Reporte del Proyecto: Neural Style Transfer para Simular Siniestros

1. Introducción

En el mundo de los seguros, contar con imágenes representativas de **distintos tipos de siniestros** (choques, incendios, daños por granizo) es clave para entrenar modelos de evaluación de riesgos. Sin embargo, muchas veces **no existen suficientes imágenes reales** de todos los escenarios posibles, especialmente los más extremos.

Este proyecto propone una solución: **usar Neural Style Transfer (NST) para generar imágenes sintéticas** de siniestros.

- **Objetivo:** Crear imágenes alteradas con distintos niveles de daño (leve, moderado, severo) a partir de fotos originales de autos y propiedades.
- **Beneficio:** Ampliar datasets de aseguradoras para mejorar la precisión en modelos de evaluación y clasificación de riesgos.

Además, se aplicó **procesamiento de imágenes y transferencia de estilo** para visualizar distintos tipos de daño en **vehículos y edificios**.
Se utilizaron máscaras segmentadas que permiten **guiar la aplicación de efectos visuales**, asegurando una integración "realista" en la imagen original.

---

2. Metodología

¿Cómo funciona NST aplicado a siniestros?

- **Separación de contenido y estilo:** Una red neuronal convolucional (VGG19) analiza una imagen y extrae su **estructura (contenido)** y **textura (estilo)**.
- **Optimización de pérdida:** Al combinar contenido y estilo, se genera una imagen nueva que **mantiene la forma original pero adopta la textura del daño**.
- **Aplicación guiada por máscara:** Se define qué zonas deben recibir el efecto de daño, asegurando que las alteraciones sean **coherentes con la imagen base**.

Proceso detallado en el código:

- **Generación de máscaras** para identificar zonas afectadas.
- **Aplicación de transferencia de estilo** guiada por máscaras.
- **Acumulación de daños progresivos** para evaluar la severidad.

En resumen, el uso de máscaras permite personalizar la transferencia de estilo, simulando múltiples tipos de daño de forma controlada y realista.

2.1 Librerías Utilizadas

Se emplearon las siguientes librerías para manipulación de imágenes y aprendizaje profundo:

```
import os
import cv2
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from PIL import Image
```

2.2 Tipos de Daño Considerados

Para cada imagen, se asocian colores específicos con **diferentes efectos de daño**, como rayones, grietas y colisiones.

```
damage_classes = {
    (255, 102, 0): 'style_abolladura.jpg',
    (255, 0, 0): 'style_choque.jpg',
    (0, 255, 0): 'style_grieta.jpg',
    (255, 255, 0): 'style_incendio.jpg',
    (0, 153, 255): 'style_parabrisas.jpg',
    (255, 0, 255): 'style_rayon.jpg',
    (0, 255, 255): 'style_ventana_auto.jpg',
    (200, 200, 200): 'style_ventana_edificio.jpg'
}
```

2.3 Carga y Procesamiento de Máscaras

Las máscaras de daño se encuentran en **dos carpetas principales**, una para vehículos y otra para edificios.

```
mask_folders = ["/content/drive/MyDrive/Ejemplo_auto",
                "/content/drive/MyDrive/Ejemplo_edificio"]

for mask_folder in mask_folders:
    print(f"\n Procesando carpeta: {mask_folder}")

    mask_paths = [os.path.join(mask_folder, file) for file in
os.listdir(mask_folder) if file.endswith(('.png'))]

    for mask_path in mask_paths:
        print(f" Procesando máscara: {mask_path}")

        # Cargar la máscara multiclase
        mask_original = cv2.imread(mask_path)
        mask_original = cv2.cvtColor(mask_original, cv2.COLOR_BGR2RGB)

        # Detectar colores únicos en la máscara
        unique_colors = np.unique(mask_original.reshape(-1,
mask_original.shape[2]), axis=0)

        # Filtrar colores que coincidan con `damage_classes`
```

```
        valid_colors = [tuple(color) for color in unique_colors if tuple(color) in
damage_classes]

        # Obtener nombre base sin la extensión
        base_name = os.path.splitext(os.path.basename(mask_path))[0]  # Esto
elimina la extensión original

        for color in valid_colors:
            damage_mask = (mask_original == color).all(axis=-1).astype(np.uint8) *
255  # Crear máscara binaria

            # Suavizar bordes
            damage_mask = cv2.GaussianBlur(damage_mask, (15, 15), 0)

            # Guarda la máscara en la misma carpeta que la imagen original
            # Limpiar el nombre del daño (quita '.jpg')
            damage_name = damage_classes[color]
            damage_name_clean = damage_name.replace(".jpg", "")
```

---

3. Generación de Máscaras Binarias

Cada máscara multiclase se procesa para **extraer tipos de daño individuales** presentes en la imagen:

```
mask_original = cv2.imread(mask_path)
mask_original = cv2.cvtColor(mask_original, cv2.COLOR_BGR2RGB)

unique_colors = np.unique(mask_original.reshape(-1, mask_original.shape[2]),
axis=0)
valid_colors = [tuple(color) for color in unique_colors if tuple(color) in
damage_classes]
```

Luego se generan máscaras binarias por tipo de daño:

```
damage_mask = (mask_original == color).all(axis=-1).astype(np.uint8) * 255
damage_mask = cv2.GaussianBlur(damage_mask, (15, 15), 0)
```

Estas se guardan en la misma carpeta del contenido original:

```
mask_path_output = os.path.join(mask_folder, f"
{base_name}_{damage_name_clean}.png")
cv2.imwrite(mask_path_output, damage_mask)
```

---

4. Transferencia de Estilo Guiada por Máscara

4.1 Modelo VGG19

Se emplea VGG19 para extraer las características de contenido y estilo de la imagen original y del patrón de daño:

```
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
vgg.trainable = False
```

4.2 Proceso de Optimización

La transferencia de estilo se realiza mediante un proceso iterativo:

```
for i in range(num_iterations):
    loss = train_step(init_image, model, loss_weights, style_features,
content_features, opt)

    if loss < best_loss:
        best_loss = loss
        best_img = init_image.numpy()

    if i % 50 == 0:
        print(f"Iteración {i}, pérdida: {loss.numpy():.2f}")
```

4.3 Fusión con Imagen Original

La imagen generada se mezcla con la original utilizando la máscara binaria como guía:

```
mask_rgb = np.concatenate([mask]*3, axis=-1)
final_blend = (content_img_np * (1 - mask_rgb) + stylized_img_np *
mask_rgb).astype('uint8')
```

---

5. Simulación de Daños en Vehículos y Edificios

Ejemplo de daño acumulado sobre un vehículo:

```
mask_global = None

# Aplicación acumulada de daños (simula escenarios con múltiples eventos)

mask_global = simular_siniestros(content_path, style_path, mask_path,
output_base_path, name_image, mask_acumulada=mask_global)
mask_global = simular_siniestros(content_path2, style_path2, mask_path2,
output_base_path, name_image2, mask_acumulada=mask_global)
```

```
mask_global = simular_siniestros(content_path3, style_path3, mask_path3,
output_base_path, name_image3, mask_acumulada=mask_global)
```

Evaluación de severidad del daño:

```
area_danada = np.sum(mask_global)
area_total = mask_global.shape[1] * mask_global.shape[2]
porcentaje_danio = (area_danada / area_total) * 100

if porcentaje_danio < 10:
    print("Nivel de daño: LEVE")
elif porcentaje_danio < 30:
    print("Nivel de daño: MODERADO")
else:
    print("Nivel de daño: SEVERO")
```

El mismo procedimiento se aplica para imágenes de edificios.

---

6. Conclusión

Este proyecto demuestra el potencial de Neural Style Transfer (NST) en seguros, generando imágenes sintéticas de siniestros para entrenar modelos de evaluación.

- Amplía datasets sin depender de imágenes reales.
- Cuantifica daños de manera objetiva y reproducible.
- Optimiza la toma de decisiones en aseguradoras.