Reporte DAE y VAE

Melanie Rodriguez, Santiago Aguirre, Regina Flores

Objetivo del Proyecto

En este proyecto trabajamos con dos modelos: un **Denoising Autoencoder (DAE)** y un **Variational Autoencoder (VAE)**. Ambos los entrenamos con imágenes de latas de refrescos (Coca-Cola y Sprite), con el objetivo de que aprendieran a representar bien las imágenes.

El DAE tenía como propósito limpiar imágenes con ruido para que se parecieran lo más posible a las originales. Por otro lado, el VAE no solo reconstruye, sino que también aprende una representación más "creativa", lo cual nos permite generar imágenes nuevas que se ven parecidas a las reales.

Recolección y Procesamiento de Imágenes

Las imágenes las conseguimos usando una extensión de Chrome, de donde recolectamos fotos de latas de Sprite y Coca. Al principio, el dataset era algo desordenado: muchas imágenes estaban fuera de foco, no centradas o con fondos complicados. Por eso, hicimos una selección manual para quedarnos solo con imágenes que estuvieran limpias, centradas y bien enfocadas.

Después, las procesamos usando una función en Python que les aplicaba **padding** con color blanco para que todas tuvieran el mismo tamaño sin deformarse. También normalizamos los valores de los píxeles para que estuvieran entre 0 y 1, y dividimos los datos en entrenamiento y prueba con `train_test_split`.

El tamaño final del dataset fue:

- Imágenes de entrenamiento: (102, 128, 128, 3)
- Imágenes de prueba: (26, 128, 128, 3)

Este proceso nos ayudó a tener datos más limpios y consistentes para que los modelos pudieran aprender mejor.

Código del Denoising Autoencoder (DAE)

```
# Encoder
input_img = layers.Input(shape=(128, 128, 3))
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img
    )
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# Decoder
x = layers.Conv2DTranspose(64, (3, 3), activation='relu', padding='same')(
    encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2DTranspose(32, (3, 3), activation='relu', padding='same')(
    x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x
    )

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse')

# Agregar ruido
def add_gaussian_noise(images, mean=0, std=0.1):
    noise = np.random.normal(mean, std, images.shape)
    noisy_images = images + noise
    return np.clip(noisy_images, 0, 1)

x_train_noisy = add_gaussian_noise(x_train)
x_test_noisy = add_gaussian_noise(x_test)
```

Código del Denoising Autoencoder (DAE)

```
# Encoder
input_img = layers.Input(shape=(128, 128, 3))
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img
    )
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# Decoder
x = layers.Conv2DTranspose(64, (3, 3), activation='relu', padding='same')(
    encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2DTranspose(32, (3, 3), activation='relu', padding='same')(
    x)
x = layers.UpSampling2D((2, 2))(x)
```

```
decoded = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x
    )

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse')

# Agregar ruido
def add_gaussian_noise(images, mean=0, std=0.1):
    noise = np.random.normal(mean, std, images.shape)
    noisy_images = images + noise
    return np.clip(noisy_images, 0, 1)

x_train_noisy = add_gaussian_noise(x_train)
x_test_noisy = add_gaussian_noise(x_test)
```

Arquitecturas de los Modelos

Denoising Autoencoder (DAE)

El DAE que implementamos tiene una arquitectura simple pero efectiva. En el **encoder**, usamos dos capas convolucionales con activación ReLU y reducción de tamaño con *Max-Pooling*. Estas capas permiten comprimir la imagen y extraer sus características más importantes. Luego, en el **decoder**, hacemos el proceso inverso: usamos capas `Conv2DTranspose` y `UpSampling` para recuperar el tamaño original y reconstruir la imagen.

La salida final pasa por una activación `sigmoid` para que los valores estén normalizados entre 0 y 1, igual que las imágenes originales. Antes de entrenar el modelo, le agregamos **ruido gaussiano** a las imágenes, para que el autoencoder aprendiera a "limpiar" la imagen y recuperar su versión original lo mejor posible. Esta idea se basa en que, al obligar al modelo a ignorar el ruido, aprende representaciones más robustas.

Variational Autoencoder (VAE)

El VAE tiene una arquitectura un poco más compleja. En lugar de solo codificar una imagen a un vector fijo, el encoder del VAE genera dos vectores: una **media** y una **varianza logarítmica**, que definen una distribución gaussiana en el espacio latente. A partir de eso, se genera un vector aleatorio z, que luego se pasa al decoder.

El decoder tiene varias capas de `Conv2DTranspose` y `UpSampling2D` que reconstruyen la imagen paso a paso. También usamos `BatchNormalization` y `LeakyReLU` para que el entrenamiento fuera más estable y rápido.

Una parte importante del VAE es su **función de pérdida**, que combina dos cosas: el error de reconstrucción (qué tan parecida es la imagen generada a la original) y una

penalización llamada **KL Divergence**, que ayuda a que el espacio latente se parezca a una distribución normal. Esto permite generar imágenes nuevas de forma más controlada y realista.

Además, agregamos una **métrica personalizada** usando una red preentrenada (DenseNet121), que compara las características visuales profundas entre imágenes originales y generadas usando distancia euclidiana. Así no solo medimos diferencias de píxeles, sino también de contenido.

Resultados y Análisis

Denoising Autoencoder (DAE)

En el caso del DAE, los resultados fueron bastante buenos. En la Figura 1, se puede observar la comparación entre la imagen original, la imagen con ruido gaussiano y la imagen reconstruida. El modelo logró eliminar la mayor parte del ruido y conservar los detalles principales como los colores, el logotipo y la forma de la lata. Aunque las reconstrucciones no son perfectas, el resultado visual es claro y entendible.

Esto muestra que el DAE fue capaz de aprender representaciones útiles de las imágenes, incluso con un dataset pequeño y limitado. Además, el hecho de que las imágenes estuvieran bien centradas y limpias ayudó a que el modelo pudiera generalizar mejor y enfocarse en los elementos relevantes.

Figure 1: Reconstrucción de imágenes con DAE: original (arriba), ruidosa (medio), reconstruida (abajo).

Variational Autoencoder (VAE)

El VAE fue más difícil de entrenar. Como se ve en la Figura 2, las imágenes reconstruidas son mucho más borrosas y pierden casi todos los detalles visuales importantes. Aunque se puede intuir la forma general de una lata, no hay texto, logotipo ni bordes definidos.

Esto puede deberse a varios factores:

- **Calidad del dataset:** Al principio teníamos imágenes de baja calidad, no centradas y con fondos ruidosos. Esto afectó el aprendizaje del VAE, ya que este modelo es más sensible a la calidad y consistencia de los datos.
- **Naturaleza del VAE:** A diferencia del DAE, el VAE introduce aleatoriedad al momento de generar el vector latente. Esto puede hacer que la salida sea más borrosa si el modelo no ha aprendido una distribución clara.
- **Pérdida KL Divergence:** En muchos casos, la pérdida de KL puede dominar y hacer que el modelo sacrifique detalles para acercarse a una distribución normal. Si no está bien balanceada con la pérdida de reconstrucción, se obtienen salidas genéricas y poco definidas.
- **Tamaño del espacio latente:** Tal vez el vector latente (z_dim = 128) no fue suficiente para capturar todos los detalles. Se podría probar con dimensiones más grandes como 256 o 512.
- **Hiperparámetros:** También se podrían ajustar el learning rate, el peso del término KL o incluso probar otros tipos de activaciones o regularización.

Aun así, tras varias pruebas y mejor curación del dataset, se logró un resultado que, aunque sigue siendo difuso, muestra una mejora clara frente a las versiones anteriores.

Figure 2: Reconstrucción de imágenes con VAE: original (arriba), reconstruida (abajo).

Comparación general

En resumen, el DAE fue más efectivo para este tipo de tarea visual porque se enfoca directamente en reconstruir imágenes limpias desde imágenes ruidosas. El VAE, aunque más interesante desde el punto de vista generativo, necesita datos más limpios, una arquitectura más robusta y mayor fine-tuning para obtener resultados comparables en calidad visual.

Conclusión

Este proyecto fue una buena oportunidad para experimentar con modelos generativos y entender cómo funcionan en la práctica. Implementar tanto el Denoising Autoencoder (DAE) como el Variational Autoencoder (VAE) nos ayudó a comparar dos enfoques distintos para reconstrucción de imágenes y ver sus ventajas y limitaciones.

En general, el DAE nos dio mejores resultados visuales. Fue más estable y reconstruyó las imágenes con buena calidad, incluso cuando se les añadía ruido. El VAE, por otro lado, fue más desafiante. Nos dimos cuenta de que este tipo de modelo es mucho más sensible a la calidad de los datos y a los hiperparámetros. Las imágenes que generaba al principio parecían manchas, y solo después de filtrar mejor nuestro dataset y hacer varios ajustes logramos un resultado un poco más claro.

Algo que podríamos mejorar sería usar más imágenes y con mejor consistencia visual (fondos neutros, centradas, buena resolución). También podríamos probar con un espacio latente más grande, ajustar mejor la función de pérdida del VAE, o incluso usar otras variantes como CVAE o GANs.

Estos modelos tienen aplicaciones interesantes. El DAE, por ejemplo, se puede usar para limpiar imágenes dañadas o mejorar calidad visual en fotos antiguas. El VAE tiene potencial para generar nuevas imágenes, hacer interpolaciones o para tareas creativas como generar diseños de productos o prototipos visuales.

En resumen, aprendimos mucho del proceso, tanto técnico como práctico. Tuvimos que lidiar con problemas reales de calidad de datos, entrenamiento y visualización, y eso nos dejó con más herramientas para futuros proyectos de visión por computadora.