En el segundo ciclo se procede diseñar una red para extraer las características de las imágenes del dataset para el modelamiento de los datos, se obtienen las métricas de evaluación de las imágenes, así como de la red neuronal y se procede a volver a generar el ciclo para la creación de la red generativa antagónica.


La cual es una metodología apropiada porque es un trabajo individual en donde se puede verificar etapa a etapa el desarrollo del trabajo de investigación, el enfoque de investigación es mixto porque requiere hacer una investigación cualitativa sobre las redes neuronales antagónicas en cuanto el diseño y una cuantitativa al momento de usar el dataset de imágenes y hacer la programación de los modelos para obtener los resultados esperados.

### 1.5 Planificación del trabajo

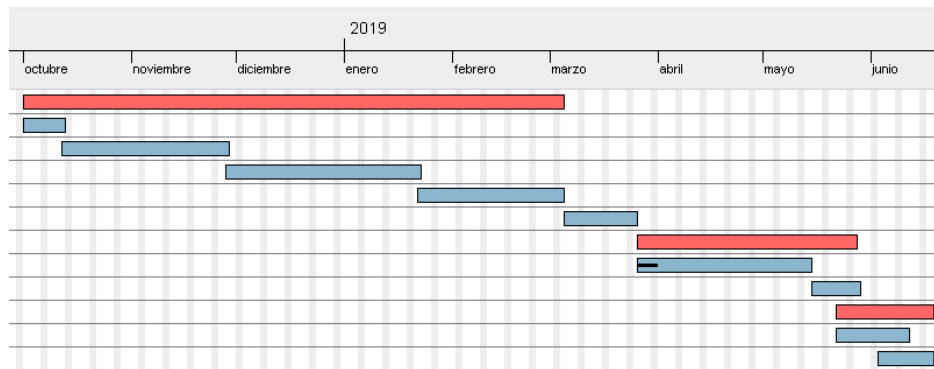
Después de las correcciones en el transcurso del proyecto la planeación se desarrolla con 3 hitos con 9 tareas, que son los siguientes:

- a. Preparación de recursos
  - Instalación entorno anaconda con Keras
  - Revisión aprendizaje profundo
  - Programación de redes neuronales en Keras
  - Estado de arte y programación de GAN'S
  - Preparación del dataset
- b. Diseño y entrenamiento GAN'S
  - Diseño GAN'S
  - Entrenamiento GAN'S
- c. Validación de resultados y posibles implementaciones.
  - Métodos de validación y extracción de resultados
  - Pruebas arquitecturas, funciones y Google colab

Esta planificación se detalla en el siguiente diagrama de Gantt en donde los Hitos se encuentran con color rojo y las tareas de color gris.

|  |                 |              |
|---|-----------------|--------------|
| Nombre  | Fecha de inicio | Fecha de fin |
| • Preparación recursos  | 1/10/18         | 4/03/19      |
| • Instalacion entorno Anaconda con Keras  | 1/10/18         | 12/10/18     |
| • Revisión aprendizaje profundo   | 12/10/18        | 28/11/18     |
| • Programación de redes neuronales en Keras                                       | 28/11/18        | 22/01/19     |
| • Estado de Arte y programación de GAN'S  | 22/01/19        | 4/03/19      |
| • Preparación del data set  | 5/03/19         | 25/03/19     |
| • Diseño y entrenamiento GAN'S  | 26/03/19        | 27/05/19     |
| • Diseño GAN'S  | 26/03/19        | 14/05/19     |
| • Entrenamiento GAN'S   | 15/05/19        | 28/05/19     |
| • Validación de resultados y posibles implementaciones                            | 22/05/19        | 18/06/19     |
| • Metodos de validación, extracción de resultados                                 | 22/05/19        | 11/06/19     |
| • Pruebas arquitecturas, funciones y Collaboratory                                | 3/06/19         | 18/06/19     |

**Ilustración 2 Fechas diagrama de Gantt**



**Ilustración 3 Cronograma Diagrama de Gantt**

A continuación, se enlistan los riesgos que tuvo la investigación para no concluirse con más características:

- Errores en las instalaciones, paquetes o librerías, así como falta de documentación que no permiten su desarrollo correcto.
- Capacidad de almacenamiento y procesamiento de las imágenes cuando se entrenen las redes neuronales.
- Tiempo del proyecto para poder probar otras implementaciones con otras características.
- Dificultad en la búsqueda de información sobre las GAN'S por ser relativamente recientes.

## 1.5 Breve resumen de productos obtenidos

Una vez culminado el proyecto se espera hacer entrega de los siguientes ítems:

- Memoria con los detalles de la investigación y los códigos de la programación de la red neuronal.
- Presentación virtual resumida sobre la investigación.
- Autoevaluación del proyecto.

## 1.6 Breve descripción de los otros capítulos de la memoria

La investigación inicia con unas precisiones de lo que es la resolución de una imagen, así como las principales características de estas, en que consiste el proceso de interpolación tradicional y las métricas utilizadas para medir que tanto se parece una imagen interpolada a una imagen original, porque así se va a evaluar el resultado de la red convolucional antagónica.

Posteriormente se seleccionan los dataset con los que se va a realizar la investigación, se crean las versiones en baja resolución a partir de la versión de alta resolución adecuando así la información para el ingreso de la red.

El siguiente capítulo consiste en una descripción de que es una red neuronal y que son las redes generativas antagónicas, así como el proceso del diseño en general del sistema, posteriormente se hace el diseño de la red extractora de características y un estudio sobre las redes ya pre-entrenadas.

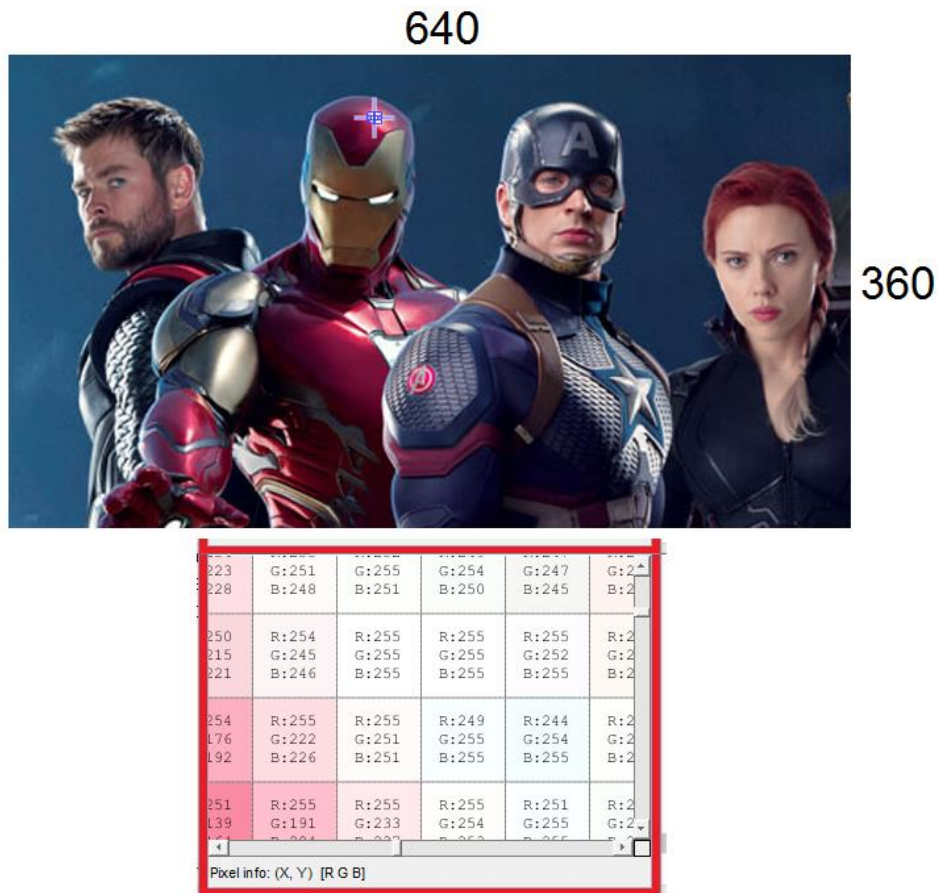
Una vez se obtienen las características de la imagen se procede a diseñar la red generativa antagónica. Cuando culmina el diseño se evalúan los resultados de las imágenes extraídas de los dataset con las respectivas métricas.

Un último capítulo consiste en exploración de funciones, modelos ya existentes o herramientas de implementación que haga más sencillo el uso de este tipo de redes.

## 2.Preparación de los datos para la red neuronal

### 2.1 ¿Por qué interpolar?

Una imagen es una representación visual de un objeto real o imaginario en un sistema computacional la cual se modela por una matriz que cuenta con un alto, ancho y una profundidad determinado por un elemento denominado píxel que depende del color, saturación y brillo de la imagen lo cual se puede evidenciar en la ilustración 4.



**Ilustración 4 Características imágenes**

El ancho y el alto está determinado por un número de píxeles se cual se llama resolución en el caso de la ilustración 4 la resolución es 640 \* 360, cuando se reduce la resolución de una imagen se reduce el tamaño, por ejemplo, si se reduce a una dimensión de 400\*200 el resultado es una imagen más pequeña como se puede visualizar en la ilustración 5.



**Ilustración 5 Imagen con resolución disminuida**

Sin embargo, cuando se desea volver a tener la imagen en el tamaño original es necesario usar diversos métodos de interpolación para evitar una imagen pixelada como en la ilustración 6 en donde solamente se aplicó una ampliación a la imagen de la ilustración 3 y no se añadió información a la imagen.



**Ilustración 6 imagen sin interpolación**

## 2.2 Métricas de evaluación de una imagen

Para poder medir la efectividad de añadir información a la imagen es necesario usar métricas de evaluación en donde la finalidad es medir que tan buenos son los resultados de comparar la imagen original con la imagen a la que se le redujo la dimensión y por distintos métodos volvió a tener la resolución original.

En el taller NTIRE [12] llevado en Utah dedicado al estudio de restauración y mejora de imagen se emplea la métrica MSE, PSNR y SSIM para comparar las imágenes.

La métrica MSE consiste en recorrer las dos imágenes, hacer la resta píxel a píxel, elevar al cuadrado y llevar la sumatoria de las diferencias, como se puede visualizar en la siguiente ecuación:

$$MSE = \frac{1}{mn} \sum_{y=1}^m \sum_{x=1}^n [s(x, y) - \tilde{s}(x, y)]^2$$

En donde m y n es el alto y ancho de la imagen, con esta medida se puede calcular el PSNR.

$$PSNR = 20 \log_{10} \left( \frac{256-1}{\sqrt{MSE}} \right)$$

El cual por sus características logarítmicas permite expresar mejor el resultado obtenido por MSE. Otra métrica es la similitud estructural SSIM la cual considera un grupo píxeles descomponiéndolo en luminancia, contraste y estructura, en donde su resultado es un dígito entre -1 y 1 en donde los resultados que se acercan a uno son las imágenes que se parecen más a las originales.

Así, en este trabajo se usarán la métrica PSNR (considerando que la métrica MSE está implícita) y SSIM para obtener un valor cuantificado de la calidad de la imagen reconstruida. La implementación de PSNR en Python se desarrolla con el siguiente método:

```
def psnr(img1, img2):  
    mse = numpy.mean( (img1 - img2) ** 2 )  
    PIXEL_MAX = 255.0  
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
```

Y la métrica SSIM se realiza usando la siguiente librería de Python

```
from skimage.measure import compare_ssim
```

La cual es una librería especializada en métricas [13], donde solo basta ejecutar el siguiente comando para obtener el valor:

```
valor = compare_ssim(original, interpolada, multichannel=True )
```

En donde haciendo uso del software GIMP se aumenta la imagen sin hacer ningún método de interpolación, interpolación lineal y cúbica que se basan en promediar los píxeles vecinos, así como NoHalo que es una máscara de enfoque especial para afinar los bordes de una imagen. En la tabla número 1, se puede hacer una comparación entre los diferentes métodos y las métricas aplicadas.

|                      |   |  |
|----------------------|---|--|
| Sin interpolación    |    | PSNR: 32.6315112010859db<br>SSIM: 0.8052376920766026 |
| Interpolación Lineal |   | PSNR: 32.851350164644db<br>SSIM: 0.8445471823164099  |
| Interpolación Cubica |  | PSNR: 32.697298394952db<br>SSIM: 0.8395484434759513  |
| NoHalo               |  | PSNR: 32.8444725913251db<br>SSIM: 0.8434087445327556 |

**Tabla 1 Comparación métodos de interpolación**

En donde se concluye que el mejor método de interpolación es la interpolación lineal, lo cual también se puede ver claramente en el fracción de la imagen en



donde si bien no se caracteriza por tener una relación como la original parece pixelada (con pixeles individuales); editores de texto como Microsoft Word permiten cambiar el tamaño de una imagen y para evitar perder pixeles e información usan técnicas como vectorizar las imágenes, que es una técnica donde a partir de la figura se hace un modelo matemático según su estructura y forma de tal manera que se puede escalar fácilmente cambiando su tamaño sin perder información.

Las dificultades ocurren cuando no se tiene la imagen original para cambiar el tamaño de la imagen y mejorar su resolución, siendo esta dificultad una de las principales causas de usar redes neuronales e inteligencia artificial.

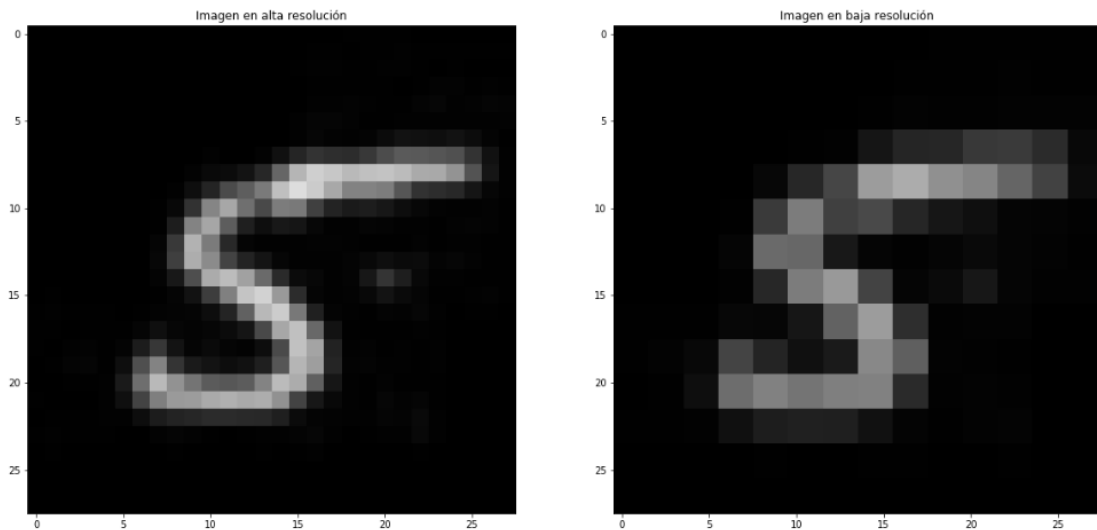
### 2.3 Preparación del dataset

En la búsqueda de dataset se encuentran multiplicidad de diferentes universidades y programas de investigación que los ofrecen, como CIFAR que es una agrupación de 60000 imágenes dividida en 10 diferentes clases (aviones, carros, gatos, perros, caballos, venados, ranas, barcos, automóviles y camiones) donde la finalidad es entrenar una red neuronal para que agrupe la imagen según unas de las clases mencionadas.

También se pueden extraer de Google imágenes con algunas características particulares, existen dataset presentados por Google, así como de páginas como [deeplearning.net](http://deeplearning.net), [analyticsvidhya](http://analyticsvidhya.com), [kaggle](http://kaggle.com), la universidad de Texas [17] que además presentan dataset de música, clasificación por imágenes naturales, imágenes de rostros, textos, sistemas de recomendación entre otros.

La entrada de la red neuronal son imágenes, sin embargo, por las necesidades del problema deben tener la imagen original, así como su versión en menor resolución, un dataset que ofrece estas características es DIV2K que es un conjunto de datos de gran tamaño que surgió de una competencia que consiste en el mismo planteamiento del problema que es aumentar la resolución de una imagen en baja resolución a partir de una versión de alta resolución. DIV2K que consta de grupos de 800 imágenes divididas en diferentes resoluciones para entrenar los sistemas, sin embargo, es una base de datos muy pesada donde cada imagen pesa 1,5 M.

Así que se inicialmente se diseña la red usando un dataset más liviano con imágenes con características sencillas y el más común es MINST que consiste en 600 imágenes blanco y negro de un tamaño  $28 \times 28$  de resolución, se escoge por su tamaño y por que todas las imágenes pertenecen a la misma clase que son números obtenidos de forma manual, la cual se puede ver en la imagen izquierda de la ilustración 7, como este dataset no contiene una versión con baja resolución se programa en Python una función donde el resultado es la imagen de la ilustración 7.



**Ilustración 7 Ejemplo dataset MINST**

Se escogen cuatro dataset considerando que en la mayoría de los casos los dataset tienen múltiples clases para hacer el proceso de identificación, en este caso solo se toma una clase a la cual se le reduce la resolución para que la red se entrena. Un segundo dataset en escoger una clase es el dataset CIFAR [18] que son imágenes a color de una resolución de  $32 \times 32$ , un dataset con tamaño invariante denominado SVHN [19] que consiste en dígitos a color y por último uno con las imágenes de la competencia DIV2K que tienen una resolución de  $640 \times 468$  para comprobar la eficiencia de la red con diferentes tipos de datos. En la ilustración 8 se puede observar una imagen de cada dataset empleado.



**MINST**



**CIFAR**



**DIV2K**

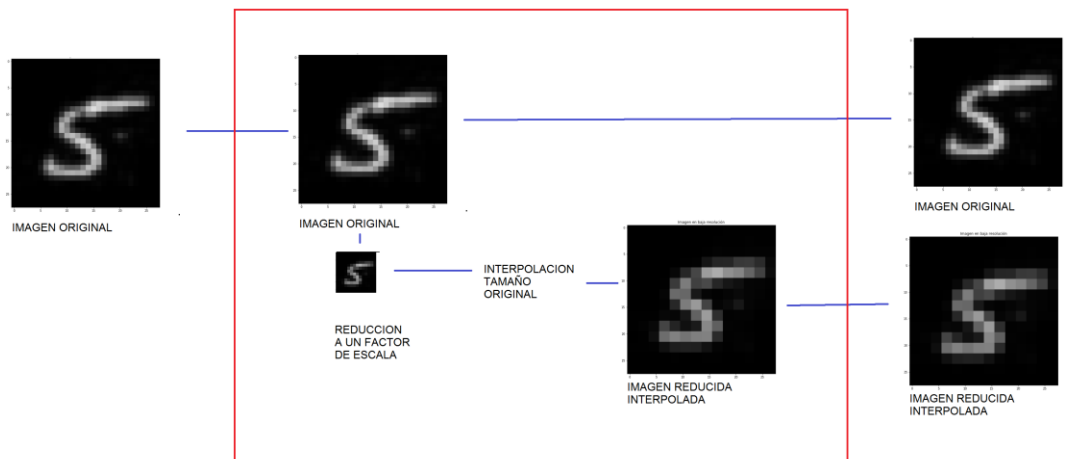


**SVHN**

**Ilustración 8 Dataset empleados**

## 2.4 Preparación de las imágenes

Una vez se seleccionan los dataset, se programa una función para inicialmente extraer las imágenes del directorio, hacer uniforme el tamaño de las imágenes, ya que en los dataset caltech y svhn las imágenes tienen diferentes tamaños y según una escala de reducción dimensionar la imagen a una baja resolución aplicando posteriormente una ampliación al tamaño original, la función se denomina **carga\_imagenes** y un proceso grafico de la imagen se puede observar en la ilustración 9.



**Ilustración 9 función carga imágenes**

Para obtener las imágenes del dataset, se utiliza el comando **glob** que obtiene el acceso a todas las imágenes en una lista, posteriormente se selecciona aleatoriamente un numero de imágenes definidas por el usuario, así como inicializar la lista que contiene las imágenes en alta y baja resolución, lo cual se implementa con el siguiente código:

```
ruta = glob('./datasets/%s/*' % (dataset_name))
conjunto_imagenes = np.random.choice(ruta, size=n_imagenes)
imgs_hr = []
imgs_lr = []
```

Una vez inicializadas las listas, se recorre la matriz con las rutas de las imágenes cargándolas como variables con el comando **imread** y se modifica la imagen con el comando **imresize**, la imagen original cambia de tamaño al introducido por el usuario y la imagen en baja resolución se construye dividiendo el tamaño de la imagen original por el factor de escala donde posteriormente se agregan las imágenes en alta y baja definición a sus respectivas listas con el comando **append**.

Por último se convierte la lista en matriz con el comando **np.array** para obtener las matrices con las que se va a procesar la información. A continuación, se presenta el código implementado:

```
for i in conjunto_imagenes:
    img = scipy.misc.imread(i)
```

```

img_hr = scipy.misc.imresize(img, img_res)
nX=img_hr.shape[0]/reduccion
nY=img_hr.shape[1]/reduccion

img_lr = scipy.misc.imresize(img, (int(nX), int(nY)))
img_lr= cv2.resize(img_lr,img_res, interpolation = cv2.INTER_NEAREST )
img_lr = scipy.misc.imresize(img, img_res)

imgs_hr.append(img_hr)
imgs_lr.append(img_lr)

imgs_hr = np.array(imgs_hr)
imgs_lr = np.array(imgs_lr)

```

Para comprobar el método con una imagen como la de la ilustración7, se utiliza la biblioteca matplotlib.pyplot con el método figure y subplot para visualizar las dos imágenes, se implementa de la siguiente manera:

```

plt.figure(figsize=(20,10)) #tamaño de las imagenes
plt.subplot(121)
plt.title('Imagen en alta resolución') # Ponemos un título
plt.imshow(imgs_hr[0,:,:,:])
plt.subplot(122)
img=scipy.misc.imread(ruta[1], mode='RGB')
plt.title('Imagen en baja resolución') # Ponemos un título
plt.imshow(imgs_lr[0,:,:,:])

```

También se guarda la imagen en el directorio para futuras revisiones y usando Pillow que es una librería de Python se procede a mostrar la imagen en el visor de imágenes del sistema operativo para que sea más rápida la revisión, así como la verificación.

```

plt.savefig("HRtoLR")
from PIL import Image
imagen = Image.open("HRtoLR.png")
imagen.show()

```

Para culminar de programar el método se extrae el tamaño de la imagen y este valor retorna junto a las matrices con la información de las imágenes lo cual se implementa de la siguiente manera:

```

size_lr=[imgs_lr.shape[1], imgs_lr.shape[2],3]
return imgs_hr[:,:,:,:1], imgs_lr[:,:,:,:1] , size_lr , size_hr

```

Una vez se programa el método que prepara las imágenes del dataset, la única consideración previa que deben tener las imágenes para el ingreso a una red neuronal es que deben tener la siguiente estructura:

Índice, alto, ancho y profundidad

Lo cual se corrige empleando el siguiente código de Python.

```

image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

```

Ya con las métricas y los dataset a emplear se procede a diseñar la GAN.

## 3 Diseño de la GAN.

### 3.1 Que son las redes neuronales y el aprendizaje profundo

Las redes neuronales son un modelo inspirado en el comportamiento biológico del cerebro, en donde elementos llamados neuronas comparten información para solucionar diferentes tipos de necesidades, en los últimos años ha tenido gran fuerza como rama de inteligencia artificial reconociendo voces, sonidos e imágenes a niveles de precisión increíbles.

La neurona artificial más simple es aquella que solo tiene una neurona que clasifica dos tipos de datos, como se puede visualizar en la ilustración número 10.

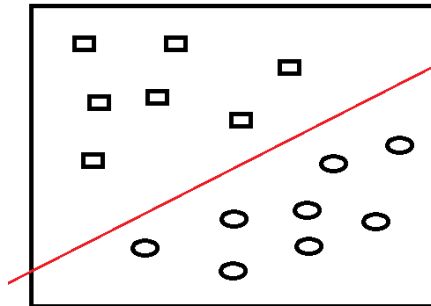


Ilustración 10 Red neuronal más simple

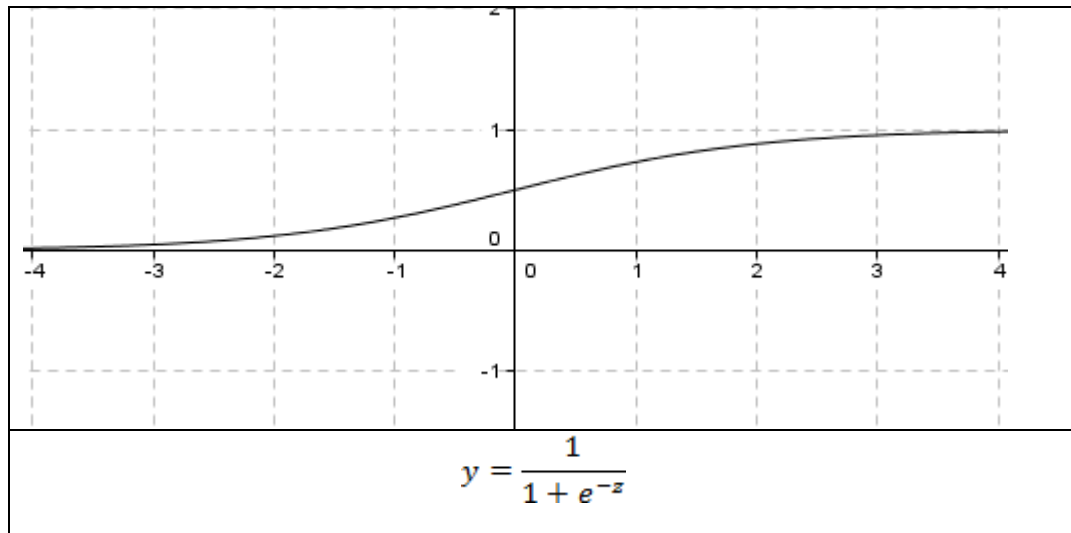
En donde los cuadros y círculos representan un tipo de dato diferente, en la versión más simple la recta roja separa correctamente los elementos, la cual se define con su ecuación:

$$y = wx + b$$

En donde  $w$  es un valor llamado **peso** y  $b$  se llama **sesgo**, donde el cálculo correcto de estos valores indica una correcta separación, así la red neuronal iteración tras iteración denominada **época** ajusta la recta de la mejor manera la cual se denomina **clasificador** lo cual se puede definir con la siguiente sumatoria:

$$z = \sum_0^i wixi + b$$

En donde la sumatoria indica la corrección realizada al peso en la época anterior, así modificando el peso que es la pendiente de la ecuación de la recta y el sesgo se mantiene igual, una vez se terminan las iteraciones se usa una función denominada de activación la cual se puede observar en la ilustración 11.



**Ilustración 11 Función de activación**

En donde según el valor dado por la ecuación de la recta que clasifica la información se obtiene un valor de 1 o 0 según el valor de  $z$ .

$$y = \begin{cases} 1 & \text{si } z > 0 \\ 0 & \text{si } z < 0 \end{cases}$$

De tal manera cuando es cuadrado el resultado es 0 o si es círculo el resultado es 1. Sin embargo, la clasificación de los problemas reales no es así de simple así que se requiere ajustar modelos que no sean lineales y por este motivo se usan más neuronas, en capas lo que tiene como consecuencias la variación de diferentes pesos para lograr el ajuste correspondiente.

El aprendizaje profundo (Deep learning) consiste en un conjunto de algoritmos basados en métodos de aprendizaje automáticos como lo son las redes neuronales en donde se manejan capas y pesos que son invisibles para la necesidad a solucionar y que intercambian la información para determinar la solución. Se divide en aprendizaje supervisado en donde se entrena la red a partir de un resultado al cual se quiere llegar y no supervisado lo cual es un paradigma de aprendizaje porque se obtienen elementos que vienen de patrones que no se tenían previstos.

### 3.2 Que son las GANS

En los últimos 10 años han surgido grandes investigaciones en redes neuronales, Ian Goodfellow presentó en 2014 las GANS que es como su nombre lo indica redes neuronales antagónicas, en donde su propósito es entrenar una red junto a otra red.

Se basa en la teoría de juegos de suma cero donde un jugador aprende de la competencia, un ejemplo claro es en el ajedrez en donde se analiza que se hizo mal, que se hizo bien y que se podría hacer para vencer al oponente.

Así el jugador que siempre le gana al otro tiene el nombre de **discriminador** y el que aprende constantemente se denomina **generador**. Se suele puntualizar con el ejemplo del investigador y el falsificador donde el investigador sabe cuáles son las propiedades que hacen de algo original y de esta manera le da pistas al falsificador para hacer que desarrolle más original [8], lo cual se ha implementado en múltiples situaciones [7] donde se requiere crear información para algún fin.

La implementación de una GAN en un software se puede visualizar en la ilustración 12 en donde el generador a partir de información no existente genera un tipo de información que se compara con la información real y el resultado del discriminador es determinar si es real o falso con un 1 o 0, y así se hace tantas veces hasta que el discriminador reconoce algo real como falso o algo falso como real.

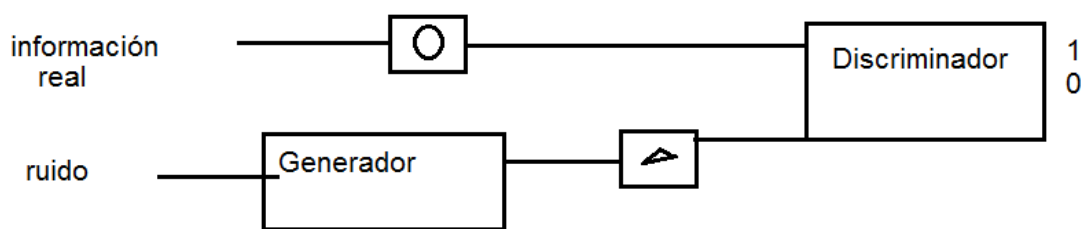


Ilustración 12 estructura de una GAN

### 3.3 Programación de una red neuronal en Keras

La creación de la red neuronal básica de un modelo en keras se hace mediante el comando

```
Model= sequential()
```

Inicialmente se define la estructura de la red neuronal antes de hacer el proceso de aprendizaje, para añadir definiciones las definiciones y capas al modelo se usa el comando **add**

```
model.add(Dense(10,activation='sigmoid',input_shape=(28,28)))
```

nodos
funcion de activación
Datos de entrada

En donde se especifica el número de neuronas que se pueden ver como los nodos que conectan a la siguiente capa, la función de activación y el tamaño de los datos de entrada, en cualquier momento se puede resumir la red usando el comando **model.summary()**.

Una vez se define la estructura de la red se programa el proceso de aprendizaje utilizando el método **compile** el cual define la función de pérdida (loss) que es el error entre las salidas deseadas y calculadas, por ejemplo, si es un problema de categorización se suele usar **categorical\_crossentropy** que define a cuál categoría permanecen los datos entrenados, donde este parámetro varía según el problema.

También permite definir el optimizador que permite calcular los pesos partir de los datos de entrada y la función de perdida existen entre estos optimizadores Adam, Adadelta, Adagrad, RMSprop y SGD (Gradient descent) [22] que haciendo uso de la primera derivada actualiza los parámetros, en donde existen hiperparametros como momentum o learning rate que evitan el estancamiento en ceros o multiplican el gradiente. Con la definición del optimizador se define como las capas se conectan con otras capas hacia adelante **forward propagation** o hacia atrás **back propagation** , lo cual se puede evidenciar en la ilustración 13 del hiperparametro SGD que realimenta las capas anteriores hasta que el valor de error sea 0, o se terminen las épocas.

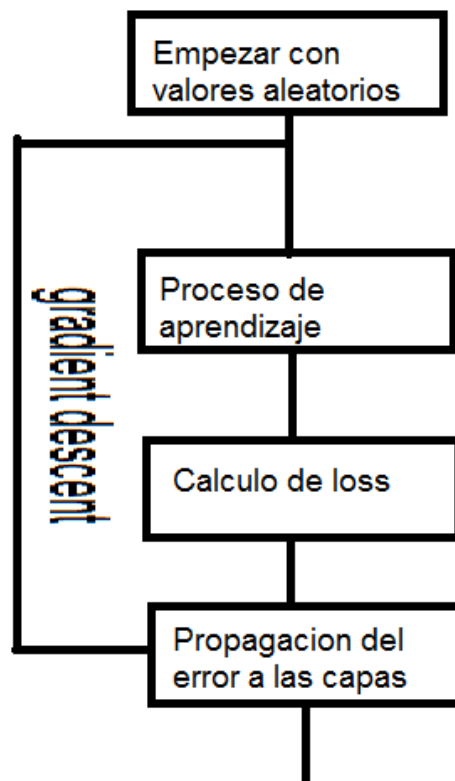


Ilustración 13 Proceso de aprendizaje de una red

El ultimo valor del método compile consiste en la métrica que sirve para monitorear el proceso de aprendizaje dependiendo del problema

```
model.compile (loss="categorical_crossentropy", optimizer="sgd", metrics=['accuracy'])
```

funcion de perdida                      optimizador                      metrica

Una vez se establece el proceso de aprendizaje, se procede a entrenar la red empleando el método **fit** el cual tiene como argumento los datos correctos de entrenamiento con su respectiva etiqueta, una variable denominada **batch\_size** que consiste en los bloques de la cantidad de datos con los que se entrena la red y las épocas o cantidad de veces con las cuales se entrena la red.



`model.fit(X_train, y_train, batch_size=100, epoch=5)`  
datos de bloques de datos epocas  
entrenamiento

Para evaluar la red y definir la efectividad del entrenamiento de la red con datos de prueba se emplea el método `evaluate`, el cual retorna los resultados según la métrica escogida anteriormente o el método `predict` que predice los resultados según el entrenamiento de la red.

### 3.4 Arquitecturas extractoras de características previamente entrenadas

El diseño previo a la GAN implica generar una red que tiene características de imágenes ya existentes, existen algunas ya diseñadas las cuales se enlistan en la siguiente tabla obtenida de la WEB de keras [5]

| Model             | Size   | Top-1 Accuracy | Top-5 Accuracy | Parameters  | Depth |
|-------------------|--------|----------------|----------------|-------------|-------|
| Xception          | 88 MB  | 0.790          | 0.945          | 22,910,480  | 126   |
| VGG16             | 528 MB | 0.713          | 0.901          | 138,357,544 | 23    |
| VGG19             | 549 MB | 0.713          | 0.900          | 143,667,240 | 26    |
| ResNet50          | 98 MB  | 0.749          | 0.921          | 25,636,712  | -     |
| ResNet101         | 171 MB | 0.764          | 0.928          | 44,707,176  | -     |
| ResNet152         | 232 MB | 0.766          | 0.931          | 60,419,944  | -     |
| ResNet50V2        | 98 MB  | 0.760          | 0.930          | 25,613,800  | -     |
| ResNet101V2       | 171 MB | 0.772          | 0.938          | 44,675,560  | -     |
| ResNet152V2       | 232 MB | 0.780          | 0.942          | 60,380,648  | -     |
| ResNeXt50         | 96 MB  | 0.777          | 0.938          | 25,097,128  | -     |
| ResNeXt101        | 170 MB | 0.787          | 0.943          | 44,315,560  | -     |
| InceptionV3       | 92 MB  | 0.779          | 0.937          | 23,851,784  | 159   |
| InceptionResNetV2 | 215 MB | 0.803          | 0.953          | 55,873,736  | 572   |
| MobileNet         | 16 MB  | 0.704          | 0.895          | 4,253,864   | 88    |
| MobileNetV2       | 14 MB  | 0.713          | 0.901          | 3,538,984   | 88    |
| DenseNet121       | 33 MB  | 0.750          | 0.923          | 8,062,504   | 121   |
| DenseNet169       | 57 MB  | 0.762          | 0.932          | 14,307,880  | 169   |
| DenseNet201       | 80 MB  | 0.773          | 0.936          | 20,242,984  | 201   |
| NASNetMobile      | 23 MB  | 0.744          | 0.919          | 5,326,716   | -     |
| NASNetLarge       | 343 MB | 0.825          | 0.960          | 88,949,818  | -     |

**Tabla 2 Redes pre-entrenadas**

En donde las más usadas en aplicaciones de SRGAN son VGG16 o VGG19 que contienen más parámetros que las otras redes, la finalidad de una red precargada es predecir una clase de un objeto, así si la imagen ingresada es de un animal específico y la red se precargo con una base de imágenes de animales, puede predecir que animal es.

Sin embargo, para el diseño de este trabajo no se requiere una red que clasifique, si uno una red para obtener las características de una imagen la cual permite disminuir la función de pérdida de la GAN, una red precargada como VGG16 tiene parámetros de entrada los cuales se pueden observar en la tabla 3.

| Parametros VGG16    |   |
|---------------------|---|
| <b>Include_top</b>  | Incluir capas densas  |
| <b>Weights</b>      | Preentrenamiento con base de datos Imagenet   |
| <b>Input_tensor</b> | Usar un modelo como entrada de la red.  |
| <b>Input_shape</b>  | Entrada del tensor, donde debe tener un alto, un ancho no menor a 32 pixeles y una profundidad. Por defecto el valor es (200,200,3) |
| <b>Pooling</b>      | Si se desea agregar un tipo de agrupamiento   |
| <b>Classes</b>      | Numero de clases, si no se carga alguna base de imágenes y la red se entrena, se definen con cuantas clases se entrena la red.      |

Tabla 3 parametros VGG16

Una vez se obtienen las imágenes se carga la red en Python, la cual tiene la siguiente estructura:

| ConvNet Configuration       |                        |                               |  |  |   |
|-----------------------------|------------------------|-------------------------------|--|--|---|
| A                           | A-LRN                  | B                             | C  | D  | E   |
| 11 weight layers            | 11 weight layers       | 13 weight layers              | 16 weight layers                           | 16 weight layers                           | 19 weight layers  |
| input (224 × 224 RGB image) |                        |                               |  |  |   |
| conv3-64                    | conv3-64<br><b>LRN</b> | conv3-64<br><b>conv3-64</b>   | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                                    |
| maxpool                     |                        |                               |  |  |   |
| conv3-128                   | conv3-128              | conv3-128<br><b>conv3-128</b> | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                                  |
| maxpool                     |                        |                               |  |  |   |
| conv3-256<br>conv3-256      | conv3-256<br>conv3-256 | conv3-256<br>conv3-256        | conv3-256<br>conv3-256<br><b>conv1-256</b> | conv3-256<br>conv3-256<br><b>conv3-256</b> | conv3-256<br>conv3-256<br>conv3-256<br><b>conv3-256</b> |
| maxpool                     |                        |                               |  |  |   |
| conv3-512<br>conv3-512      | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
| maxpool                     |                        |                               |  |  |   |
| conv3-512<br>conv3-512      | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
| maxpool                     |                        |                               |  |  |   |
| FC-4096                     |                        |                               |  |  |   |
| FC-4096                     |                        |                               |  |  |   |
| FC-1000                     |                        |                               |  |  |   |
| soft-max                    |                        |                               |  |  |   |

Tabla 4 Arquitectura VGG16

La implementación en Python se escribe a continuación [21]

```
from keras.preprocessing.image import load_img
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
```

```
image = load_img('mug.jpg', target_size=(224, 224))
```

```

image = img_to_array(image)
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

image = preprocess_input(image)
yhat = model.predict(image)

# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))

```

tomado de: [https://github.com/zubairahmed-ai/Keras\\_DeepLearning/blob/master/Model.py](https://github.com/zubairahmed-ai/Keras_DeepLearning/blob/master/Model.py)

Donde según la imagen que se carga se obtiene una predicción de que clase es así como un porcentaje de acierto, y usando el comando `model.summary()` obtiene la siguiente estructura del modelo:

| Layer (type)               | Output Shape          | Param #   |
|----------------------------|-----------------------|-----------|
| input_3 (InputLayer)       | (None, 224, 224, 3)   | 0         |
| block1_conv1 (Conv2D)      | (None, 224, 224, 64)  | 1792      |
| block1_conv2 (Conv2D)      | (None, 224, 224, 64)  | 36928     |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64)  | 0         |
| block2_conv1 (Conv2D)      | (None, 112, 112, 128) | 73856     |
| block2_conv2 (Conv2D)      | (None, 112, 112, 128) | 147584    |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128)   | 0         |
| block3_conv1 (Conv2D)      | (None, 56, 56, 256)   | 295168    |
| block3_conv2 (Conv2D)      | (None, 56, 56, 256)   | 590080    |
| block3_conv3 (Conv2D)      | (None, 56, 56, 256)   | 590080    |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256)   | 0         |
| block4_conv1 (Conv2D)      | (None, 28, 28, 512)   | 1180160   |
| block4_conv2 (Conv2D)      | (None, 28, 28, 512)   | 2359808   |
| block4_conv3 (Conv2D)      | (None, 28, 28, 512)   | 2359808   |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512)   | 0         |
| block5_conv1 (Conv2D)      | (None, 14, 14, 512)   | 2359808   |
| block5_conv2 (Conv2D)      | (None, 14, 14, 512)   | 2359808   |
| block5_conv3 (Conv2D)      | (None, 14, 14, 512)   | 2359808   |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512)     | 0         |
| flatten (Flatten)          | (None, 25088)         | 0         |
| fc1 (Dense)                | (None, 4096)          | 102764544 |
| fc2 (Dense)                | (None, 4096)          | 16781312  |
| predictions (Dense)        | (None, 1000)          | 4097000   |

**Ilustración 14 Resumen Keras VGG16**

Donde se puede evidenciar que se emplean redes convolucionales para extraer información de la imagen, así como para reducir la información, es importante

observar la cantidad de parámetros que maneja una arquitectura en la capa densa fc1(1027645544), la cual al momento de implementarlo en una GAN por la cantidad de información requiere un clúster de forma obligatoria.

A continuación, es muestra la imagen original con 10 características diferentes obtenidas en diferentes etapas del procesamiento.




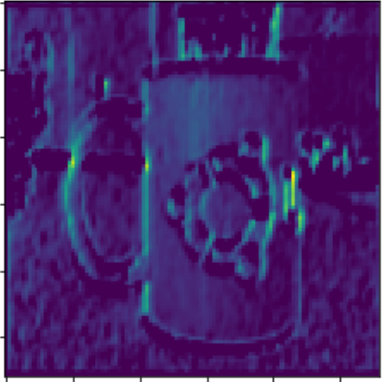
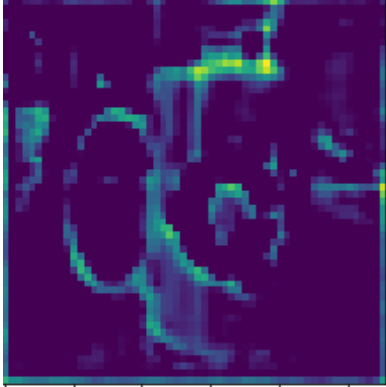
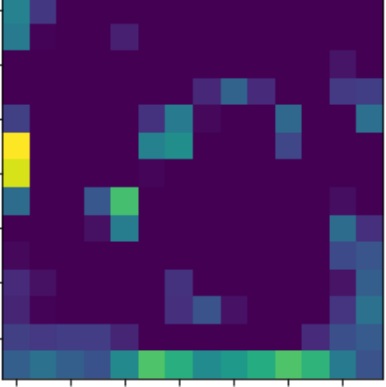
| Original   | Primer filtro del primer bloque convolucional  |
|--|--|
|                     |                      |
| <p>Tercer filtro del primer bloque convolucional donde la imagen tiene una resolución de 224*224</p> | <p>Filtro número 32 del 2 bloque donde la imagen tiene una resolución de 112 * 112.</p>                |
|                   |                     |
| <p>Filtro número 60 del 3 bloque convolucional donde la imagen tiene una resolución de 56*56</p>     | <p>Filtro número 512 del último bloque convolucional donde la imagen tiene una resolución de 14*14</p> |
|                   |                    |

Tabla 5 Prueba arquitectura vgg16

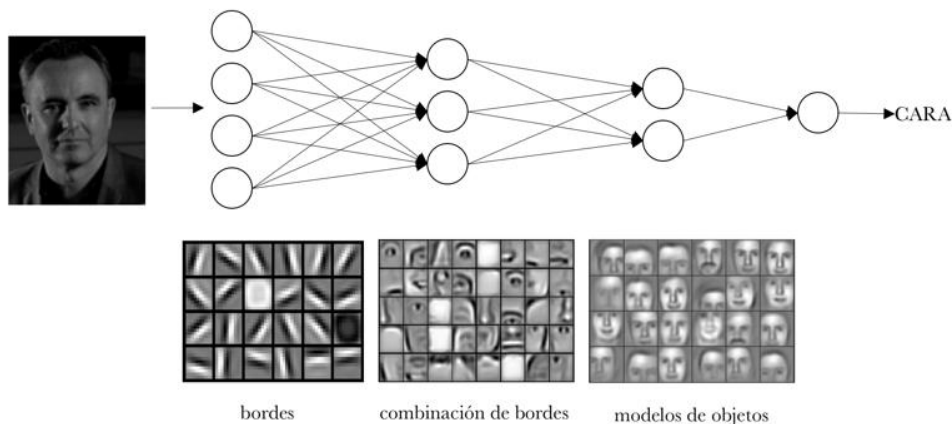
Donde se puede evidenciar claramente como la red obtiene iteración tras iteración las características de las imágenes donde posteriormente se juntan

todas las imágenes generando así una matriz con todas las características y donde se **compara** con una ya existente comprobando así si se parece a una clase de tipo pocillo con el comando **decode\_predictions**.

Así se procede a diseñar una red en donde se puedan controlar los tipos de datos con los que se entrenan la red, y no depender de estructuras ya realizadas porque la comparación con múltiples clases hace que el tiempo de procesamiento de la red sea demorado.

### 3.5 Diseño de una red extractora de características

La entrada es una matriz de imágenes, creando así una CNN [4] la cual se caracteriza porque las entradas son imágenes donde su principal característica es que aprende jerarquías de esta; así inicialmente se toman las formas de un borde, posteriormente de una figura geométrica, y así sucesivamente hasta obtener descrita la imagen como se puede visualizar en la ilustración 15 y en la tabla 5 de la arquitectura VGG donde cada filtro adquiere una característica diferente.

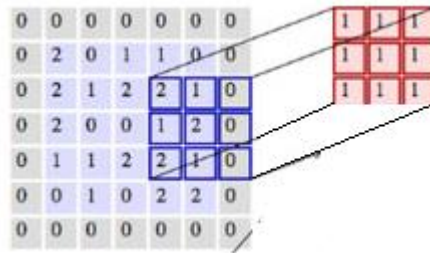


**Ilustración 15 Características CNN**

Inicialmente se diseña define el modelo con el comando `sequential()`, y posteriormente para obtener las características se establecen los filtros que establecen los parámetros de las capas ocultas de la red, en donde inicialmente inicia una capa de convolución con el siguiente comando en keras

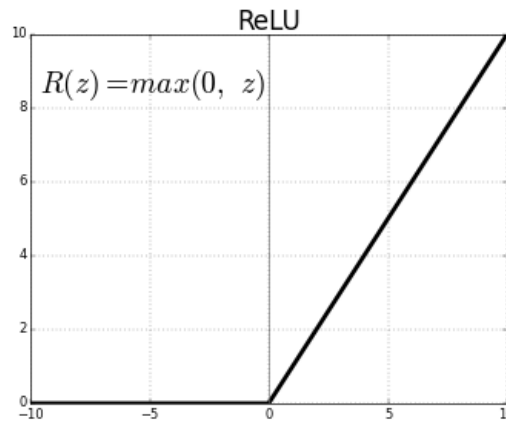
```
srcnn=Sequential()
srcnn.add(Conv2D(filters = 64, kernel_size = (16,16),padding = 'Same',
activation ='relu', input_shape = tam))
```

Como se había descrito anteriormente el primer dataset es MINST que contiene imágenes de  $28 * 28 * 1$  porque son a blanco y negro, las cuales se dividen sobre 255 que es la resolución de una imagen a blanco y negro para normalizar los valores entre 0 y 1. Cuando ingresa a la red neuronal se emplea una máscara convolucional de  $16 * 16$  la cual recorre la imagen para obtener las características de la imagen como se puede apreciar en la ilustración 16:



**Ilustración 16 Convolución en una imagen**

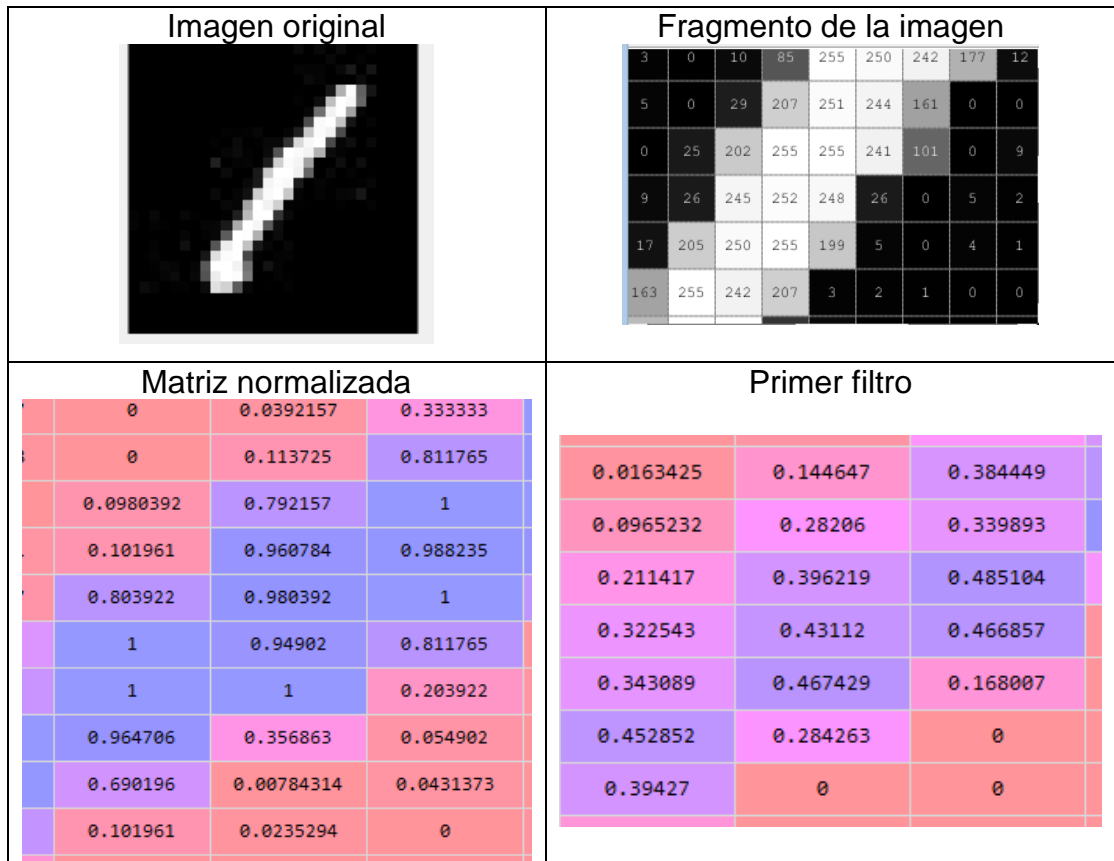
En donde según la ventana establecida va recorriendo la imagen original, donde por cada ventana una neurona en la capa oculta procesa la información, siempre se inicia con la ventana en la esquina arriba-izquierda de la imagen y se desplaza hacia la derecha, se rellena la información de alrededor de la imagen con 0 usando el comando padding=same, el tamaño de entrada de la imagen que es 28 \* 28, por último se emplea una función de activación **relu**, en donde la función de activación limita únicamente a valores positivos o un rango específico, lo cual se puede apreciar en la ilustración 17:



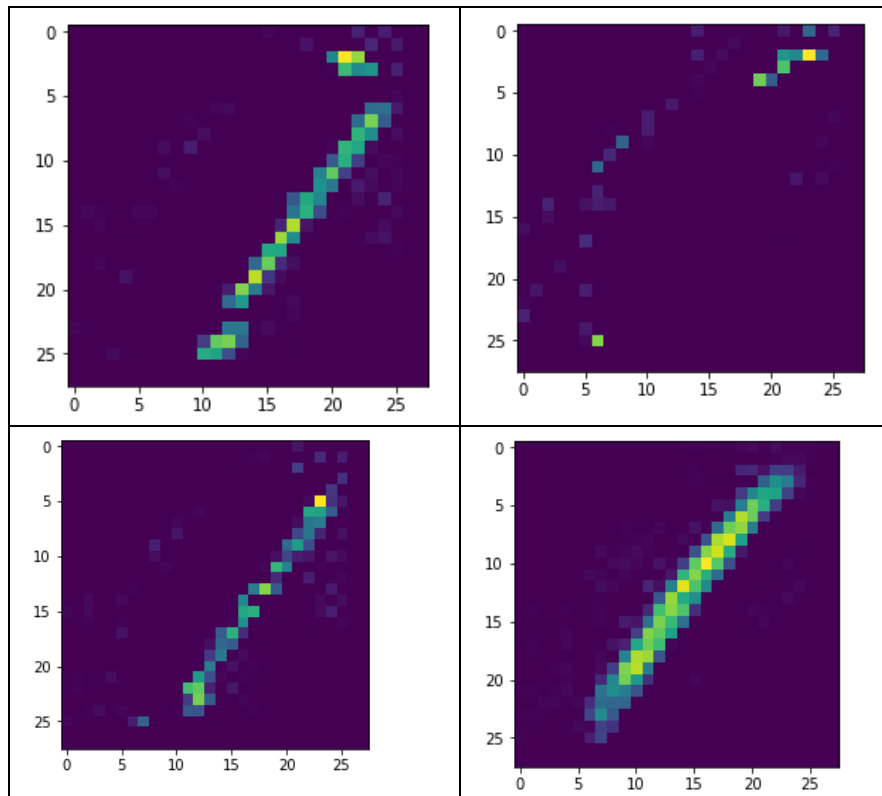
**Ilustración 17 Función de activación ReLu**

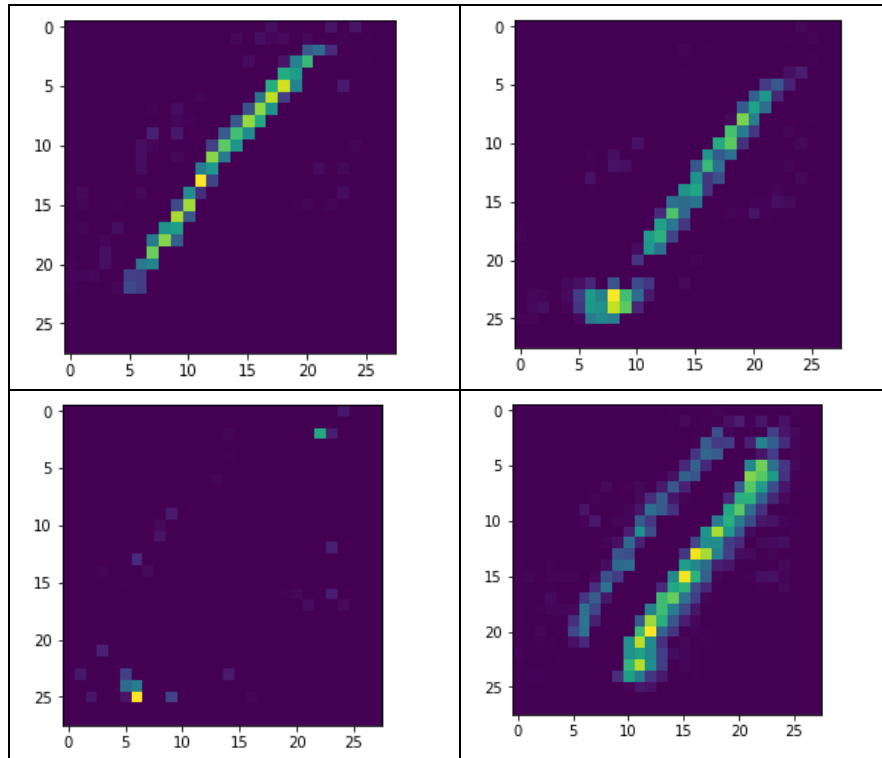
Se puede comprobar que los números mayores a 0, tiene el mismo el valor y los valores negativos equivalen a 0. Así es primordial aclarar que después de la normalización nombrada en capítulos anteriores la imagen está en un intervalo de 0 a 1 como se observar en la tabla 6, también se puede observar que el resultado del primer filtro es una matriz con números en este mismo intervalo, así como el resultado del primer filtro.

Otra característica que se puede apreciar es que cada filtro determina algunas características y las cuantifica, en la tabla 7 se puede observar cómo cada filtro obtiene alguna característica de la imagen.



**Tabla 6 Normalización y primer filtro**





**Tabla 7 Diferentes capas de convolución**

Así se programan tres funciones de convolución para obtener diferentes filtros con diferentes tamaños de la ventana de convolución, la relación que se utiliza se expone en la siguiente tabla:

| Numero de filtros | Tamaño de la ventana |
|-------------------|----------------------|
| 64                | 16                   |
| 32                | 8                    |
| 3                 | 2                    |

**Tabla 8 Relación filtros con ventana de convolución**

En donde por costes computacionales entre más grande es el filtro más grande es el tamaño de la ventana, al finalizar solo se obtienen 3 filtros que es la dimensión de una imagen RGB, para imágenes a blanco y negro la información se triplica en los 3 canales para obtener el mismo procesamiento, el resumen de la red se muestra en la ilustración 18.

| Layer (type)       | Output Shape       | Param # |
|--------------------|--------------------|---------|
| conv2d_68 (Conv2D) | (None, 28, 28, 64) | 49216   |
| conv2d_69 (Conv2D) | (None, 28, 28, 32) | 131104  |
| conv2d_70 (Conv2D) | (None, 28, 28, 3)  | 387     |

**Ilustración 18 Resumen SRCNN**

Donde se puede contemplar la cantidad de parámetros que se van a modificar en las etapas de entrenamiento y en donde se finaliza con una capa de salida



con una resolución de  $28 * 28 * 3$  con la mejor versión según las características aprendidas previamente. Así se termina de definir la red con la función **construir\_red**

```
def construir_red(tam=(7,7,1)):
    srcnn=Sequential()

    srcnn.add(Conv2D(filters = 64, kernel_size = (16,16),padding = 'Same',
                    activation = 'relu', input_shape = tam))

    srcnn.add(Conv2D(filters = 32, kernel_size = (8,8),padding = 'Same',
                    activation = 'relu'))
    srcnn.add(Conv2D(filters = 3, kernel_size = (2,2),padding = 'Same',
                    activation = 'relu'))

    opt = optimizers.Adam(lr=0.001)
    srcnn.compile(optimizer=opt,loss='mean_squared_error')
    return srcnn
```

Donde aparte de añadir las capas de convolución se procede a definir un optimizador que como se explicó previamente determina la variación de los pesos en este caso se opta por el optimizador Adam [22] ya que tiene mejores resultados en diversas investigaciones, y la pérdida de evalúa con la métrica **mse** mencionada anteriormente porque se comparan dos imágenes una original y su versión en baja resolución.

Existen diseños de SRCNN (Super resolution CNN) [20], que consisten en extraer las características usando las capas de convolución, así el diseño de la SRCNN que extrae las características se aprecia en la ilustración 19:

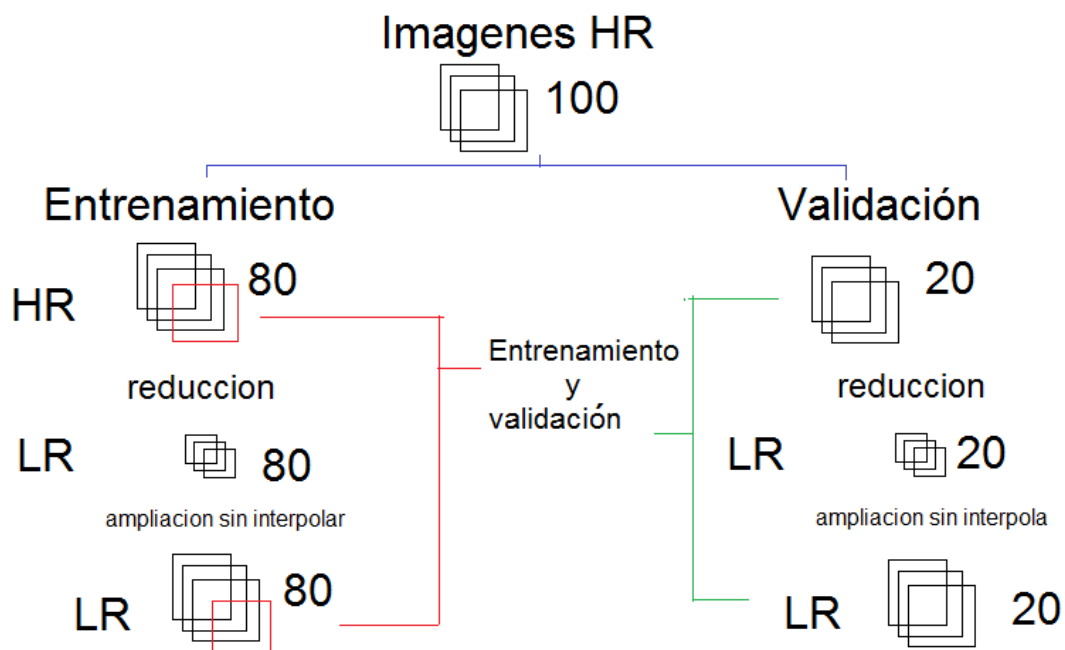


Ilustración 19 Preparación datos SRCNN

Donde se puede observar que a partir de 100 imágenes se dividen en 80 para entrenamiento y 20 para validación, con la función **carga\_imagenes** se obtiene

la versión del mismo tamaño, pero con una escala de reducción, lo cual se implementa en el siguiente fragmento de código:

```
escala=2
nimagenes=100
tam=(28,28)
imgs_hr, imgs_lr, tam_lr, tam_hr=carga_imagenes(nimagenes,"minst",tam,escala)
Div_en=80 # %
```

Obteniendo así las imágenes en alta y baja definición con su respectivo tamaño para operaciones futuras, procediendo así a transformar las imágenes en matrices para el ingreso a la red neuronal, tanto para las imágenes de entrenamiento y validación, la primera parte consiste en el siguiente ciclo el cual guarda una lista con las posiciones de cada imagen y la segunda parte en su conversión a un array.

```
for i in range(0,int(nimagenes*Div_en/100)):
    vx = imgs_hr[i,:,:]/255
    vy = imgs_lr[i,:,:]/255
    h = vy.shape[0]
    w = vy.shape[1]

    for x in range(0, h-tam[0]+1, paso):
        for y in range(0, w-tam[1]+1, paso):

            tx = vx[x:x+tam[0], y:y+tam[1,:]].reshape(tam[0],tam[1],3)
            ty = vy[x:x+tam[0], y:y+tam[1,:]].reshape(tam[0],tam[1],3)
            X_train.append(tx)
            Y_train.append(ty)

X_train = np.array(X_train)
Y_train = np.array(Y_train)
X_val = np.array(X_val)
Y_val = np.array(Y_val)
```

Una vez con las imágenes adecuadas se entrena la red previamente creada utilizando el siguiente comando:

```
srcnn=construir_red(tx.shape)

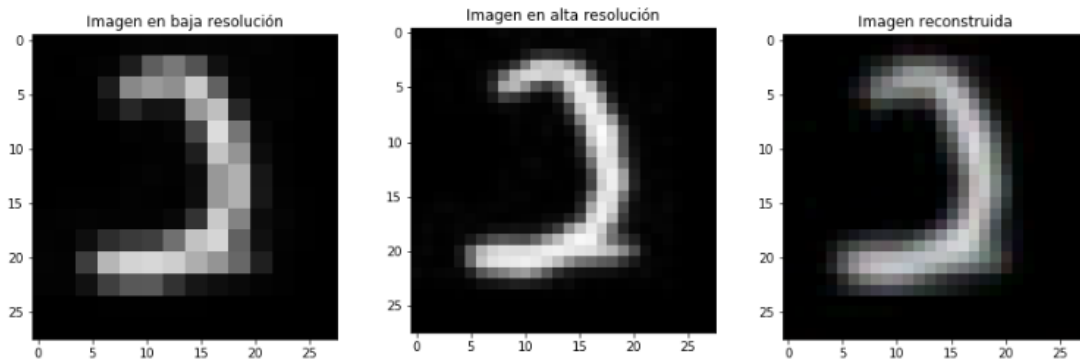
srcnn.fit(X_train, Y_train, batch_size = 20, epochs = 60, validation_data=(X_val, Y_val))
```

Donde la primera línea sirve para generar la red previamente diseñada, la segunda para entrenar el modelo llamado **srcgan** en las que los parámetros de ingreso son las imágenes previamente creadas, las épocas y el tamaño de imágenes para cada una.

Por último, se guarda la red neuronal que va a ser el insumo para reconocer las características de la GAN empleando el siguiente comando

```
srcnn.save('srcgan.h5')
```

El resultado del código se puede observar en la ilustración 20, en donde se evidencia la imagen original, la imagen con baja resolución y la imagen reconstruida de una imagen de las generadas por la red.

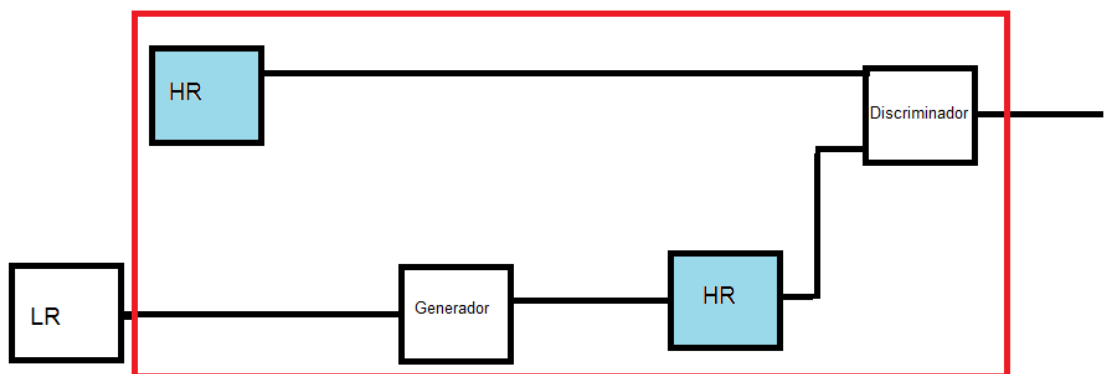


**Ilustración 20 Resultados SRCNN**

Se puede observar que visualmente se parece la imagen de alta resolución a la imagen reconstruida, también se utiliza la función para obtener la métrica de equivalencia dando como resultado un **PSNR** de 32.84 db y un **SSIM** de 0.915 que son valores no alejados a los de la tabla 1.

### 3.6 Esquema de la GAN

El objetivo de la red es aumentar la resolución de una imagen a partir de unos datos reales que entrenan la red, la GAN tiene la estructura que se puede visualizar en la ilustración 21 en donde el ingreso son imágenes en baja resolución (LR) y tiene 2 salidas una es la imagen que se predice producida por el generador y el porcentaje que parece a la imagen original producida por el discriminador.

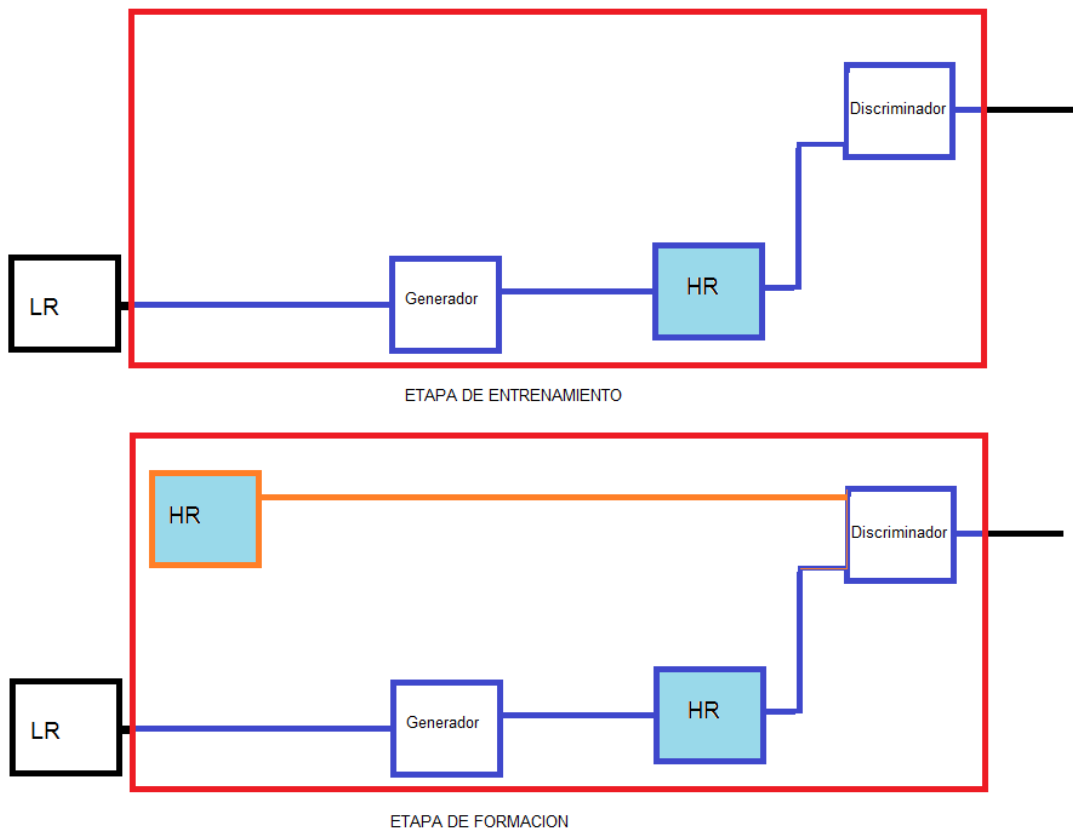


Estructura GAN

**Ilustración 21 Estructura GAN**

Se deben diseñar para la GAN dos redes neuronales una red donde ingresan las imágenes en baja resolución y su salida es una imagen en alta resolución como se puede visualizar en la ilustración 21 y un discriminador que toma una imagen falsa y real y determina la validez de cada una. Inicialmente el generador y discriminador evalúan las imágenes reales y las creadas por el generador en alta resolución para determinar cuáles son correctas y cuales son erróneas.

Para entender el funcionamiento es necesario identificar que las dos redes creadas funcionan como una sola por este motivo se encierra en el cuadro rojo y que funcionan en 2 etapas que se pueden observar en la ilustración 22.



**Ilustración 22 Etapas de entrenamiento GAN**

Una vez se entrena el discriminador en cuales imágenes son reales y cuales son falsas y se adecuan los pesos, se genera una etapa denominada de entrenamiento en la cual no se ingresan imágenes nuevas al discriminador y se congela solamente; se actualizan los pesos hasta que el generador cree imágenes parecidas a las originales. En esta etapa es importante considerar que se cargan los pesos de la red que obtiene las características de la imagen la cual genera el parámetro para que mejore la red.

Posteriormente en una etapa denominada de formación, se ingresan nuevas imágenes en el discriminador lo cual se puede observar con la línea naranja en la ilustración 22 y se comprueba con el discriminador la calidad de las nuevas imágenes con las ya generadas. Este proceso se repite hasta que el discriminador no distinga entre una imagen real y una imagen creada por el generador.

Así una vez es claro el proceso de diseño y con la red que adecua los pesos según las características de la imagen se procede a hacer el generador.

### 3.7 Programación de la GAN

Para la estructura general de la red se hace uso de tensores que consisten en crear estructuras simbólicas con las redes creadas, el diseño que se usó en el código es el siguiente:

```
input_high_resolution = Input(shape=size_hr)
input_low_resolution = Input(shape=size_lr)

generated = generador(input_low_resolution)

features = vgg(generated)

discriminador.trainable = False

probs = discriminador(generated)
```

Donde las entradas según la ilustración 21 son dos imágenes una en baja resolución y una en alta resolución, al generador ingresa la imagen en baja resolución generando una versión en alta resolución con el generador, después con la red precargada se obtienen las características de la imagen de esta imagen generada y con el discriminador la probabilidad de que tan cerca está a una imagen real.

En la etapa de entrenamiento el discriminador no se entrena con las imágenes con alta definición por este motivo su atributo es **false** y por último con el siguiente comando se crea y se compila la red neuronal antagónica

```
adversarial_model = Model([input_low_resolution, input_high_resolution], [probs, features])
adversarial_model.compile(loss=['binary_crossentropy', 'mse'], loss_weights=[1e-3, 1],
optimizer=optimizador)
```

Así al modelo le ingresan el grupo de las dos imágenes y se obtienen las características y probabilidades, donde el objetivo es reducir la probabilidad usando la pérdida `binary_crossentropy` [23], la cual es una función de pérdida logarítmica que por su característica logarítmica acentúa las predicciones erróneas y un valor denominado **loss\_weights** el cual pondera las pérdidas en un rango, que en este caso está entre 0.001 y 1.

Una vez definida la estructura de la GAN, tanto conceptualmente como su programación, se procede a implementar el código completo, inicialmente se usa la función **carga\_imagenes** para tener el grupo de imágenes en alta y baja definición, las cuales se normalizan dividiendo el valor de cada canal del píxel entre 255 para obtener los valores entre 0 y 1.

Posteriormente se llaman las funciones `crear_generador` y `crear_discriminador`, los cuales tienen la estructura de la red neuronal generadora y discriminadora, el código implementado se muestra a continuación y la estructura de la red generadora y discriminadora en la tabla 9.

```
epocas=20
nimagenes=100
loteimagenes=10
```

```
tam=(40,40)
optimizador=adam(lr=0.0002, beta_1=0.5)
escala=2
```

```
imgs_hr, imgs_lr, size_lr, size_hr=carga_imagenes(nimagenes, "div2k", (40,40), escala)
imgs_hr=imgs_hr/255
imgs_lr=imgs_lr/255
```

```
generador= crear_generador(size_lr, escala)
discriminador = crear_discriminador(size_hr)
```

| Programación generador   | Programación discriminador  |
|--|---|
| <pre>def crear_generador(size=(7,7,1),escala=1):      generator=Sequential()      generator.add(Dense(units=256,input_shape = size))     generator.add(Flatten())     generator.add(LeakyReLU(0.2))      generator.add(Dense(units=512))     generator.add(LeakyReLU(0.2))      generator.add(Dense(units=1024))     generator.add(LeakyReLU(0.2))      generator.add(Dense(units=2352, activation='sigmoid'))     generator.add(Reshape((28,28,3)))     generator.summary()      return generator</pre> | <pre>def crear_discriminador(size=(7,7,1)):      discriminator=Sequential()      discriminator.add(Dense(units=1024,input_shape = size))     discriminator.add(LeakyReLU(0.2))     discriminator.add(Dropout(0.3))      discriminator.add(Dense(units=512))     discriminator.add(LeakyReLU(0.2))     discriminator.add(Dropout(0.3))      discriminator.add(Dense(units=256))     discriminator.add(LeakyReLU(0.2))      discriminator.add(Dense(units=3, activation='sigmoid'))      return discriminator</pre> |

**Tabla 9 Programación generador y discriminador**

El generador tiene como entrada una imagen en baja resolución y obtiene como salida una imagen en alta resolución con el mismo tamaño, inicialmente se aplana la matriz usando el comando **flatten**, posteriormente con el comando **LeakyReLU** la cual difiere a Relu porque cuando esta inactiva o tiene un valor negativo permite una pendiente como se puede observar en la siguiente función a trozos:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

**Ilustración 23 Funcion LeayRelu**

Donde el valor de la entrada es el coeficiente de la pendiente que en este casi equivale a 0.2 lo cual equivale a 11 grados, se escoge esta función porque es de las rectificaciones más usadas en la creación de las GAN, las etapas de densidad consisten en las conexiones que se generan entre red y red. La última etapa consiste en una función para reacomodar las 2352 neuronas con la resolución de la imagen de 28 \* 28 \*3 que es la versión en alta resolución, se emplea una función de activación sigmoid (ilustración 25) por su carácter exponencial [24].

Existen procesos como **Pooling** que consiste en reducir la imagen con la información más valiosa como se puede observar en la ilustración 24 sin embargo creas más capas lo cual hace computacionalmente más complejo el cálculo.

## Max Pooling

|   |   |   |   |
|---|---|---|---|
| 2 | 7 | 9 | 7 |
| 4 | 9 | 1 | 4 |
| 4 | 7 | 6 | 2 |
| 5 | 1 | 8 | 2 |

|   |   |
|---|---|
| 9 | 9 |
| 7 | 8 |

Ilustración 24 Etapa de Pooling

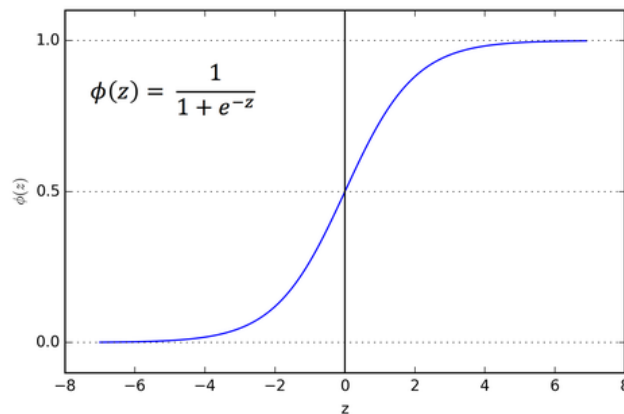


Ilustración 25 Funcion Sigmoid

El discriminador a diferencia del generador a partir de una los datos de entrada donde tiene resultados correctos y erróneos genera una matriz de salida en donde clasifica los datos que son correctos y los que no lo son.

Para el diseño del discriminador tampoco se emplean capas convolucionales solo la imagen con un 1 si es real y 0 si no lo es, se itera y del proceso el resultado es el porcentaje de qué tan igual es así como se puede observar en la tabla 9 ingresa la imagen se obtienen 3 capas donde cada una tiene cada vez menos neuronas sin modificar la resolución de la imagen y la última capa tiene una profundidad de 3 los cuales son los bits que determinan el color de la imagen y a su vez contienen la distribución probabilística del parecido a una imagen real, en la ilustración 26 se puede observar un resumen del generador obtenido por keras y en la ilustración 27 un resumen del generador.

| Layer (type)                  | Output Shape        | Param #   |
|-------------------------------|---------------------|-----------|
| dense_17 (Dense)              | (None, 28, 28, 256) | 1024      |
| flatten_3 (Flatten)           | (None, 200704)      | 0         |
| leaky_re_lu_13 (LeakyReLU)    | (None, 200704)      | 0         |
| dense_18 (Dense)              | (None, 512)         | 102760960 |
| leaky_re_lu_14 (LeakyReLU)    | (None, 512)         | 0         |
| dense_19 (Dense)              | (None, 1024)        | 525312    |
| leaky_re_lu_15 (LeakyReLU)    | (None, 1024)        | 0         |
| dense_20 (Dense)              | (None, 2352)        | 2410800   |
| reshape_3 (Reshape)           | (None, 28, 28, 3)   | 0         |
| =====                         |                     |           |
| Total params: 105,698,096     |                     |           |
| Trainable params: 105,698,096 |                     |           |
| Non-trainable params: 0       |                     |           |

**Ilustración 26 Resumen del generador**

| Layer (type)               | Output Shape         | Param # |
|----------------------------|----------------------|---------|
| dense_21 (Dense)           | (None, 28, 28, 1024) | 4096    |
| leaky_re_lu_16 (LeakyReLU) | (None, 28, 28, 1024) | 0       |
| dropout_5 (Dropout)        | (None, 28, 28, 1024) | 0       |
| dense_22 (Dense)           | (None, 28, 28, 512)  | 524800  |
| leaky_re_lu_17 (LeakyReLU) | (None, 28, 28, 512)  | 0       |
| dropout_6 (Dropout)        | (None, 28, 28, 512)  | 0       |
| dense_23 (Dense)           | (None, 28, 28, 256)  | 131328  |
| leaky_re_lu_18 (LeakyReLU) | (None, 28, 28, 256)  | 0       |
| dense_24 (Dense)           | (None, 28, 28, 1)    | 257     |
| =====                      |                      |         |
| Total params: 660,481      |                      |         |
| Trainable params: 660,481  |                      |         |
| Non-trainable params: 0    |                      |         |

**Ilustración 27 Resumen del discriminador**



Una vez definido el modelo de la GAN, la construcción del generador y discriminador se procede a implementar el código principal, se declaran las épocas, el número de imágenes, el lote de imágenes de entrenamiento, la escala reducción y el optimizador que se va a utilizar en cada caso. También se emplea la función **carga\_imagenes** creada anteriormente, se normalizan las variables y se crea el generador y el discriminador.

```

epocas=30
nimagenes=100
loteimagenes=10
optimizador=adam(lr=0.0002, beta_1=0.5)
escala=2
#Carga las imagenes de los dataset, y obtiene las imagenes en alta y baja resolucion

imgs_hr, imgs_lr, size_lr, size_hr=carga_imagenes(nimagenes,"MINST", (28,28), escala)
imgs_hr=imgs_hr/255
imgs_lr=imgs_lr/255
#Crear generador y discriminador

generador= crear_generador(size_lr, escala)
discriminador = crear_discriminador(size_hr)

```

Posteriormente se emplea a crear una red llamada vgg para estandarizar a las redes conocidas extractoras de datos [25], que en este caso es la red SRGAN entrenada previamente, la cual se presenta a continuación:

```

def construir_vgg(size=(7,7,1)):

    path_model = 'srgan.h5'
    input_shape = size
    vgg = load_model(path_model)
    vgg.summary()
    vgg.outputs = [vgg.layers[2].output]
    input_layer = Input(shape=input_shape)
    features = vgg(input_layer)
    model = Model(inputs=[input_layer], outputs=[features])
    return model

```

En donde se carga el modelo obtenido anteriormente y se crea el respectivo modelo al igual que la gan, donde la entrada es una imagen y la salida es una imagen con las características a una imagen parecida que sea real, una vez se construye la red extractora de características se procede a modificar las imágenes para la entrada a la red, así como explico al finalizar el capítulo 1.

```

vgg=construir_vgg(size_hr)
vgg.trainable = False
vgg.compile(loss='mse', optimizer=optimizador, metrics=['accuracy'])
discriminador.compile(loss="binary_crossentropy", optimizer=optimizador)

imgs_lr=imgs_lr.reshape(imgs_lr.shape[0], imgs_lr.shape[1], imgs_lr.shape[2], 3)
imgs_hr=imgs_hr.reshape(imgs_hr.shape[0], imgs_hr.shape[1], imgs_hr.shape[2], 3)

```

Posteriormente se realiza el ciclo que corresponde a las épocas, aunque como en el diseño de la red extractora de características la función **fit** contiene el parámetro **epoch** la red antagónica generativa es un modelo y requiere realizar un proceso de entrenamiento personalizado.

Inicialmente se generan los ciclos en donde uno se encarga de las épocas y otra entrena la red según el lote de imágenes establecido, se obtienen aleatoriamente las imágenes para crear un lote en baja resolución y un lote en alta resolución, a continuación, se presenta el código:

```
for e in range(1, epocas+1):
    print ('-'*15, 'Epoch %d' % e, '-'*15)
    for _ in range(loteimagenes):

        rand_nums = np.random.randint(0, imgs_hr.shape[0], size=loteimagenes)
        lote_hr = imgs_hr[rand_nums]
        lote_lr = imgs_lr[rand_nums]
```

Posteriormente se procede a generar imágenes en alta resolución a partir del generador y se crean matrices con 1 para la información real y 0 para la información falsa

```
generated_images_sr = generador.predict(lote_lr)
real_data_Y = np.ones((loteimagenes, 28, 28, 3))
fake_data_Y = np.zeros((loteimagenes, 28, 28, 3))
```

Se procede a entrenar el discriminador para la etapa de entrenamiento contemplada en la ilustración 22, entrenando con 1 las imágenes reales y con 0 las imágenes creadas por el generador, añadiendo a una matriz el resultado obtenido, que va a ser un indicador de la efectividad de la GAN.

```
discriminador.trainable = True

d_loss_real = discriminador.train_on_batch(lote_hr, real_data_Y)
d_loss_fake = discriminador.train_on_batch(generated_images_sr, fake_data_Y)
d_loss = 0.5 * np.add(d_loss_fake, d_loss_real)
```

Una vez entrenada red con el bloque de imágenes se procede a generar para la GAN de nuevo un lote de imágenes en alta y baja definición, se crea un vector con 1 que son el indicador de que una imagen es real, y se ingresa a la otra etapa donde el discriminador no se entrena, se entrena la GAN.

```
rand_nums = np.random.randint(0, imgs_hr.shape[0], size=loteimagenes)
lote_hr = imgs_hr[rand_nums]
lote_lr = imgs_lr[rand_nums]

gan_Y = np.ones((loteimagenes, 28, 28, 3)) # Acomodar automaticamente

discriminador.trainable = False
```

Para finalizar el proceso se obtienen las características del lote de alta definición y se procede a entrenar la GAN, el comando empleado al igual que el discriminador se denomina **train\_on\_batch** que se utiliza en situaciones donde se entrenan por lotes de datos, al proceso de entrenamiento ingresan los lotes en baja resolución con el vector creado con 1 así generando que la GAN los identifique como correctos para ajustar los pesos a este objetivo, y el lote de alta definición con las características para guiar el mismo proceso. El código se presenta a continuación:

```
features = vgg.predict(lote_hr)
g_loss = adversarial_model.train_on_batch([lote_lr, lote_hr],
                                          [gan_Y, features])
```

## 4. Resultados de la GAN

### 4.1 Resultados en general

A continuación, se hace una descripción de los resultados de las imágenes con las respectivas métricas de medición, así como sus épocas, inicialmente la tabla 10 se muestran los resultados que se obtienen con la red extractora de características:

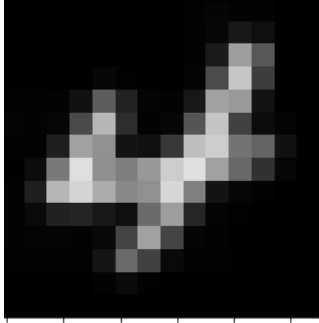
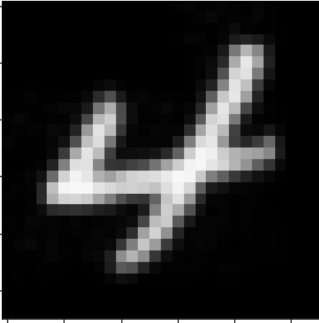
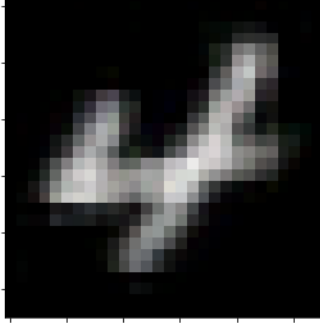
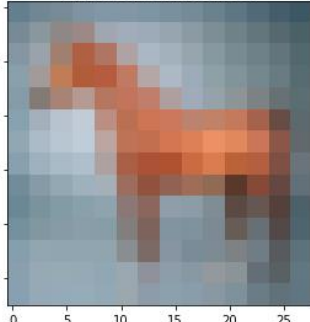
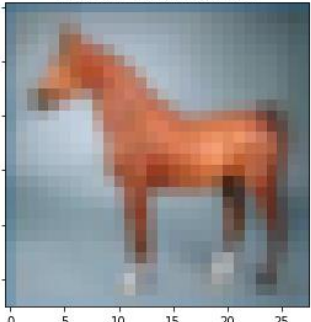
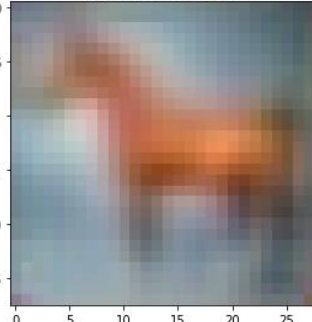


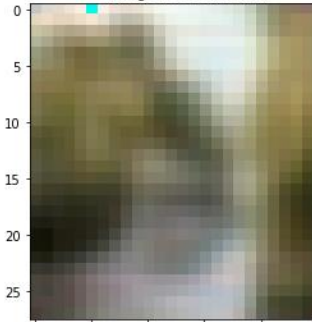
| <b>Dataset MINST</b>  |   |   |
|---|---|---|
| Imagen en baja resolución   | Imagen en alta resolución   | Imagen reconstruida   |
|    |    |    |
| PSNR: 32.25db   | SSIM:0.91   | EPOCAS: 200   |
| <b>Dataset CIFAR</b>  |   |   |
| Imagen en baja resolución   | Imagen en alta resolución   | Imagen reconstruida   |
|  |  |  |
| PSNR: 29.78db   | SSIM: 0.83  | EPOCAS: 200   |
| <b>Dataset DIV2K (Reducida)</b>   |   |   |
| Imagen en baja resolución   | Imagen en alta resolución   | Imagen reconstruida   |
|  |  |  |
| PSNR: 29.204db  | SSIM: 0.8510  | EPOCAS: 300   |

Tabla 10 Resultados SRCNN

En donde se puede observar una parecido en las imágenes reconstruidas a las originales, en la tabla 11 se presentan los resultados de la red generativa antagonica con diferentes épocas:

| <b>Dataset MINST</b>             |                                  |                            |
|----------------------------------|----------------------------------|----------------------------|
| <p>Imagen en baja resolución</p> | <p>Imagen en alta resolución</p> | <p>Imagen reconstruida</p> |
| PSNR: 61.295 db                  | SSIM: 0.71                       | EPOCAS: 200                |
| <b>Dataset CIFAR</b>             |                                  |                            |
| <p>Imagen en baja resolución</p> | <p>Imagen en alta resolución</p> | <p>Imagen reconstruida</p> |
| PSNR: 73.22 db                   | SSIM: 0.8902                     | EPOCAS: 2000               |
| <b>Dataset DIV2K (Reducida)</b>  |                                  |                            |
| <p>Imagen en baja resolución</p> | <p>Imagen en alta resolución</p> | <p>Imagen reconstruida</p> |
| PSNR: 82,5 db                    | SSIM : 0.54                      | EPOCAS 20                  |
| <p>Imagen en baja resolución</p> | <p>Imagen en alta resolución</p> | <p>Imagen reconstruida</p> |

|       |         |       |      |         |     |
|-------|---------|-------|------|---------|-----|
| PSNR: | 52,6 db | SSIM: | 0.65 | EPOCAS: | 800 |
|-------|---------|-------|------|---------|-----|

Tabla 11. Resultados de la red generativa antagonica con diferentes épocas

Los valores denominados **g\_loss** y **d\_loss** son los valores que denotan la pérdida del generador y del generador donde cada época se reduce como se puede observar en la ilustración 28.



```

----- Epoch 199 -----
[0.0032977196, 2.6359477, 0.0006617716]
[0.0033162131, 2.6387074, 0.00067750545]
[0.002978383, 2.3919525, 0.0005864306]
[0.003217537, 2.578226, 0.0006393109]
[0.003151295, 2.5311332, 0.00062016165]
[0.0029905832, 2.4265425, 0.0005640406]
[0.0025956403, 2.1899965, 0.0004056437]
[0.0032312106, 2.5479991, 0.00068321114]
[0.0032889172, 2.6171997, 0.00067171763]
[0.0030198623, 2.3392105, 0.00068065163]
----- Epoch 200 -----

```

Ilustración 28 Perdidas generador y discriminador

En donde a pesar de que por cada bloque de imágenes es diferente, época a época se reduce la perdida. Para finalizar se emplean las métricas con una imagen donde se reduce la resolución y se aumenta empleando el método de interpolación lineal y cubico, así como el resultado de la red, los resultados se pueden visualizar en la tabla 12 donde se reduce la imagen original a la mitad de la resolución y se vuelve a ampliar.

|                             |   |  |
|-----------------------------|---|--|
| <b>Original</b>             |  |  |
| <b>Interpolación Lineal</b> |  | PSNR: 29.95 db<br>SSIM: 0.6884372929161162 |
|                             |   |  |

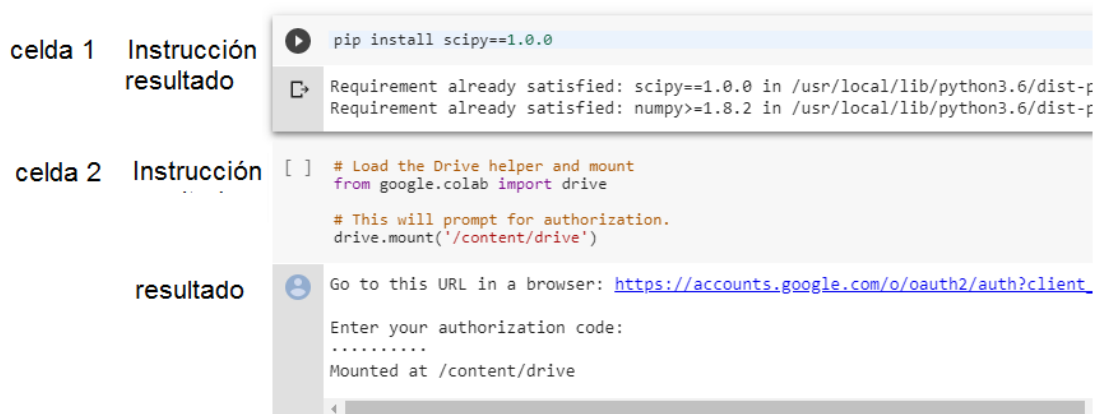
|                             |   |                                |
|-----------------------------|---|--------------------------------|
| <b>Interpolación Cubica</b> |  | PSNR: 29.92 db<br>SSIM: 0.6855 |
| <b>GAN</b>                  |  | PSNR: 29.80 db<br>SSIM: 0.7379 |

**Tabla 12 Comparación con diferentes métodos**

Donde se puede visualizar que la GAN ofrece mejores resultados tanto en las métricas de evaluación como en la comparación visual.

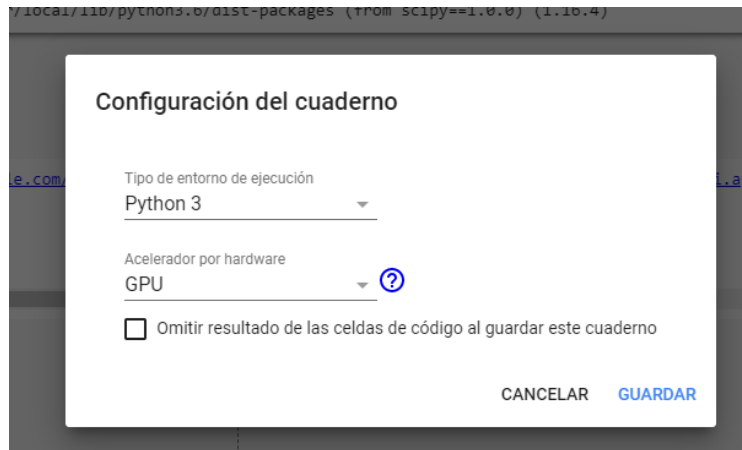
#### 4.2 Google colabory

Google colabory es un notebook de jupyter(entorno interactivo de Python) gratuito que se ejecuta en la nube que cuenta soporte con Google Drive y Git hub el cual ejecuta el código celda a celda como se puede observar en la siguiente ilustración:



**Ilustración 29 Celdas notebook de Jupyter**

También permite ejecutar un entorno de ejecución determinado por la GPU como se puede apreciar en la ilustración 30



**Ilustración 30 Configuración entorno de ejecución Google Colaboratory**

Para emplear Google Colaboratory se creó un nuevo usuario para Gmail con las siguientes credenciales (cuenta que se cerrara al finalizar la investigación) el cual es el siguiente:

Acceso: <https://colab.research.google.com/notebooks/welcome.ipynb>

Usuario: tfm2019srgan@gmail.com

Contraseña: 123SRaHRgan

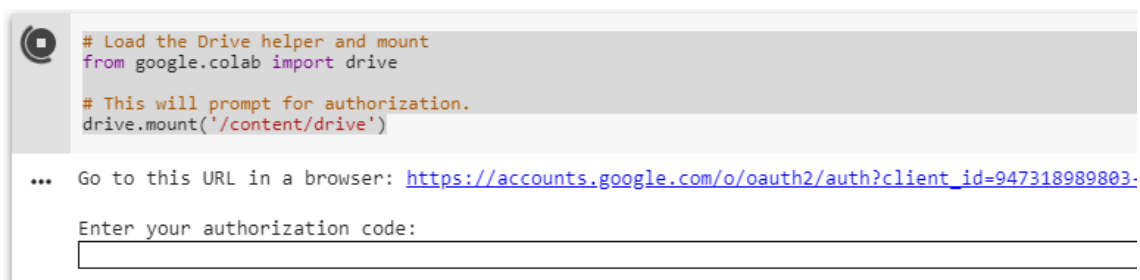
Se creo para utilizar Google Drive con los dataset, usar el clúster, así como el notebook de Jupyter, inicialmente para poder emplear los códigos explicados anteriormente es necesario instalar la primera versión de scipy porque algunas funciones están desactualizadas y si no se hace referencia al uso de la primera versión causa problemas en la ejecución, lo cual se hace con el siguiente código:

```
pip install scipy==1.0.0
```

Posteriormente se procede a solicitar el permiso para usar Google Drive empleando las siguientes instrucciones:

```
from google.colab import drive
drive.mount('/content/drive')
```

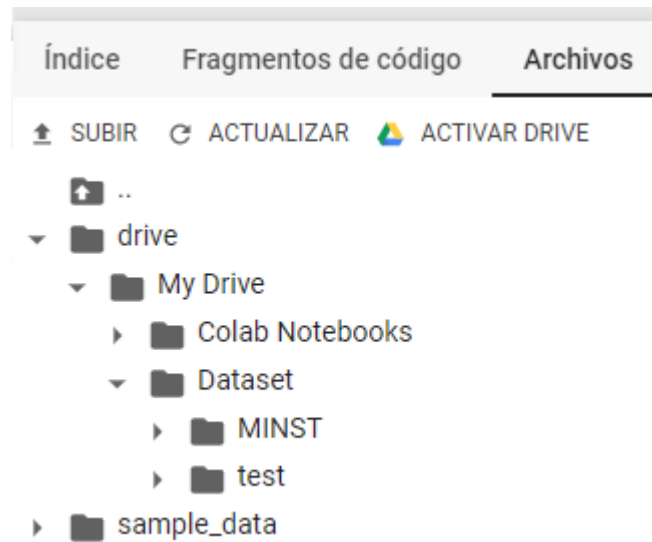
Una vez se ejecuta la celda Google Colaboratory solicita un acceso como se puede visualizar a continuación:



**Ilustración 31 Autorización Google Drive**

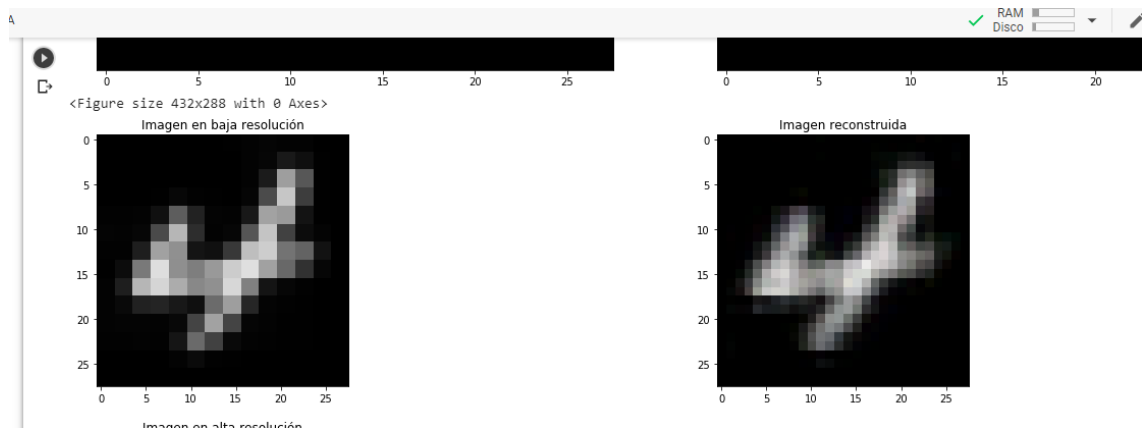
Donde se debe acceder al enlace se solicita la contraseña de la cuenta de Gmail, generando así el respectivo código de autorización, se pega en el

cuadro y con enter se autoriza el ingreso, en donde una vez se ejecuta correctamente el código un gestor de archivos añade el directorio de Drive para poderlo emplear, lo cual se puede visualizar en la ilustración 32.



**Ilustración 32 Archivos de Google Drive**

Una vez se alista el entorno de ejecución y los archivos de Google Drive se procede a ejecutar los respectivos códigos, permitiendo ejecutar el mismo código que en Anaconda mostrando los resultados de la misma manera como se puede observar en la siguiente imagen:



**Ilustración 33 Resultados Google Colaboratory**

Donde se puede observar en la parte superior derecha el uso de la memoria RAM y el disco que es un excelente indicador para tener en cuenta la eficiencia computacional.



## 5. Conclusiones

- El objetivo de diseñar una red generativa antagónica que aumente la resolución de una imagen con un parecido a la real es satisfactorio, en el transcurso de la investigación a pesar de tener dificultades por los recursos computacionales el concepto aplicado a la librería de Keras de realizar una red con los recursos implicó evaluar los conceptos más básicos para hacer un sistema que funcionara con los recursos existentes e hiciera posible el objetivo del trabajo.
- Definitivamente la inteligencia artificial implementada con redes neuronales es un hecho que va a suplir algunas necesidades en un futuro sin embargo es necesario evaluar la cantidad de recursos que puede requerir cualquier implementación y la importancia de un excelente diseño para suplir una necesidad real humana.
- Encontrar información clara sobre el funcionamiento de la GAN o funciones que disminuyeran el tiempo de desarrollo es difícil, sin embargo, existen portales como médium o towards data science que permiten un desarrollo más fácil del tema.
- Se implementaron 3 dataset de los 4 propuestos porque solo aportaba el tamaño invariante y no ofrecía nuevas características a la red generativa antagónica, así también todas las imágenes se redujeron a una resolución menor a  $50 \times 50 \times 3$  porque en el clúster de Google Colaboratory es lo permitido.
- Google Colaboratory es una excelente herramienta porque evita inconvenientes que puede prever una instalación local, que incluso en esta investigación demora más días de lo presupuestado, sin embargo, existe detalles como el control de las variables que no están presentes en este entorno y dificulta modificar los códigos, una combinación entre anaconda con Colaboratory puede crear un complemento muy eficiente.
- La metodología aplicada fue la correcta, sin embargo, la poca documentación del tema así como la cantidad de recursos utilizados no permitió todos los desarrollos planeados como es utilizar una red extractora de características ya existente como la red VGG16 o VGG 19, además estas redes con una resolución de  $200 \times 200 \times 3$  en el clúster de Google Colaboratory no pudieron realizarse, por el tamaño de los tensores, es necesario para futuras implementaciones emplear un clúster y cuantificar la eficiencia computacional.

## 6. Glosario

**Peso:** Son los coeficientes que pueden adaptarse dentro de la red que determinan el coeficiente de la señal de entrada registrada por la neurona artificial.

**Sesgo:** También denominado bias es una neurona que siempre esta activa y permite fijar el peso de las demás neuronas

**Época:** Iteración en la cual se modifican los pesos de las neuronas que realizan el proceso de entrenamiento.

**Clasificador:** Red encargada de categorizar un dato en un tipo, así si el data set es de deportes y la imagen es un balón blanco y negro posiblemente lo categorice en el deporte de futbol.

**Dataset:** Conjunto de información que se usan para entrenar una red neuronal.

**Interpolar:** Aumentar la resolución de una imagen.

**Discriminador:** Red neuronal encargada de determinar cuando una entrada es real o falsa.

**Generador:** Red encargada de generar información a partir de unas apreciaciones previas.

**Filtro:** Operación matemática aplicada a un conjunto de información para obtener una característica de una imagen, así si es una imagen puede obtener características de color, bordes, tamaños, etc.

**Convolución:** Operación matemática que representa la translación de una función sobre otra función, en el contexto de imágenes hace referencia a trasladar una ventana de bits sobre una imagen para extraer características.

**Normalización:** Dividir los bits de una imagen en un intervalo para un futuro procesamiento.

**Activación:** Función matemática aplicada a un conjunto de datos, para categorizar los resultados.

**Precisión:** Dispersión de un conjunto de valores obtenidos de mediciones.

**Capas:** Conexión entre neuronas de una red.

## 7. Bibliografía

- [1] R. Benitez, G. Escudero , S. Kanaan, R. M. David y A. Cencerrado Barraque, inteligencia artificial avanzada, Barcelona: Oberta UOC Publishing, SL, 2018.
- [2] K. Kelly, Descripción de LO INEVITABLE: 12 FUERZAS TECNOLÓGICAS PARA EL FUTURO, New York: Penguin Random House, 2016.
- [3] A. Gulli, Deep learning with Keras.
- [4] J. Torres, Deep Learning – Introducción práctica con Keras.
- [5] «Keras,» [En línea]. Available: <https://keras.io/>.
- [6] technologyreview, «.technologyreview,» [En línea]. Available: <https://www.technologyreview.es/s/10016/el-senor-de-las-gan-el-hombre-que-dio-imaginacion-las-maquinas>.
- [7] github, [En línea]. Available: <https://github.com/eriklindernoren/Keras-GAN>.
- [8] geoffreylitt, [En línea]. Available: <http://geoffreylitt.com/2017/06/04/enhance-upscaling-images-with-generative-adversarial-neural-networks.html>.
- [9] analyticsvidhya, [En línea]. Available: <https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/>.
- [10] heartbeat, [En línea]. Available: <https://heartbeat.fritz.ai/introduction-to-generative-adversarial-networks-gans-35ef44f21193>.
- [11] medium, [En línea]. Available: <https://medium.com/@mattiaspinelli/simple-generative-adversarial-network-gans-with-keras-1fe578e44a87>.
- [12] R. T. Eirikur Agustsson, «NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study,» p. 8, 2017.
- [13] scikit-image.org, [En línea]. Available: <https://scikit-image.org/docs/dev/api/skimage.measure.html#structural-similarity>.
- [14] keras, «keras,» [En línea]. Available: <https://keras.io/applications/>.
- [15] «enmilocalfunciona,» [En línea]. Available: <https://enmilocalfunciona.io/deep-learning-basico-con-keras-parte-3-vgg/>.
- [16] scikit, [En línea]. Available: <https://scikit-learn.org/stable/>.
- [17] kaggle, «<https://www.kaggle.com/datasets>,» [En línea].
- [18] Instituto de Tecnología de California, [En línea]. Available: [http://www.vision.caltech.edu/Image\\_Datasets/Caltech101/](http://www.vision.caltech.edu/Image_Datasets/Caltech101/).
- [19] Universidad Stanford, [En línea]. Available: <http://ufldl.stanford.edu/housenumbers/>.
- [20] S.-H. Tsang, «medium,» [En línea]. Available: <https://medium.com/coinmonks/review-srcnn-super-resolution-3cb3a4f67a7c>.
- [21] J. Brownlee, «machinelearningmastery,» [En línea]. Available: <https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/>.
- [22] J. Brownlee, «machinelearningmastery,» [En línea]. Available:

<https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>.

- [23] D. Godoy, «towardsdatascience,» [En línea]. Available: <https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>.
- [24] S. Sarma, «towardsdatascience,» [En línea]. Available: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>.
- [25] eriklindernoren. [En línea]. Available: <https://github.com/eriklindernoren/Keras-GAN/tree/master/srgan>.

## 8. Anexos

### 8.1 Librería MSE

```
import numpy
import math
import cv2
import matplotlib.pyplot as plt

def psnr(img1, img2):
    mse = numpy.mean( (img1 - img2) ** 2 )
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
#TOMADO DE: https://tutorials.technical.com/how-to-calculate-psnr-value-of-two-images-using-python/
```

### 8.2 Librería carga\_ imágenes

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 4 09:51:54 2019

@author: JHON
"""
from glob import glob
import numpy as np
import matplotlib.pyplot as plt
import scipy
import cv2

def carga_imagenes(n_imagenes=1,dataset_name="",img_res=(128, 128),reduccion=1):

    ruta = glob('./datasets/%s/*' % (dataset_name))    #ruta de carga de los datos
    print("primera ruta:")
    print(ruta[0])
    conjunto_imagenes = np.random.choice(ruta, size=n_imagenes)

    #Definir matrices que guardan las matrices en imagen en baja y alta resolucion
    imgs_hr = []
    imgs_lr = []
    #Obtener automaticamente la resolucion de las imagenes
    img=scipy.misc.imread(ruta[1], mode='RGB')
    size=img.shape
    print("tamaño original")
    print(img.shape)

    for i in conjunto_imagenes:
        img = scipy.misc.imread(i)    #leer la ruta y obtener la imagen

        img_hr = scipy.misc.imresize(img, img_res)    #dejar la imagen original

        nX=img_hr.shape[0]/reduccion #nuevo alto
        nY=img_hr.shape[1]/reduccion #nuevo ancho

        img_lr = scipy.misc.imresize(img, (int(nX), int(nY)))    #reducir la resolucion a un cuarto
        img_lr= cv2.resize(img_lr,img_res, interpolation = cv2.INTER_NEAREST )
        #img_lr = scipy.misc.imresize(img, img_res)

    #agregar todo a una sola matriz
    imgs_hr.append(img_hr)
    imgs_lr.append(img_lr)
```

```

imgs_hr = np.array(imgs_hr)
imgs_lr = np.array(imgs_lr)
#plt.imshow(imgs_hr[0,:,:,:])

plt.figure(figsize=(20,10)) #tamaño de las imagenes

plt.subplot(121)

    plt.title('Imagen en alta resolución') # Ponemos un título
plt.imshow(imgs_hr[0,:,:,:])

plt.subplot(122)
img=scipy.misc.imread(ruta[1], mode='RGB')

plt.title('Imagen en baja resolución') # Ponemos un título
plt.imshow(imgs_lr[0,:,:,:])

plt.savefig("HRtoLR")

from PIL import Image

imagen = Image.open("HRtoLR.png")
imagen.show()

size_lr=[imgs_lr.shape[1], imgs_lr.shape[2],3]# Obtener el tamaño
size_hr=[imgs_hr.shape[1], imgs_hr.shape[2],3]# Obtener el tamaño

return imgs_hr[:,:,:,:], imgs_lr[:,:,:,:], size_lr , size_hr

print(conjunto_imagenes)

```

### 8.3 Código red extractora de características

```

from carga_imagenes import carga_imagenes
import numpy as np
from keras.models import Sequential, Model
from keras.layers import Dense, Conv2D, Activation, Input
from keras import optimizers
import matplotlib.pyplot as plt
from metricas.MSE import psnr #importa metrica
from skimage.measure import compare_ssim
import numpy as np

def lista_div(n=8):

    lista = []
    for i in range (1, n):
        if (n%i == 0):
            lista.append(i)
    return lista

def construir_red(tam=(7,7,1)):

    LB=lista_div(tam[0])
    LH=lista_div(tam[1])
    LB.reverse()
    LH.reverse()
    print(LB)
    srcnn=Sequential()
    # Con 64 neuronas se hace el procesamiento con una ventana grande
    srcnn.add(Conv2D(filters = 64, kernel_size = (16,16),padding = 'Same',
                    activation = 'relu', input_shape = tam))

```

```

# No se toman en cuenta los 2 ultimos numeros porque es el 1 y el 2 que se
#afuera del ciclo

#for i in range (1, (len(LB)-2)):
#   srcnn.add(Conv2D(filters = 32, kernel_size = (LB[i],LH[i]),padding = 'Same',
#       activation ='relu'))

srcnn.add(Conv2D(filters = 32, kernel_size = (8,8),padding = 'Same',
    activation ='relu'))
srcnn.add(Conv2D(filters = 3, kernel_size = (2,2),padding = 'Same',
    activation ='relu'))

opt = optimizers.Adam(lr=0.001)
srcnn.compile(optimizer=opt,loss='mean_squared_error')
srcnn.summary()

return srcnn

if __name__ == '__main__':

    escala=2
    nimagenes=100
    tam=(28,28)
    paso=20 # tamaño de la ventana
    imgs_hr, imgs_lr,tam_lr,tam_hr=carga_imagenes(nimagenes,"minst",tam,escala)
    Div_en=80 # % porcentaje que divide a las imagenes de entrenamiento y validacion
    plt.figure()

    X_train = []
    Y_train = []
    X_val = []
    Y_val = []

    # imagenes para entrenamiento añadiendolas a una matriz
    # Con el formato de entrada a la red

    for i in range(0,int(nimagenes*Div_en/100)):
        vx = imgs_hr[i,:,:]/255
        vy = imgs_lr[i,:,:]/255
        h = vy.shape[0]
        w = vy.shape[1]

        for x in range(0, h-tam[0]+1, paso):
            for y in range(0, w-tam[1]+1, paso):
                tx = vx[x:x+tam[0], y:y+tam[1,:]].reshape(tam[0],tam[1],3)
                ty = vy[x:x+tam[0], y:y+tam[1,:]].reshape(tam[0],tam[1],3)
                X_train.append(tx)
                Y_train.append(ty)

    # imagenes para validacion añadiendolas a una matriz
    # Con el formato de entrada a la red

    for i in range(int(nimagenes*Div_en/100),nimagenes):
        vy = imgs_hr[i,:,:]/255
        vx = imgs_lr[i,:,:]/255
        h = vy.shape[0]
        w = vy.shape[1]

        for x in range(0, h-tam[0]+1, paso):
            for y in range(0, w-tam[1]+1, paso):
                tx = vx[x:x+tam[0], y:y+tam[1,:]].reshape(tam[0],tam[1],3)
                ty = vy[x:x+tam[0], y:y+tam[1,:]].reshape(tam[0],tam[1],3)
                X_val.append(tx)
                Y_val.append(ty)

```

```

# Prepara las imagenes para entrenar la red neuronal
X_train = np.array(X_train)
Y_train = np.array(Y_train)
X_val = np.array(X_val)
Y_val = np.array(Y_val)

# Cosntruccion de la SRCNN el valor de ingreso es el tamaño de la imagen
srcnn=construir_red(tx.shape)

# Entrenamiento de la red

srcnn.fit(X_train, Y_train, batch_size = 20, epochs = 60, validation_data=(X_val, Y_val))
srcnn.save('srgan.h5')

# Imagen generada por la red

prueba = X_train[0,:,:,:]
prueba= prueba.reshape(1,tam[0],tam[1],3)
generada=srcnn.predict(prueba)

# Adecuar imagen para ser visualizada
generada = generada.reshape(tam[0],tam[1],3)
generada= (generada *255)
generada= generada.astype(np.uint8)

# Metricas de evaluacion
original=imgs_hr[0,:,:]
PSNR=psnr(original,generada)
SSIM = compare_ssim(original,generada,multichannel=True )

print("Metricas:")
print(PSNR)
print(SSIM)

#####visualizar el resultado#####
plt.figure(figsize=(20,10)) #tamaño de las imagenes

plt.subplot(221)

plt.title('Imagen en baja resolución') # Ponemos un título
plt.imshow(imgs_lr[0,:,:,:],cmap='gray')
plt.savefig("LR")
plt.subplot(222)

plt.imshow(generada,cmap='gray')

plt.title('Imagen reconstruida') # Ponemos un título

plt.subplot(223)

plt.title('Imagen en alta resolución') # Ponemos un título
plt.imshow(imgs_hr[0,:,:,:],cmap='gray')
t = ("Metricas de medicion")
t1 = ("PSNR   : "+str(PSNR))
t2 = ("SSIM   : "+str(SSIM))

plt.text(50, 1, t, ha='left', rotation=0,fontsize=15, wrap=True)
plt.text(50, 3, t1, ha='left', rotation=0,fontsize=15, wrap=True)
plt.text(50, 5, t2, ha='left', rotation=0,fontsize=15, wrap=True)
plt.savefig("HRtoLR")

from PIL import Image

imagen = Image.open("HRtoLR.png")

```



```

imagen.show()

plt.figure()
plt.imshow(imgs_lr[0,:,:,:],cmap='gray')
plt.savefig("LR")

plt.figure()
plt.imshow(generada,cmap='gray')
plt.savefig("R")

```

## 8.4 Código GAN

```

from keras.models import Sequential
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.layers import Conv2D, Input, MaxPool2D, Reshape, Activation, Flatten, Dense, Dropout
from keras.optimizers import adam
import matplotlib.pyplot as plt
from keras.applications.vgg19 import VGG19
import keras.backend as K
from keras.layers.advanced_activations import LeakyReLU
from keras.models import Model, Sequential
import numpy as np
from keras.layers.normalization import BatchNormalization
from keras.models import load_model
from keras.layers import Concatenate, Dense, LSTM, Input, concatenate
import tensorflow as tf
from keras.layers import BatchNormalization, Activation, LeakyReLU, Add, Dense

def construir_vgg(size=(7,7,1)):

    path_model = 'srgan.h5'

    input_shape = size

    vgg = load_model(path_model)
    vgg.summary()
    vgg.outputs = [vgg.layers[2].output]

    input_layer = Input(shape=input_shape)

    features = vgg(input_layer)

    model = Model(inputs=[input_layer], outputs=[features])
    return model

def crear_generador(size=(7,7,1),escala=1):

    generator=Sequential()

    generator.add(Dense(units=256,input_shape = size))
    generator.add(Flatten())
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=512))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=1024))
    generator.add(LeakyReLU(0.2))

```

```

generator.add(Dense(units=2352, activation='sigmoid'))
generator.add(Reshape((28,28,3)))
generator.summary()

return generator

def crear_discriminador(size=(7,7,1)):

    discriminator=Sequential()
    discriminator.add(Dense(units=1024,input_shape = size))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units=512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units=256))
    discriminator.add(LeakyReLU(0.2))

    discriminator.add(Dense(units=3, activation='sigmoid'))
    discriminator.summary()

    return discriminator

if __name__ == '__main__':

    #####Declarar variables  inicializar las variables #####
    epocas=200
    nimagenes=100
    loteimagenes=10
    optimizador=adam(lr=0.0002, beta_1=0.5)
    escala=2
    #Carga las imagenes de los dataset, y obtiene las imagenes en alta y baja resolucio

    imgs_hr, imgs_lr,size_lr,size_hr=carga_imagenes(nimagenes,"MINST",(28,28),escala)
    imgs_hr=imgs_hr/255
    imgs_lr=imgs_lr/255
    #Crear generador y discriminador

    generador= crear_generador(size_lr,escala)
    discriminador = crear_discriminador(size_hr)

    #####Creacion mi vgg #####

    vgg=construir_vgg(size_hr)
    vgg.trainable = False
    vgg.compile(loss='mse', optimizer=optimizador, metrics=['accuracy'])
    discriminador.compile(loss="binary_crossentropy",optimizer=optimizador)

    #Prueba del generador, acomodacion matrices entrada red neuronal
    imgs_lr=imgs_lr.reshape(imgs_lr.shape[0],imgs_lr.shape[1],imgs_lr.shape[2],3)
    imgs_hr=imgs_hr.reshape(imgs_hr.shape[0],imgs_hr.shape[1],imgs_hr.shape[2],3)
    generated_images = generador.predict(imgs_lr)

    #####CREACION DE LA GAN #####
    #Entrada de la GAN

```

```

input_high_resolution = Input(shape=size_hr)
input_low_resolution = Input(shape=size_lr)

# Generar version en una alta resolucion
generated = generador(input_low_resolution)

# Extraer características
features = vgg(generated)

discriminador.trainable = False

# Generar la probabilidad de que la imagen sea falsa o real
probs = discriminador(generated)

# Crear y compilar la red
adversarial_model = Model([input_low_resolution, input_high_resolution], [probs, features])
adversarial_model.compile(loss=['binary_crossentropy', 'mse'], loss_weights=[1e-3, 1],
optimizer=optimizador)

#####reacomodacion para generar la imagen#####
imgen = generated_images.reshape(nimagenes,size_hr[0],size_hr[1],3)
plt.figure()
plt.imshow(imgen[1,:,:]) #prueba
#####

for e in range(1, epocas+1):
    print('-'*15, 'Epoch %d' % e, '-'*15)
    for _ in range(loteimagenes):

        #Obtener 10 imagenes aleatoria de la cantidad de imagenes cargadas
        rand_nums = np.random.randint(0, imgs_hr.shape[0], size=loteimagenes)
        lote_hr = imgs_hr[rand_nums]
        lote_lr = imgs_lr[rand_nums]

        #Prediccion del lote de baja resolucion
        generated_images_sr = generador.predict(lote_lr)

        #matrices con valores cercanos a 1 para la validez, y con 0 para los que no son validos
        real_data_Y = np.ones((loteimagenes, 28, 28, 3)) # Acomodar automaticamente
        fake_data_Y = np.zeros((loteimagenes, 28, 28, 3)) # Acomodar automaticamente

        #Etapa para entrenar el discriminador
        discriminador.trainable = True

        #Adecuar entrada para el discriminador

        d_loss_real = discriminador.train_on_batch(lote_hr, real_data_Y)
        d_loss_fake = discriminador.train_on_batch(generated_images_sr, fake_data_Y)
        d_loss = 0.5 * np.add(d_loss_fake, d_loss_real)

        #Nuevas imagenes para entrenar la red
        rand_nums = np.random.randint(0, imgs_hr.shape[0], size=loteimagenes)
        lote_hr = imgs_hr[rand_nums]
        lote_lr = imgs_lr[rand_nums]

        #Vector con ganancias de la red cercanas al 1
        gan_Y = np.ones((loteimagenes, 28, 28, 3)) # Acomodar automaticamente

        #Dejar de entrenar el discriminador

```

```

discriminador.trainable = False

#Caracteristicas de las imagenes

features = vgg.predict(lote_hr)

#Entrenamiento de la GAN
g_loss = adversarial_model.train_on_batch([lote_lr, lote_hr],
                                           [gan_Y, features])
print(g_loss)

print("fin del ciclo")
#####visualizar el resultado#####
plt.figure(figsize=(20,10)) #tamaño de las imagenes

plt.subplot(221)

plt.title('Imagen en baja resolución') # Ponemos un título
plt.imshow(imgs_lr[0,:,:,:],cmap='gray')

plt.subplot(222)
seg=imgs_lr[0,:,:,:]
seg=seg.reshape(1,imgs_lr.shape[1],imgs_lr.shape[2],3)
reconstruida = generador.predict(seg)
reconstruida=reconstruida.reshape(size_hr[0],size_hr[1],3)
plt.imshow(reconstruida,cmap='gray')

plt.title('Imagen reconstruida') # Ponemos un título

plt.subplot(223)

plt.title('Imagen en alta resolución') # Ponemos un título
plt.imshow(imgs_hr[0,:,:,:],cmap='gray')

plt.savefig("HRGAN")

```