Data Augmentation

Objetivo

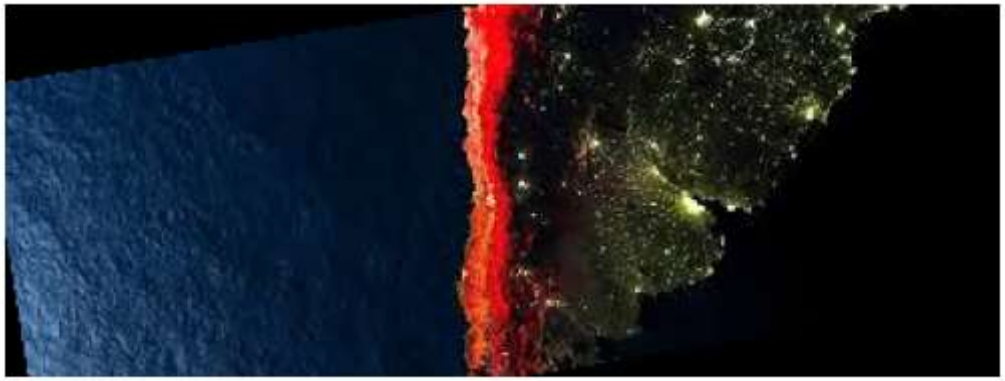
El objetivo de la **data augmentation** es aumentar la cantidad y diversidad del conjunto de datos de entrenamiento aplicando transformaciones aleatorias a los datos existentes. Esto ayuda a mejorar la capacidad de generalización del modelo y reduce el riesgo de sobreajuste.

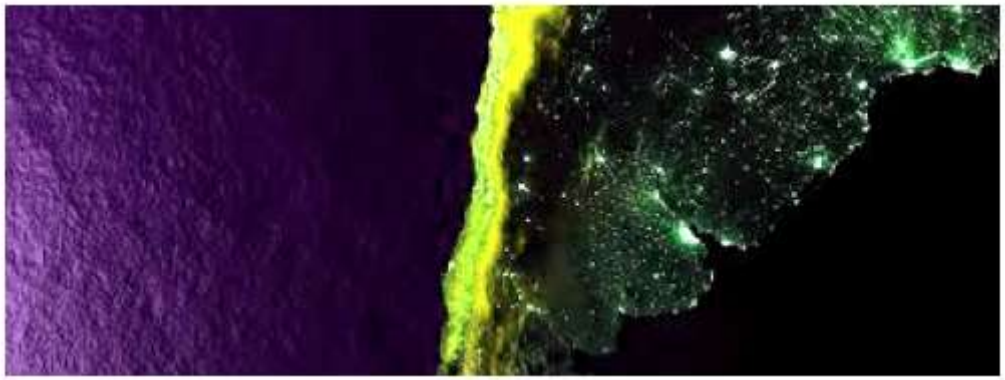
Técnicas Comunes de Data Augmentation

1. **Rotación**: Rotar las imágenes en varios ángulos.
2. **Traslación**: Desplazar las imágenes en varias direcciones.
3. **Escalado**: Ampliar o reducir el tamaño de las imágenes.
4. **Flip**: Voltear las imágenes horizontal o verticalmente.
5. **Ajuste de Brillo y Contraste**: Modificar los niveles de brillo y contraste.
6. **Recorte**: Recortar partes de las imágenes.
7. **Ruido Aleatorio**: Añadir ruido a las imágenes.

In [4]:

```python
import matplotlib.pyplot as plt
from PIL import Image
import torchvision.transforms as transforms
import numpy as np

# Cargar una imagen de ejemplo (puedes usar cualquier imagen)
#img = Image.open('images/images.jpeg')
img = Image.open('images/satelite.jpeg')


# Definir las transformaciones para data augmentation
transformations = transforms.Compose([
    transforms.RandomHorizontalFlip(p=1.0),  # Flip horizontal
    transforms.RandomRotation(10),  # Rotar la imagen
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),  # Recortar y redimensionar
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),  # Ajuste de color
    transforms.ToTensor(),  # Convertir a tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalizar
])

# Aplicar las transformaciones y mostrar los resultados
def apply_transformation(transform, img):
    transformed_img = transform(img)
    transformed_img = transformed_img / 2 + 0.5  # Desnormalizar
    npimg = transformed_img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.axis('off')
    plt.show()

# Lista de transformaciones individuales para visualización paso a paso
individual_transforms = [
    transforms.RandomHorizontalFlip(p=1.0),
    transforms.RandomRotation(10),
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2)
]

# Aplicar y mostrar cada transformación individualmente
for t in individual_transforms:
    apply_transformation(transforms.Compose([t, transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5),

# Aplicar todas las transformaciones juntas
apply_transformation(transformations, img)
```



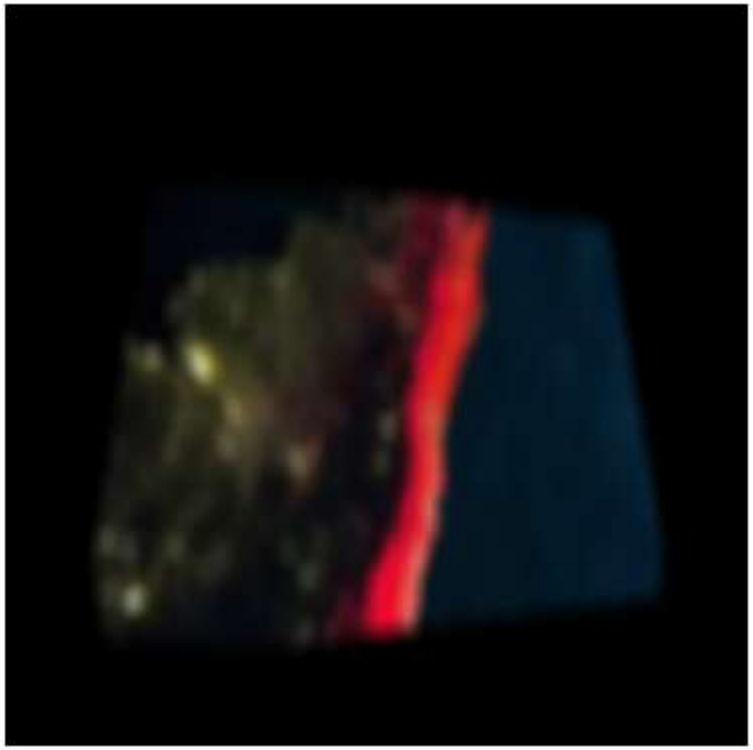


Transformaciones Adicionales

- RandomVerticalFlip: Voltea la imagen verticalmente con una probabilidad dada.
- RandomAffine: Aplica una transformación afín aleatoria.
- RandomPerspective: Aplica una transformación de perspectiva aleatoria.
- RandomGrayscale: Convierte la imagen a escala de grises con una probabilidad dada.
- GaussianBlur: Aplica un desenfoque gaussiano a la imagen.
- RandomErasing: Borra aleatoriamente un rectángulo de la imagen.

In [5]:
```python
# Definir las transformaciones para data augmentation
transformations = transforms.Compose([
    transforms.RandomHorizontalFlip(p=1.0),  # Flip horizontal
    transforms.RandomVerticalFlip(p=1.0),  # Flip vertical
    transforms.RandomRotation(10),  # Rotar la imagen
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),  # Recortar y redimensionar
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),  # Ajuste de color
    transforms.RandomAffine(degrees=15, translate=(0.1, 0.1)),  # Transformación afín
    transforms.RandomPerspective(distortion_scale=0.5, p=1.0),  # Transformación de perspectiva
    transforms.RandomGrayscale(p=0.2),  # Escala de grises
    transforms.GaussianBlur(kernel_size=(5, 9), sigma=(0.1, 5)),  # Desenfoque gaussiano
    transforms.ToTensor(),  # Convertir a tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),  # Normalizar
    transforms.RandomErasing(p=0.5, scale=(0.02, 0.33), ratio=(0.3, 3.3), value='random')  # Borrado aleatori
])
```

In [6]:
```
# Aplicar y mostrar cada transformación individualmente
for t in individual_transforms:
    apply_transformation(transforms.Compose([t, transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5),

# Aplicar todas las transformaciones juntas
apply_transformation(transformations, img)
```



In [ ]: