Proyecto 1

OBJETIVOS

- Comprender los principios fundamentales de los autoencoders y su aplicación en deep learning generativo.
- Implementar un autoencoder básico y variacionales para una tarea específica, como reducción de dimensión, denoising o generación de imágenes.
- Analizar el rendimiento y las características de las representaciones aprendidas por los autoencoders.

MI PROYECTO

En este trabajo, se explorará la aplicación de un Variational Autoencoder (VAE) para la generación de imágenes de Pokémon. Los VAE son una clase de redes neuronales que permiten modelar distribuciones de datos complejas y generar nuevos ejemplos de manera probabilística. En este contexto, el objetivo es entrenar un VAE utilizando un conjunto de datos de imágenes de Pokémon, con la finalidad de que el modelo aprenda las características intrínsecas de estas criaturas y pueda generar imágenes nuevas que se asemejen a las originales.

DATOS

“Images of all Pokemon from generation 1 to generation 7, along with their types (primary and secondary) as a csv.” Fueron extraidos de Kaggle:

https://www.kaggle.com/datasets/vishalsubbiah/pokemon-images-and-types/data

images

809 files

PREPROCESAMIENTO

Las imágenes son en PNG, por lo que al meterlas al modelo esto afecta demasiado. Así que implemente un código para añadirles un fondo blanco y que mejore el rendimiento del modelo:

```
#def quitar_transparencia(imagen_con_transparencia, color_de_fondo=(255, 255, 255)):
 #   imagen_sin_transparencia = Image.new("RGB", imagen_con_transparencia.size, color_de_fondo)
  #  imagen_sin_transparencia.paste(imagen_con_transparencia, mask=imagen_con_transparencia.split()[3])
   # return imagen_sin_transparencia

# Directorio que contiene las imágenes con transparencia
#directorio_imagenes = "/content/drive/MyDrive/DL/PROYECTO 1/pokemon/pokemon"

# Directorio donde se guardarán las imágenes sin transparencia
#directorio_destino = "/content/drive/MyDrive/DL/PROYECTO 1/pokemon_jpg/pokemon_jpg"

# Crear el directorio de destino si no existe
#if not os.path.exists(directorio_destino):
 #   os.makedirs(directorio_destino)

# Iterar sobre todas las imágenes en el directorio
#for nombre_imagen in os.listdir(directorio_imagenes):
 #   ruta_imagen = os.path.join(directorio_imagenes, nombre_imagen)

    # Ignorar archivos que no sean imágenes
  #  if not nombre_imagen.endswith(('.png', '.jpg', '.jpeg')):
   #     continue

    # Abrir la imagen con transparencia
  #  imagen_con_transparencia = Image.open(ruta_imagen)

    # Quitar la transparencia
   # imagen_sin_transparencia = quitar_transparencia(imagen_con_transparencia)

    # Guardar la imagen sin transparencia en el directorio de destino
    #ruta_imagen_sin_transparencia = os.path.join(directorio_destino, nombre_imagen)
    #imagen_sin_transparencia.save(ruta_imagen_sin_transparencia)

    #print("Transparencia eliminada y la imagen sin transparencia ha sido guardada como:", ruta_imagen_sin_transparencia)
```

ESTRUCTURA Y EXPERIMENTOS

Después de varios experimentos probando varios tamaños de imagen, varias estructuras de modelo y diferentes learning rates los que decidí al final fue el siguiente modelo y parámetros. Cabe aclarar que es el que mejor funciona pero no funciona bien:

```python
# Definir constantes
batch_size = 32  # Tamaño del lote para el entrenamiento
latent_dim = 128  # Dimensión latente para el modelo
epochs = 50  # Número de épocas de entrenamiento
input_shape = X_train.shape[1:]  # Forma de entrada para el modelo
```

ENCODER:

- Se realiza normalización por lotes en las activaciones de la capa anterior.
- Fueron puras capas convolucionales.
- Se guarda la forma de los datos antes de aplanarlos. Esto será útil más tarde durante el proceso de decodificación.
- Se aplana la salida de la última capa convolucional para prepararla para la capa densa.

```
Model: "encoder"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 encoder_input (InputLayer)  [(None, 64, 64, 3)]       0         []

 conv2d (Conv2D)             (None, 32, 32, 32)        896       ['encoder_input[0][0]']

 batch_normalization (Batch  (None, 32, 32, 32)        128       ['conv2d[0][0]']
 Normalization)

 conv2d_1 (Conv2D)           (None, 16, 16, 64)        18496     ['batch_normalization[0][0]']

 batch_normalization_1 (Bat  (None, 16, 16, 64)        256       ['conv2d_1[0][0]']
 chNormalization)

 conv2d_2 (Conv2D)           (None, 8, 8, 128)         73856     ['batch_normalization_1[0][0]'
                                                                 ]

 batch_normalization_2 (Bat  (None, 8, 8, 128)         512       ['conv2d_2[0][0]']
 chNormalization)

 conv2d_3 (Conv2D)           (None, 4, 4, 256)         295168    ['batch_normalization_2[0][0]'
                                                                 ]

 batch_normalization_3 (Bat  (None, 4, 4, 256)         1024      ['conv2d_3[0][0]']
 chNormalization)

 flatten (Flatten)           (None, 4096)              0         ['batch_normalization_3[0][0]'
                                                                 ]

 dense (Dense)               (None, 512)               2097664   ['flatten[0][0]']

 z_mean (Dense)              (None, 128)               65664     ['dense[0][0]']

 z_log_var (Dense)           (None, 128)               65664     ['dense[0][0]']

 z (Lambda)                  (None, 128)               0         ['z_mean[0][0]',
                                                                  'z_log_var[0][0]']

==================================================================================================
Total params: 2619328 (9.99 MB)
Trainable params: 2618368 (9.99 MB)
Non-trainable params: 960 (3.75 KB)
__________________________________________________________________________________________________
```

DECODER:

- Se define una entrada para los vectores latentes que se muestrean durante el proceso de decodificación.
- Se aplica una capa densa.
- Se remodela el tensor de salida de la capa densa para que coincida con las dimensiones antes de aplanar.
- Se aplican una capa de convolución transpuesta seguidas de una normalización.
- Y al final capa de convolución transpuesta final con 3 filtros (uno para cada canal de color en una imagen RGB).

```
Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 z_sampling (InputLayer)     [(None, 128)]             0

 dense_1 (Dense)             (None, 4096)              528384

 reshape (Reshape)           (None, 4, 4, 256)         0

 conv2d_transpose (Conv2DTr  (None, 8, 8, 256)         590080
 anspose)

 batch_normalization_4 (Bat  (None, 8, 8, 256)         1024
 chNormalization)

 conv2d_transpose_1 (Conv2D  (None, 16, 16, 128)       295040
 Transpose)

 batch_normalization_5 (Bat  (None, 16, 16, 128)       512
 chNormalization)

 conv2d_transpose_2 (Conv2D  (None, 32, 32, 64)        73792
 Transpose)

 batch_normalization_6 (Bat  (None, 32, 32, 64)        256
 chNormalization)

 conv2d_transpose_3 (Conv2D  (None, 64, 64, 32)        18464
 Transpose)

 batch_normalization_7 (Bat  (None, 64, 64, 32)        128
 chNormalization)

 conv2d_transpose_4 (Conv2D  (None, 64, 64, 3)         867
 Transpose)

=================================================================
Total params: 1508547 (5.75 MB)
Trainable params: 1507587 (5.75 MB)
Non-trainable params: 960 (3.75 KB)
_________________________________________________________________
```

RESULTADOS

```
Epoch 32/50
21/21 [==============================] - 16s 767ms/step - loss: 349.9775 - val_loss: 420.3201
```

El mejor los que me dio fue en anterior, y la generación me dieron imágenes de este tipo:

CONCLUSIÓN

Mi modelo VAE tiene dificultades para generar imágenes nítidas. A pesar de mis intentos de mejorar la calidad agregando más capas al modelo, no he logrado obtener resultados significativamente mejores. Cuando intento aumentar el tamaño del shape a más de 32x32x3, el rendimiento del modelo empeora en lugar de mejorar. Creo que estos problemas podrían estar relacionados con el proceso de agregar fondo a las imágenes PNG. Además, he observado que el valor del loss del modelo rara vez es satisfactorio, lo que sugiere que el modelo no está aprendiendo de manera efectiva. Incluso al intentar reducir el learning rate, el problema persiste y el loss sigue siendo alto.