APLICACIÓN DE LA ARQUITECTURA DE REDES NEURONALES (cGANS) PARA GENERAR IMÁGENES DE FACHADAS UTILIZANDO EL MODELO PIX2PIX.

Ing. Brian Santeliz. Universidad Valle Del Momboy

santelizb@uvm.edu.ve

Trujillo 3101, Venezuela

Recibido julio 2020 Aceptado octubre 2020

RESUMEN:

El problema que se intenta solucionar en la presente investigación es el Aprovechamiento eficiente del uso de las redes neuronales. El creciente uso de la informática hace que las empresas almacenen grandes cantidades de información que en general no son aprovechadas, sea por desconocimiento del valor de la misma o de las técnicas para poder sacarle provecho. Con técnicas como las redes neuronales no solo se puede inferir y predecir el futuro de una empresa en el tiempo, sino que también podría crear modelos de aprendizaje automático para optimizar funciones específicas y mejorar la calidad de los productos.
El uso de la arquitectura cGans y del modelo pix2pix permite resolver múltiples problemas que serán mencionados a trevés de esta investigación.

PALABRAS CLAVE: Redes Neuronales, Información, empresas, Modelos de Aprendizaje Automático, Arquitectura cGans, Modelo pix2pix.

APPLICATION OF THE NEURONAL NETWORKS ARCHITECTURE (cGANS) TO GENERATE IMAGES OF FACADES USING THE PIX2PIX MODEL.

ABSTRACT

The problem to be solved in this research is the efficient use of neural networks. The growing use of information technology means that companies store large amounts of information that are generally not used, either due to lack of knowledge of its value or of the techniques to be able to take advantage of it. With techniques such as neural networks, not only can the future of a company be inferred and predicted over time, but it could also create machine learning models to optimize specific functions and improve the quality of products.
The use of the cGans architecture and the pix2pix model allows solving multiple problems that will be mentioned throughout this research.

Planteamiento de la situación problemática.

Son pocas las empresas que analizan dicha información en búsqueda de patrones o conclusiones útiles para el negocio. En caso de hacerlo, en general se utilizan algoritmos lineales sin realimentación ni “aprendizaje”, los cuales insumen más tiempo y requieren mayor preparación previa y análisis posterior, arrojando resultados menos eficientes. El problema que se intenta solucionar en el presente trabajo es el Aprovechamiento eficiente del e las redes neuronales, El creciente uso de la informática hace que las empresas almacenen grandes cantidades de información que en general no son

aprovechadas, sea por desconocimiento del valor de la misma o de las técnicas para poder sacarle provecho. Con técnicas como las redes neuronales no solo se puede inferir y predecir el futuro de una empresa en el tiempo, sino que también podría crear modelos de aprendizaje automático para optimizar funciones específicas y mejorar la calidad de los productos.

El uso de la arquitectura cGans y del modelo pix2pix permite resolver múltiples problemas que serán mencionados posteriormente, entre ellos se encuentra. Crear nuevos escenarios que pueden ser utilizados en área de entrenamiento (cine, videojuegos). Uno de los ejemplos más famosos es el caso de la marca de hardware popularmente conocida que tiene por nombre Nvidia ellos hicieron uso de esta arquitectura para crear un modelo denominado “GauGan” este modelo de Deep learning permite dibujar un boceto de un paisaje y se transforma en una imagen realista con el estilo del pintor “Paul Gauguin” de allí el nombre de este modelo.

Al igual que existe una página dedica a imprimir en pantalla imágenes de personas que fueron generados artificialmente utilizando Redes generativas adversarias. Otra de las aplicaciones de este tipo de arquitectura son las siguientes, usando la imagen de un espectrograma convertir el sonido de un instrumento como el de un cuatro a otro sonido de un instrumento distinto como al de un piano. Cambiar el estilo de una imagen. Utilizando los bordes de una imagen generar la imagen completa. Así como esta son muchas las aplicaciones que se pueden generar aplicando pix2pix.

El propósito de esta investigación es desarrollar un modelo de redes generativas adversarias condicionadas correctamente para sintetizar imágenes de fachadas de edificios basándose en un mapa de segmentación.

Objetivo general:

Implementar el modelo Pix2Pix para generar una imagen de salida (output) a partir de una imagen de entrada (input) tomando como entrada un mapa de segmentación.

Objetivos específicos

1. Utilizar el potencial del modelo Pix2Pix y de las GANs para crear avances a futuro en las generaciones de imágenes.

2. Desarrollar el modelo para producir nuevos escenarios de videojuegos, es decir convertirse en una gran herramienta para la creación de mundos virtuales,nuevos rostros para el área de entretenimiento, combatir amenazas como los Deep fakes y aplicaciones que la utilizan de una forma incorrecta, como Deep nude que hace uso de las GANs.

3. Resolver múltiples problemas en el procesamiento de imagen, tener como entrada un mapa de segmentación y obtener su salida (caso de estudio), transformar una imagen en blanco y negro y obtener la imagen a color, cambiar una imagen realizando modificaciones en su estilo, obtener una imagen completa solo con los bordes de un objeto.

4. Utilizar el modelo pix2pix y el Deep learning para contribuir en el campo de image processing que hacen uso los coches autónomos en situaciones donde las vías pueden presentar problemas de visibilidad y entorpecer la cámara que utilizan para conducir (condiciones de lluvia y niebla).

Justificación

La presente investigación expone una relevante conjetura por estar basada en referentes teóricos específicos que indagan en el tema objeto de estudio. De la misma manera tiene como propósito servir como antecedentes para nuevas investigaciones, tomando en cuenta su importancia en el campo del machine learning y la Ciencia de datos, los cuales van a la par de los cambios exigidos y la continua evolución de la tecnología incluyendo su impacto en la sociedad. De esta manera, tomando en cuenta la importancia y relevancia teórica de la tesis contribuye al aporte de futuras investigaciones y al avance en este campo.

En cuanto al aspecto metodológico, se ofrecerán en forma de cuestionarios confiables, validados por personal capacitado para tal fin, que puedan servir de apoyo y referencia a otras investigaciones en el mismo campo de acción de las variables que sustenta este estudio. Como son el Dataset de

imágenes para el entrenamiento del algoritmo de ML y la aplicación de la arquitectura de redes neuronales generativas adversarias condicionadas.

De igual manera, este tipo de investigación presenta relevancia, ya que generará información importante para el crecimiento de los algoritmos supervisados y no supervisados, como también métodos de optimización probabilísticos no determinista y modulares entre los cuales se encuentra el algoritmo de Backpropagation, Descenso del gradiente estocástico, funciones de activación y el prototipo de redes neuronales, que sin duda ayudara a enfrentar los problemas relacionados al área de Inteligencia Artificial y así favorecer al perfeccionamiento del Machine learning y Deep learning.

Por otro lado se hace necesario darle relevancia a la enseñanza de este campo de la computación el cual tiene un gran impacto en la sociedad y muchos de estos algoritmos son utilizados a nivel global en aspectos que van desde la medicina hasta la física cuántica. De esta manera la presente investigación puede ofrecer el aporte a los conceptos más importantes del aprendizaje automático. Al dejar en evidencia la necesidad académica que se presenta. De modo que se cuente con portadores de información, talleres y personal capacitado en el área basado en los parámetros establecidos.

En la presente investigación se utilizó un tipo de investigación proyectiva.
Según Hurtado (2000), “consiste en la elaboración de una propuesta o de un modelo, como solución a un problema o necesidad de tipo práctico, ya sea de un grupo social, o de una institución, en un área particular del conocimiento, a partir de un diagnóstico preciso de las necesidades del momento, los procesos explicativos o generadores involucrados y las tendencias futuras”. (p.325).

Efectivamente, la propuesta de la elaboración de una arquitectura de redes generativas adversarias condicionadas, permitirá orientar y entender como es el funcionamiento de uno de los modelos de Deep learning generativos más conocidos y aplicables en el mercado, instituyendo en las bases para la generación del conocimiento para que en próximos trabajos estos puedan ser aprovechados y usados en cualquier área donde puedas necesitar más contenido generado por inteligencia artificial.

Del mismo modo, es importante establecer que la investigación presenta

Un enfoque holístico bajo el cual, a globalidad está dada por la unión sintagmática de los diversos paradigmas (cualitativo- Cuantitativo), donde el todo es más que las sumas de sus partes.

La investigación Holística surge como una necesidad de proporcionar criterios de apertura y una metodología más completa y efectiva a las personas que realizan investigación en las diversas áreas del conocimiento. Es una propuesta que presenta la investigación como un proceso global, evolutivo, integrador, concatenado y organizado. (p.20).

Diseño de la Investigación

Esta investigación se encuentra dentro del diseño de campo experimental. Dado que se observan los datos, hechos, situaciones o sujetos en su ambiente o realidad y se alteran por el investigador.

Según el autor Santa palella y feliberto Martins (2010), define: El diseño experimental es aquel según el cual el investigador manipula una variable experimental no comprobada, bajo condiciones estrictamente controladas. Su objetivo es describir de qué modo y porque causa se produce o puede producirse un fenómeno. Busca predecir el futuro, elaborar pronósticos que una vez confirmados, se convierten en leyes y generalizaciones tendentes a incrementar el cúmulo de conocimientos pedagógicos y el mejoramiento de la acción educativa.

Según la perspectiva temporal, la investigación es de tipo evolutivo contemporáneo, dado que los datos obtenidos se analizaron en el momento actual para de esta forma diseñar la propuesta del modelo de línea de investigación en área contable acorde a la realidad del contexto. Asimismo, se considera que la misma, de ser formalizada por la institución, tendrá continuidad en el tiempo, ya que abrirá oportunidades de investigación coordinada, pertinente y ajustada al área social en la cual está inserta.

El estudio es descriptivo ya que se analizó la situación actual de las unidades de producción, desde el punto de vista de su situación contable. Tamayo (1990), al referirse al estudio descriptivo, la define como “aquel que comprende la descripción, análisis e interpretación de la naturaleza actual, composición o procesos de los fenómenos” (p.36).

Obtención de datos

En el desarrollo del algoritmo utilicé el dataset llamado CMP Base de datos de Fachadas, proporcionado por el centro para la percepción de imágenes de la Universidad Czech Técnica en Praga. En función de mantener el ejemplo sencillo, use la copia pre procesada de este dataset, el cual contiene cada imagen está constituida por un par donde tiene su representación en el mapa de segmentación y la misma imagen de la fachada, este conjunto de datos fue el que usaron autores del paper de Pix2Pix. Cada imagen tiene un ancho y un alto de 256 por 256 pixeles, un buffer y batch size de 400 y 1. Para el conjunto de entrenamiento se utilizaron 400 imágenes y para el conjunto de prueba 100 imágenes, así mismo se utilizó una función llamada **ramdom jitter** el cual la imagen se cambia en tamaño a 286x286 y lego se recorta de manera aleatoria a 256x256. La función **random mirroring** tiene como objetivo voltear aleatoriamente la imagen de manera horizontal de izquierda a derecha para de esta forma aumentar el dataset de entrenamiento y simular más datos.

Definición operacional de las variables

Variable 1

Aplicación de la arquitectura de redes neuronales cGANs

Definición Conceptual

Las redes neuronales son un tipo de algoritmo computacional que forma parta del Deep Learning. Las redes cGANs son una variación que nacieron en el año 2018 gracias al poder computacional exponencial. Estas redes tienen como objetivo generar nuevo contenido ya que forman parte de las redes generativas adversarias con la variación de que el input está condicionado a un vector de etiquetas para así obtener un resultado más específico.

Definición Operacional

En relación a la primera variable de estudio Aplicación de la arquitectura de redes neuronales cGANs, hace mención a la arquitectura del generador y discriminador el cual está programada en la librería para Machine Learning Tensorflow 2.0 y el objetivo es generar imágenes de fachadas de edificios condicionadas a un input que es el mapa de segmentación o el boceto de la imagen que tiene como entrada.

Variable 2

Generar imágenes utilizando el modelo Pix2Pix

Definición Conceptual

PixPix es un modelo que tiene como enfoque entrenar una red del tipo cGANs, diseñado para tareas de traducción de imagen a imagen, el enfoque fue presentado en CVPR en 2017. El modelo discriminador se actualiza directamente, mientras que el modelo generador se actualiza a través del modelo discriminador. Como tal, los dos modelos se entrenan simultáneamente en un proceso de confrontación en el que el generador busca engañar mejor al discriminador y el discriminador busca identificar mejor las imágenes falsificadas.

Definición Operacional

La Pix2Pix GAN se ha demostrado en una variedad de tareas de traducción de imagen a imagen, como la conversión de mapas a fotografías satelitales, fotografías en blanco y negro a color y bocetos de productos en fotografías de productos. En este caso fachadas de edificios en un boceta a fachadas reales.

Arquitectura del modelo

En esta sección explicare como está construido modelo de red neuronal para generar imágenes de fachadas empezando por las librerías básicas y el dataset. El código es una modificación del modelo que se encuentra en la documentación de TensorFlow.

Importando TensorFlow y otras bibliotecas

```
from __future__ import absolute_import, division, print_function,
unicode_literals
```

```
import tensorflow as tf

import os
import time

from matplotlib import pyplot as plt
fromIPythonimport display
```

```
pip install -q -U tensorboard
```

Cargando el conjunto de datos

Como se menciona en el documento, se aplicaron fluctuaciones y reflejos aleatorios al conjunto de datos de entrenamiento.

- En fluctuaciones aleatorias, la imagen cambia de tamaño `286 x 286`y luego se recorta aleatoriamente a`256 x 256`
- En la duplicación aleatoria, la imagen se voltea aleatoriamente horizontalmente, es decir, de izquierda a derecha.

```
_URL
='https://people.eecs.berkeley.edu/~tinghuiz/projects/pix2pix/datasets/facades
.tar.gz'

path_to_zip = tf.keras.utils.get_file('facades.tar.gz',
                                      origin=_URL,
                                      extract=True)

PATH = os.path.join(os.path.dirname(path_to_zip),'facades/')
```

```
Descarga de datos de
https://people.eecs.berkeley.edu/~tinghuiz/projects/pix2pix/dataset
s/facades.tar.gz
30171136/30168306 [==============================] - 2s 0us / paso
```

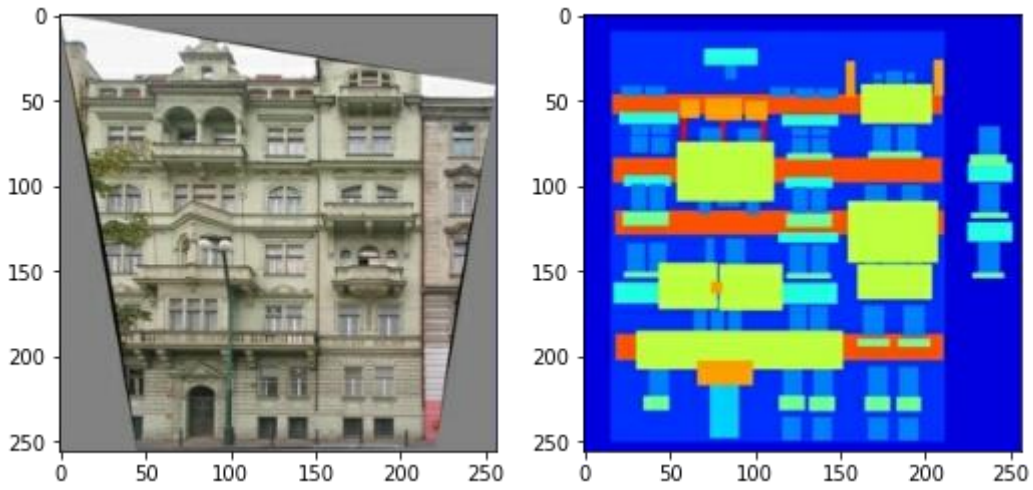
```
BUFFER_SIZE =400
BATCH_SIZE =1
```

```
IMG_WIDTH =256
IMG_HEIGHT =256
```

```
def load(image_file):
  image = tf.io.read_file(image_file)
  image = tf.image.decode_jpeg(image)

  w = tf.shape(image)[1]

  w = w // 2
  real_image = image[:,:w,:]
  input_image = image[:, w:,:]

  input_image = tf.cast(input_image, tf.float32)
  real_image = tf.cast(real_image, tf.float32)

  return input_image, real_image
```

```
inp, re = load(PATH+'train/100.jpg')
# casting to int for matplotlib to show the image
plt.figure()
plt.imshow(inp/255.0)
plt.figure()
plt.imshow(re/255.0)
```

```
<matplotlib.image.AxesImage en 0x7f0fa1829748>
```



```
def resize(input_image, real_image, height, width):
  input_image = tf.image.resize(input_image,[height, width],

method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
  real_image = tf.image.resize(real_image,[height, width],

 method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)

  return input_image, real_image
```

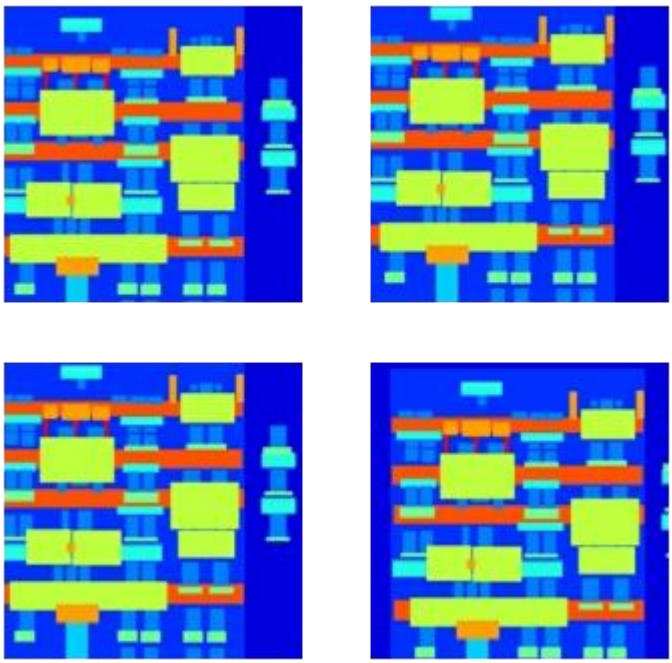
```
def random_crop(input_image, real_image):
  stacked_image = tf.stack([input_image, real_image], axis=0)
  cropped_image = tf.image.random_crop(
      stacked_image, size=[2, IMG_HEIGHT, IMG_WIDTH,3])

  return cropped_image[0], cropped_image[1]
```

```
# normalizing the images to [-1, 1]

def normalize(input_image, real_image):
  input_image =(input_image /127.5)-1
  real_image =(real_image /127.5)-1

  return input_image, real_image
```

```
@tf.function()
def random_jitter(input_image, real_image):
  # resizing to 286 x 286 x 3
  input_image, real_image = resize(input_image, real_image,286,286)

  # randomly cropping to 256 x 256 x 3
  input_image, real_image = random_crop(input_image, real_image)

  if tf.random.uniform(())>0.5:
    # random mirroring
    input_image = tf.image.flip_left_right(input_image)
    real_image = tf.image.flip_left_right(real_image)

  return input_image, real_image
```

Como puede ver en las imágenes a continuación, están pasando por fluctuaciones aleatorias Las fluctuaciones aleatorias como se describe en el documento es para

- Cambiar el tamaño de una imagen a mayor altura y ancho
- Recortar aleatoriamente al tamaño objetivo
- Voltear aleatoriamente la imagen horizontalmente



```
def load_image_train(image_file):
  input_image, real_image = load(image_file)
  input_image, real_image = random_jitter(input_image, real_image)
  input_image, real_image = normalize(input_image, real_image)

  return input_image, real_image
```

```
def load_image_test(image_file):
  input_image, real_image = load(image_file)
  input_image, real_image = resize(input_image, real_image,
                                   IMG_HEIGHT, IMG_WIDTH)
  input_image, real_image = normalize(input_image, real_image)

  return input_image, real_image
```

Input Pipeline

```
train_dataset = tf.data.Dataset.list_files(PATH+'train/*.jpg')
train_dataset = train_dataset.map(load_image_train,
```

```
num_parallel_calls=tf.data.experimental.AUTOTUNE)
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.batch(BATCH_SIZE)
```

```
test_dataset = tf.data.Dataset.list_files(PATH+'test/*.jpg')
test_dataset = test_dataset.map(load_image_test)
test_dataset = test_dataset.batch(BATCH_SIZE)
```

Construyendo al Generador

- La arquitectura del generador es una U-Net modificada.
- Cada bloque en el codificador es (Conv -> Batchnorm -> Leaky ReLU)
- Cada bloque en el decodificador es (Transposed Conv -> Batchnorm -> Dropout (aplicado a los primeros 3 bloques) -> ReLU)
- Hay conexiones de omisión entre el codificador y el decodificador (como en U-Net).

```
OUTPUT_CHANNELS =3
```

```
def downsample(filters, size, apply_batchnorm=True):
  initializer = tf.random_normal_initializer(0.,0.02)

  result = tf.keras.Sequential()
  result.add(
      tf.keras.layers.Conv2D(filters, size, strides=2,
padding='same',
                             kernel_initializer=initializer,
use_bias=False))

  if apply_batchnorm:
    result.add(tf.keras.layers.BatchNormalization())

  result.add(tf.keras.layers.LeakyReLU())

  return result
```

```
down_model = downsample(3,4)
down_result = down_model(tf.expand_dims(inp,0))
print(down_result.shape
```

```
(1, 128, 128, 3)
```

```
def upsample(filters, size, apply_dropout=False):
  initializer = tf.random_normal_initializer(0.,0.02)

  result = tf.keras.Sequential()
  result.add(
    tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
                                    padding='same',
                                    kernel_initializer=initializer,
                                    use_bias=False))

  result.add(tf.keras.layers.BatchNormalization())

  if apply_dropout:
      result.add(tf.keras.layers.Dropout(0.5))

  result.add(tf.keras.layers.ReLU())

  return result
```

```
up_model = upsample(3,4)
up_result = up_model(down_result)
print(up_result.shape)
```

```
(1, 256, 256, 3)
```

```
defGenerator():
  inputs = tf.keras.layers.Input(shape=[256,256,3])

  down_stack =[
    downsample(64,4, apply_batchnorm=False),# (bs, 128, 128, 64)
    downsample(128,4),# (bs, 64, 64, 128)
    downsample(256,4),# (bs, 32, 32, 256)
    downsample(512,4),# (bs, 16, 16, 512)
    downsample(512,4),# (bs, 8, 8, 512)
    downsample(512,4),# (bs, 4, 4, 512)
    downsample(512,4),# (bs, 2, 2, 512)
    downsample(512,4),# (bs, 1, 1, 512)
  ]

  up_stack =[
    upsample(512,4, apply_dropout=True),# (bs, 2, 2, 1024)
```

```
    upsample(512,4, apply_dropout=True),# (bs, 4, 4, 1024)
    upsample(512,4, apply_dropout=True),# (bs, 8, 8, 1024)
    upsample(512,4),# (bs, 16, 16, 1024)
    upsample(256,4),# (bs, 32, 32, 512)
    upsample(128,4),# (bs, 64, 64, 256)
    upsample(64,4),# (bs, 128, 128, 128)
  ]

  initializer = tf.random_normal_initializer(0.,0.02)
  last= tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS,4,
                                        strides=2,
                                        padding='same',

 kernel_initializer=initializer,
                                        activation='tanh')# (bs,
256, 256, 3)

  x = inputs

  # Downsampling through the model
  skips =[]
  for down in down_stack:
    x = down(x)
    skips.append(x)

  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    x = tf.keras.layers.Concatenate()([x, skip])

  x =last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

```
generator =Generator()
tf.keras.utils.plot_model(generator, show_shapes=True, dpi=64)
```

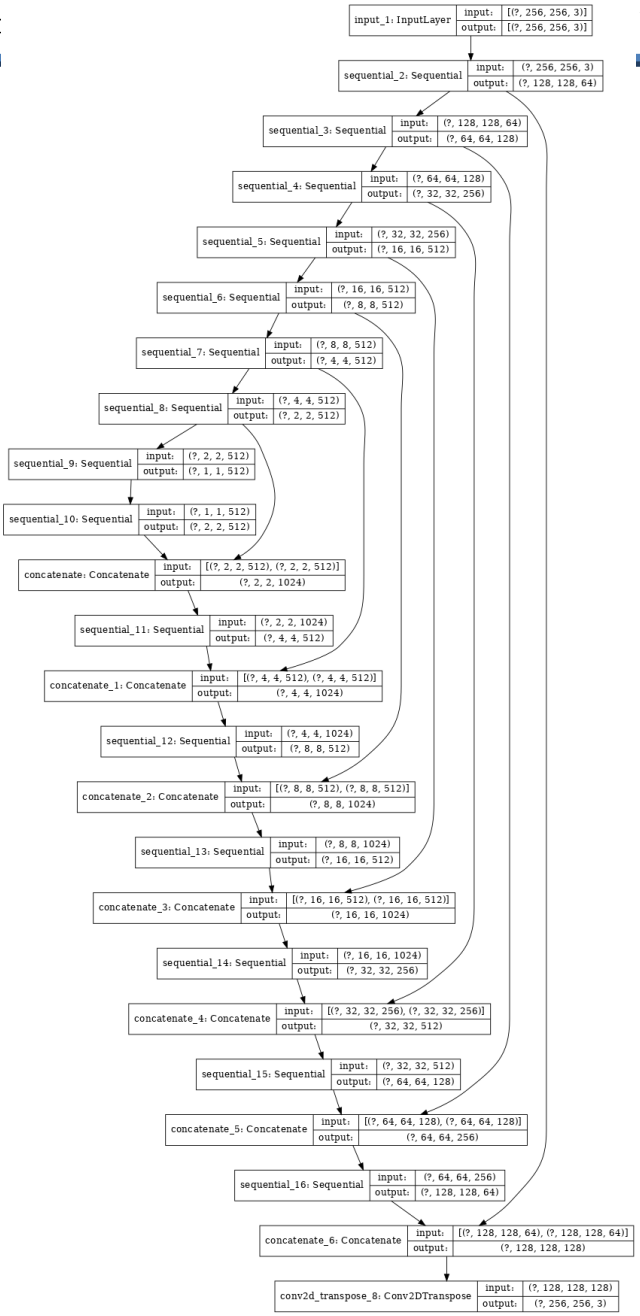
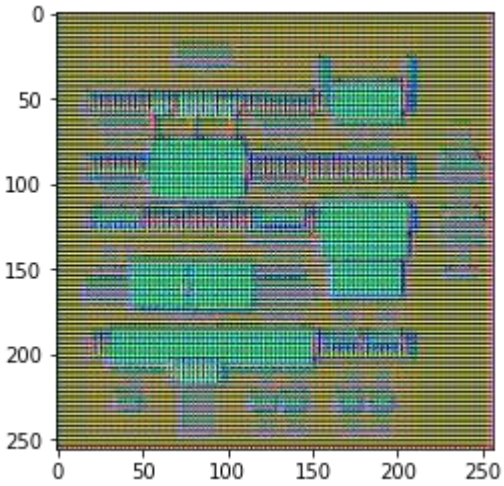


Ilustración 1 Workflow del generador.

```
gen_output = generator(inp[tf.newaxis,...], training=False)
plt.imshow(gen_output[0,...])
```

```
Recorte los datos de entrada al rango válido para imshow con datos
RGB ([0..1] para flotantes o [0..255] para enteros).

<matplotlib.image.AxesImage en 0x7f0f90280ac8>
```



Pérdida del generador

- Es una pérdida de entropía cruzada sigmoidea de las imágenes generadas y una **serie de** imágenes.
- El documento también incluye la pérdida de L1, que es MAE (error absoluto medio) entre la imagen generada y la imagen objetivo.
- Esto permite que la imagen generada se vuelva estructuralmente similar a la imagen de destino.
- La fórmula para calcular la pérdida total del generador = gan_loss + LAMBDA * l1_loss, donde LAMBDA = 100. Este valor fue decidido por los autores del artículo.

El procedimiento de entrenamiento para el generador se muestra a continuación:

```
LAMBDA =100
```

```
def generator_loss(disc_generated_output, gen_output, target):
  gan_loss = loss_object(tf.ones_like(disc_generated_output),
disc_generated_output)

  # mean absolute error
  l1_loss = tf.reduce_mean(tf.abs(target - gen_output))

  total_gen_loss = gan_loss +(LAMBDA * l1_loss)
  return total_gen_loss, gan_loss, l1_loss
```

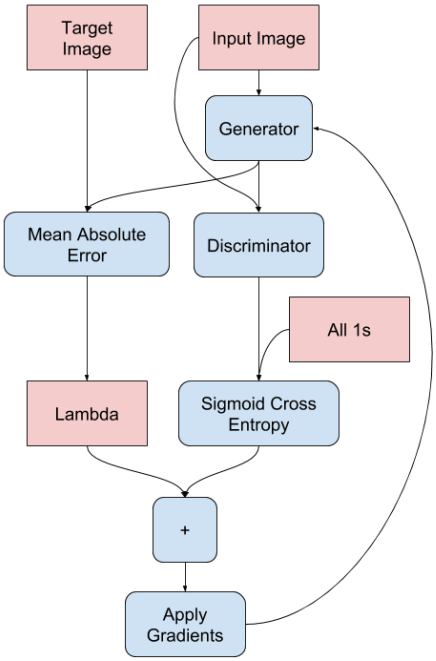


Ilustración 2 Grafo Computacional del generador.

Construyendo el discriminador

- El discriminador es un PatchGAN.
- Cada bloque en el discriminador es (Conv -> BatchNorm -> Leaky ReLU)
- La forma de la salida después de la última capa es (batch_size, 30, 30, 1)
- Cada parche de 30x30 de la salida clasifica una porción de 70x70 de la imagen de entrada (dicha arquitectura se llama PatchGAN).
- El discriminador recibe 2 entradas.
    - Imagen de entrada y la imagen de destino, que debe clasificar como real.
    - Imagen de entrada y la imagen generada (salida del generador), que debe clasificar como falsa.
    - Se concatena estas 2 entradas juntas en el código (tf.concat([inp, tar], axis=-1)).

```
defDiscriminator():
  initializer = tf.random_normal_initializer(0.,0.02)

  inp = tf.keras.layers.Input(shape=[256,256,3],
name='input_image')
  tar = tf.keras.layers.Input(shape=[256,256,3],
name='target_image')

  x = tf.keras.layers.concatenate([inp, tar])# (bs, 256, 256,
channels*2)

  down1 = downsample(64,4,False)(x)# (bs, 128, 128, 64)
  down2 = downsample(128,4)(down1)# (bs, 64, 64, 128)
  down3 = downsample(256,4)(down2)# (bs, 32, 32, 256)

  zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)# (bs, 34, 34,
256)
  conv = tf.keras.layers.Conv2D(512,4, strides=1,
                                kernel_initializer=initializer,
                                use_bias=False)(zero_pad1)# (bs,
31, 31, 512)

  batchnorm1 = tf.keras.layers.BatchNormalization()(conv)

  leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)

  zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)# (bs, 33,
33, 512)

  last= tf.keras.layers.Conv2D(1,4, strides=1,

kernel_initializer=initializer)(zero_pad2)# (bs, 30, 30, 1)

  return tf.keras.Model(inputs=[inp, tar], outputs=last)
```

```
discriminator =Discriminator()
tf.keras.utils.plot_model(discriminator, show_shapes=True, dpi=64)
```

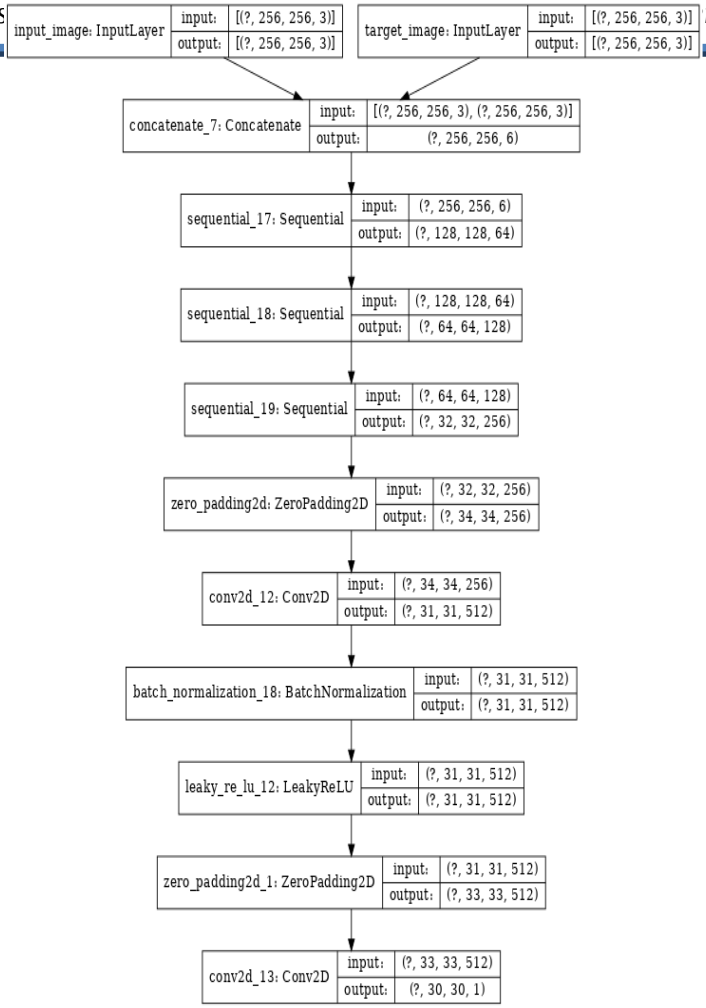
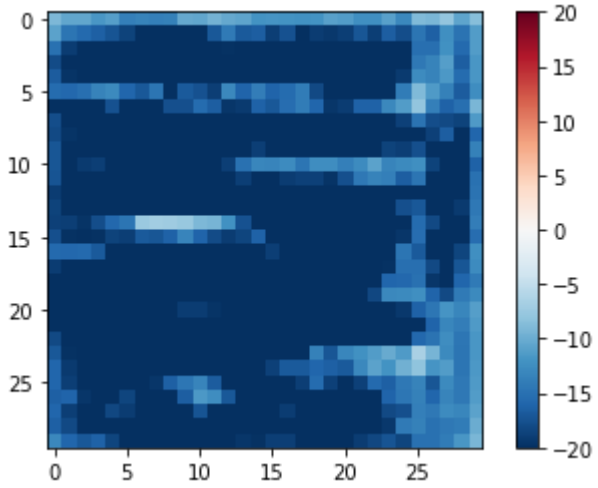


Ilustración 3 Workflow del discriminador.

```
disc_out = discriminator([inp[tf.newaxis,...], gen_output],
training=False)
plt.imshow(disc_out[0,...,-1], vmin=-20, vmax=20, cmap='RdBu_r')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar en 0x7f0f900b1630>
```



Pérdida del discriminador

- La función de pérdida discriminadora toma 2 entradas; **imágenes reales, imágenes generadas**
- real_loss es una pérdida de entropía cruzada sigmoidea de las **imágenes reales** y una **serie de imágenes (ya que estas son las imágenes reales)**
- generate_loss es una pérdida de entropía cruzada sigmoidea de las **imágenes generadas** y una **matriz de ceros (ya que estas son las imágenes falsas)**
- Entonces el total_loss es la suma de real_loss y el generate_loss

El procedimiento de entrenamiento para el discriminador se muestra a continuación.

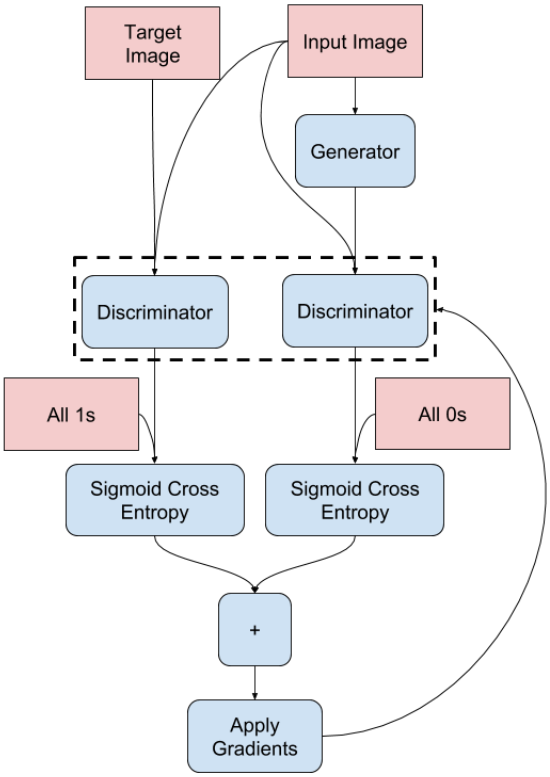


Ilustración 4 Grafo computacional del discriminador.

Definir los optimizadores y el punto de control de ahorro

```
generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4,
beta_1=0.5)
```

```
checkpoint_dir ='./training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir,"ckpt")
checkpoint =
tf.train.Checkpoint(generator_optimizer=generator_optimizer,

 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
```

Generando imágenes

Esta es una función para trazar algunas imágenes durante el entrenamiento.

- Las imágenes se pasan del conjunto de datos de prueba al generador.
- El generador traducirá la imagen de entrada a la salida.
- El último paso es trazar las predicciones y **¡listo!**

El training=True es intencional aquí, ya que queremos que las estadísticas por lotes mientras se ejecuta el modelo en el conjunto de datos de prueba. Si usamos training = False, obtendremos las estadísticas acumuladas aprendidas del conjunto de datos de entrenamiento (que no queremos).

```
def generate_images(model, test_input, tar):
  prediction = model(test_input, training=True)
  plt.figure(figsize=(15,15))

  display_list =[test_input[0], tar[0], prediction[0]]
  title =['Input Image','Ground Truth','Predicted Image']

  for i in range(3):
    plt.subplot(1,3, i+1)
    plt.title(title[i])
    # getting the pixel values between [0, 1] to plot it.
    plt.imshow(display_list[i]*0.5+0.5)
    plt.axis('off')
  plt.show()
```

```
for example_input, example_target in test_dataset.take(1):
  generate_images(generator, example_input, example_target)
```

Entrenamiento

- Para cada ejemplo, la entrada genera una salida.

- El discriminador recibe input_image y la imagen generada como la primera entrada. La segunda entrada es input_image y target_image.
- A continuación, calculamos el generador y la pérdida discriminadora.
- Luego, calculamos los gradientes de pérdida con respecto a las variables (entradas) del generador y del discriminador y las aplicamos al optimizador.
- Luego registre las pérdidas en TensorBoard.

```
EPOCHS =150
```

```
import datetime
log_dir="logs/"

summary_writer = tf.summary.create_file_writer(
  log_dir +"fit/"+ datetime.datetime.now().strftime("%Y%m%d-
%H%M%S"))
```

```
@tf.function
def train_step(input_image, target, epoch):
  with tf.GradientTape()as gen_tape, tf.GradientTape()as disc_tape:
    gen_output = generator(input_image, training=True)

    disc_real_output = discriminator([input_image, target],
training=True)
    disc_generated_output = discriminator([input_image,
gen_output], training=True)

    gen_total_loss, gen_gan_loss, gen_l1_loss =
generator_loss(disc_generated_output, gen_output, target)
    disc_loss = discriminator_loss(disc_real_output,
disc_generated_output)

  generator_gradients = gen_tape.gradient(gen_total_loss,

generator.trainable_variables)
  discriminator_gradients = disc_tape.gradient(disc_loss,

 discriminator.trainable_variables)

  generator_optimizer.apply_gradients(zip(generator_gradients,

generator.trainable_variables))
```

```
discriminator_optimizer.apply_gradients(zip(discriminator_gradients
,
discriminator.trainable_variables))

  with summary_writer.as_default():
    tf.summary.scalar('gen_total_loss', gen_total_loss, step=epoch)
    tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=epoch)
    tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=epoch)
    tf.summary.scalar('disc_loss', disc_loss, step=epoch)
```

El circuito de entrenamiento real:

- Itera sobre el número de épocas.
- En cada época, borra la pantalla y se ejecuta `generate_images`para mostrar su progreso.
- En cada época, itera sobre el conjunto de datos de entrenamiento, imprimiendo un '.' para cada ejemplo
- Se ahorra un punto de control cada 20 épocas.

```
def fit(train_ds, epochs, test_ds):
  for epoch in range(epochs):
    start = time.time()

    display.clear_output(wait=True)

    for example_input, example_target in test_ds.take(1):
      generate_images(generator, example_input, example_target)
```

```
    print("Epoch: ", epoch)

    # Train
    for n,(input_image, target)in train_ds.enumerate():
      print('.',end='')
      if(n+1)%100==0:
        print()
      train_step(input_image, target, epoch)
    print()

    # saving (checkpoint) the model every 20 epochs
    if(epoch +1)%20==0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print('Time taken for epoch {} is {} sec\n'.format(epoch
+1,time.time()-start))checkpoint.save(file_prefix =
checkpoint_prefix)
```

Este ciclo de entrenamiento guarda registros que puede ver fácilmente en TensorBoard para monitorear el progreso del entrenamiento. Trabajando localmente lanzarías un proceso separado de tensorboard. En un cuaderno, si desea monitorear con TensorBoard, es más fácil iniciar el visor antes de comenzar el entrenamiento.

Para iniciar el visor, pegue lo siguiente en una celda de código:

```
%load_ext tensorboard
%tensorboard --logdir {log_dir}
```

Ahora ejecuta el ciclo de entrenamiento:

```
fit(train_dataset, EPOCHS, test_dataset)
```

```
Época: 146
.....................................................
.....................................................
.....................................................
.....................................................
.....................................................
.....................................................
.....................................
```

TensorBoard.dev es una experiencia administrada para alojar, rastrear y compartir experimentos de ML con todos.

También se puede incluir en línea usando un `<iframe>` en esta puede visualizer los resultados de su entrenamiento y verificar la function de perdida del discriminador y generador.

Restaurar el último punto de control y prueba

```
$ls {checkpoint_dir}
```

```
checkpoint           ckpt-5.data-00000-of-00002
ckpt-1.data-00000-of-00002  ckpt-5.data-00001-of-00002
ckpt-1.data-00001-of-00002  ckpt-5.index
ckpt-1.index            ckpt-6.data-00000-of-00002
ckpt-2.data-00000-of-00002  ckpt-6.data-00001-of-00002
ckpt-2.data-00001-of-00002  ckpt-6.index
ckpt-2.index            ckpt-7.data-00000-of-00002
ckpt-3.data-00000-of-00002  ckpt-7.data-00001-of-00002
ckpt-3.data-00001-of-00002  ckpt-7.index
ckpt-3.index            ckpt-8.data-00000-of-00002
ckpt-4.data-00000-of-00002  ckpt-8.data-00001-of-00002
ckpt-4.data-00001-of-00002  ckpt-8.index
ckpt-4.index
```

```
# restoring the latest checkpoint in checkpoint_dir
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f0a575f0c88>
```

En esta punto la arquitectura del modelo ya esta creada y entrenada, solo queda probar el modelo y generar imagines con el dataset de prueba, esta sección corresponde al siguiente capítulo. En el capitulo actual se explicó el tipo de investigación, diseño así como la arquitectura y el código del modelo de red neuronal.

A N A L I S I S Y R E S U L T A D O S

Para explorar la generalidad de las GAN condicionales, se han probado en una variedad de tareas y conjuntos de datos, incluyendo ambos tareas de gráficos, como generación de fotos, y tareas de visión, como segmentación semántica, requisitos de datos y velocidad se puede ver en común que resultados decentes a menudo se puede obtener incluso en pequeños conjuntos de datos. Nuestra fachada el conjunto de entrenamiento consta de solo 400 imágenes y el conjunto de entrenamiento de día a noche consta de solo
91 cámaras web únicas.

Ilustración 5 Comparación con otros resultados de diferentes datasets.

Métricas de evaluación

Evaluar la calidad de las imágenes sintetizadas es un proceso abierto y difícil problema. Métricas tradicionales como perpixel

error cuadrático medio no evalúa estadísticas conjuntas de el resultado, y por lo tanto no mida la estructura misma que las pérdidas estructuradas apuntan a capturar. Para evaluar de manera más integral la calidad visual de nuestros resultados. Se empleó dos tácticas. Primero, corremos "real versus falso".

Estudios perceptuales sobre Amazon Mechanical Turk (AMT). Para problemas gráficos como la coloración y la generación de fotos, plausibilidad para un observador humano es a menudo lo último objetivo. Por lo tanto, probamos nuestra generación de mapas, foto aérea generación y coloración de imágenes usando este enfoque.

En segundo lugar, medimos si nuestro sintetizado o no los paisajes urbanos son lo suficientemente realistas como para que el reconocimiento inmediato sistema puede reconocer los objetos en ellos. Esta métrica es similar a la "puntuación inicial" de la detección de objetos evaluación y la "interpretabilidad semántica"medidas en. Estudios perceptuales de AMT Para el experimento segui el protocolo de Turkers con una serie de juicios que enfrentaron una imagen "real" contra un Imagen "falsa" generada por nuestro algoritmo. En cada prueba, cada imagen apareció durante 1 segundo, después de lo cual la imagen desapareció y a los turcos se les dio tiempo ilimitado para responder en cuanto a cuál era falso. Las primeras 10 imágenes de cada la sesión fue práctica y los Turkers recibieron retroalimentación. No Se proporcionaron comentarios sobre los 40 ensayos del experimento principal. Cada sesión probó solo un algoritmo a la vez, y A los turkers no se les permitió completar más de una sesión. 50 Turkers evaluaron cada algoritmo.

Análisis de la arquitectura del generador.

Una arquitectura U-Net permite que la información de bajo nivel tenga acceso directo a través de la red. Esto conduce a mejores resultados El codificador-decodificador es creado simplemente cortando las conexiones de omisión en U-Net. Las ventajas de la U-Net parece ser específica de las GAN condicionales. Se han realizado pruebas con otros tipos de arquitectura y los autores del paper han llegado a la conclusión de que la U-Net nuevamente logra los mejores resultados

Los resultados en este documento sugieren que las redes adversas condicionales son un enfoque prometedor para muchas tareas de traducción de imagen a imagen, especialmente aquellas que involucran salidas gráficas estructuradas Estas redes aprenden una pérdida

adaptado a la tarea y los datos disponibles, lo que los hace aplicables en una amplia variedad de entornos.

Segmentación semántica

Las GAN condicionales parecen ser efectivas en problemas donde la salida es altamente detallada o fotográfica, como es Común en el procesamiento de imágenes y tareas gráficas. Qué sobre problemas de visión, como segmentación semántica, donde la salida es menos compleja que la entrada. Aunque las CGAN logran cierto éxito, están lejos de ser el mejor método disponible para resolver esto
problema: simplemente usando la regresión L1 se obtienen mejores puntajes que
usando un cGAN. Argumentamos que para la visión problemas, el objetivo (es decir, predecir resultados cercanos a la verdad básica) puede ser menos ambigua que las tareas gráficas, y las pérdidas de reconstrucción como L1 son en su mayoría suficientes.

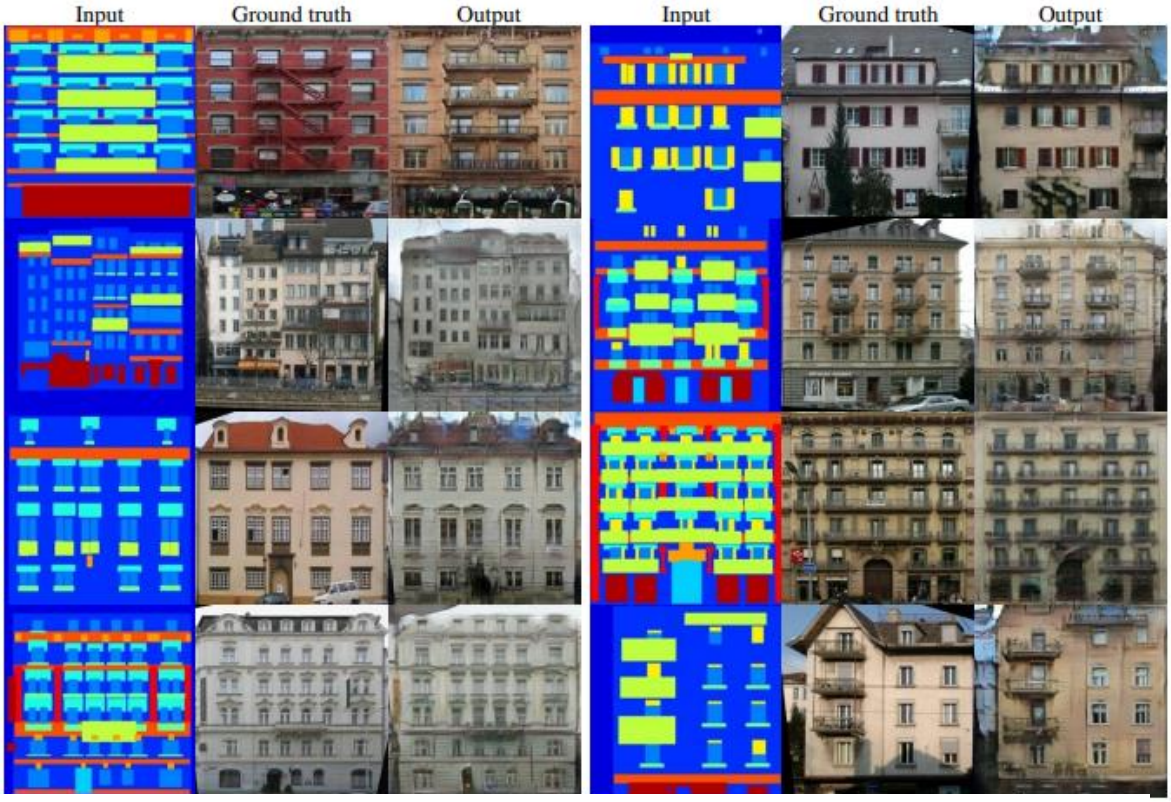


Ilustración 6 Resultados obtenidos después de entrenar a la red y ejecutar con el dataset de prueba.

Conclusiones

Los resultados en este documento sugieren que las redes adversas condicionales son un enfoque prometedor para muchas tareas de traducción de imagen a imagen, especialmente aquellas que involucran salidas gráficas estructuradas Estas redes aprenden una pérdida adaptado a la tarea y los datos disponibles, lo que los hace aplicables en una amplia variedad de entornos. Investigamos las redes adversas condicionales como una solución de propósito general a los problemas de traducción de imagen a imagen. Estas redes no solo aprenden el mapeo de la imagen de entrada a la imagen de salida, sino que también aprenden una función de pérdida para entrenar este mapeo. Esto hace posible aplicar el mismo enfoque genérico a problemas que tradicionalmente requerirían formulaciones de pérdidas muy diferentes. Demostramos que este enfoque es efectivo para sintetizar fotos a partir de mapas de etiquetas, reconstruir objetos a partir de mapas de bordes e colorear imágenes, entre otras tareas. De hecho, desde el lanzamiento del software pix2pix asociado con este documento, un gran número de usuarios de Internet (muchos de ellos artistas) han publicado sus propios experimentos con nuestro sistema, demostrando aún más su amplia aplicabilidad y facilidad de adopción sin la necesidad de ajustes de parámetros.

Referencias Bibliográficas

Bertrand gondouin. https://twitter.com/ bgondouin/status/818571935529377792. Accessed, 2017-04-21.

Brannon dorsey. https://twitter.com/ brannondorsey/status/806283494041223168. Accessed, 2017-04-21.

Christopher hesse. https://affinelayer.com/ pixsrv/. Accessed: 2017-04-21.

Gerda bosman, tom kenter, rolf jagerman, and daan gosman. https://dekennisvannu.nl/site/artikel/ Help-ons-kunstmatige-intelligentie-testen/ 9163. Accessed: 2017-08-31.

Jack qiao. http://colormind.io/blog/. Accessed: 2017-04-21.

Kaihu chen. http://www.terraai.org/ imageops/index.html. Accessed, 2017-04-21.

Mario klingemann. https://twitter.com/ quasimondo/status/826065030944870400. Accessed, 2017-04-21.

Memo akten. https://vimeo.com/260612034. Accessed, 2018-11-07

A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In CVPR, 2005.

L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In ICLR, 2015.

T. Chen, M.-M. Cheng, P. Tan, A. Shamir, and S.-M. Hu. Sketch2photo: internet image montage. ACM Transactions on Graphics (TOG), 28(5):124, 2009.

M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In CVPR, 2016.

E. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In NIPS, 2015.

C.Doersch,S.Singh,A.Gupta,J.Sivic,andA.Efros. What makesparislooklikeparis? ACMTransactionsonGraphics, 31(4), 2012.

A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In NIPS, 2016.

A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In SIGGRAPH, 2001.