

```

# =====
# 1. Imports & Setup
# =====
import os
import numpy as np
import matplotlib.pyplot as plt
from glob import glob
from PIL import Image
import cv2
import tensorflow as tf
from tensorflow.keras import layers, models
import kagglehub

# Download Rain Dataset
kaggle_dataset_path = kagglehub.dataset_download("bshaurya/rain-dataset")
print("✅ Dataset downloaded at:", kaggle_dataset_path)

AUTOTUNE = tf.data.AUTOTUNE
print("Using TensorFlow version:", tf.__version__)

# =====
# 2. FFT-based Transform Functions
# =====
def to_frequency_domain(img_tensor, mode='real_imag'):
    img_tensor = tf.cast(img_tensor, tf.complex64)
    fft = tf.signal.fft2d(tf.transpose(img_tensor, perm=[2, 0, 1]))
    real = tf.math.real(fft)
    imag = tf.math.imag(fft)
    if mode == 'real_imag':
        return tf.stack([real, imag], axis=-1)
    elif mode == 'mag_phase':
        mag = tf.math.sqrt(real**2 + imag**2)
        phase = tf.math.atan2(imag, real)
        return tf.stack([mag, phase], axis=-1)
    else:
        raise ValueError("Mode must be 'real_imag' or 'mag_phase'")

def from_frequency_domain(freq_tensor):
    h, w, c, _ = freq_tensor.shape
    freq_tensor = tf.transpose(freq_tensor, perm=[2, 0, 1, 3])
    real = freq_tensor[..., 0]
    imag = freq_tensor[..., 1]
    complex_tensor = tf.complex(real, imag)
    ifft = tf.signal.ifft2d(complex_tensor)
    ifft = tf.math.real(ifft)
    ifft = tf.transpose(ifft, perm=[1, 2, 0])
    return tf.clip_by_value(ifft, 0.0, 1.0)

# =====
# 3. Dataset & DataLoader
# =====
def load_image_pair(rainy_path, clean_path, img_size=(256, 256), freq_mode='real_imag'):
    rainy = Image.open(rainy_path).convert('RGB').resize(img_size)
    clean = Image.open(clean_path).convert('RGB').resize(img_size)
    rainy = np.array(rainy) / 255.0
    clean = np.array(clean) / 255.0
    rainy_tf = tf.convert_to_tensor(rainy, dtype=tf.float32)
    clean_tf = tf.convert_to_tensor(clean, dtype=tf.float32)
    x_freq = to_frequency_domain(rainy_tf, mode=freq_mode)
    x_freq = tf.transpose(x_freq, perm=[1, 2, 0, 3])
    x_freq = tf.reshape(x_freq, [img_size[0], img_size[1], -1])
    return x_freq, clean_tf

def build_rain100_dataset(root_dir, batch_size=8, img_size=(256, 256), freq_mode='real_imag'):
    all_files = glob(os.path.join(root_dir, '*'))
    rainy_files = sorted([f for f in all_files if os.path.basename(f).lower().startswith('rain-')])
    clean_files = sorted([f for f in all_files if os.path.basename(f).lower().startswith('norain-')])
    print(f"Found {len(rainy_files)} rainy images and {len(clean_files)} clean images.")

    def gen():
        for r_path, c_path in zip(rainy_files, clean_files):
            yield load_image_pair(r_path, c_path, img_size, freq_mode)

    dataset = tf.data.Dataset.from_generator(
        gen,

```

```

        output_signature=(
            tf.TensorSpec(shape=(img_size[0], img_size[1], 6), dtype=tf.float32),
            tf.TensorSpec(shape=(img_size[0], img_size[1], 3), dtype=tf.float32)
        )

    return dataset.shuffle(100).batch(batch_size).prefetch(AUTOTUNE)

# =====
# 4. Model Definitions (D-Net & N-Net)
# =====
def build_dnet(input_shape=(256, 256, 6)):
    inputs = tf.keras.Input(shape=input_shape)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(inputs)
    x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
    x = layers.Conv2D(256, 3, padding='same', activation='relu')(x)
    return models.Model(inputs, x, name='DNet')

def build_nnet(input_shape=(256, 256, 256)):
    inputs = tf.keras.Input(shape=input_shape)
    x = layers.Conv2D(128, 3, padding='same', activation='relu')(inputs)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
    x = layers.Conv2D(3, 3, padding='same', activation='sigmoid')(x)
    return models.Model(inputs, x, name='NNet')

def build_derain_model():
    input_freq = tf.keras.Input(shape=(256, 256, 6))
    dnet = build_dnet()
    nnet = build_nnet()
    features = dnet(input_freq)
    rain_map = nnet(features)
    return models.Model(inputs=input_freq, outputs=rain_map, name='DerainModel')

model = build_derain_model()
model.summary()

# =====
# 5. Training Loop
# =====
def perceptual_loss(y_true, y_pred):
    l1 = tf.reduce_mean(tf.abs(y_true - y_pred))
    ssim = 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=1.0))
    return 0.7 * l1 + 0.3 * ssim

optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)

@tf.function
def train_step(x_freq, clean):
    with tf.GradientTape() as tape:
        rain_pred = model(x_freq, training=True)
        loss = perceptual_loss(clean, rain_pred)
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
    return loss

def train_model(dataset, epochs=5):
    for epoch in range(epochs):
        total_loss = 0
        num_batches = 0
        for x_freq, clean in dataset:
            loss = train_step(x_freq, clean)
            total_loss += loss.numpy()
            num_batches += 1
        if num_batches > 0:
            print(f"Epoch {epoch + 1}, Loss: {total_loss / num_batches:.4f}")
        else:
            print(f"Epoch {epoch + 1}, No batches processed")

train_dataset = build_rain100_dataset(kaggle_dataset_path, batch_size=8)
train_model(train_dataset, epochs=5)

# =====
# 6. Inference & Enhancement
# =====
def refinement_block(img):
    x = tf.expand_dims(img, 0)
    x = layers.Conv2D(32, 3, padding='same', activation='relu')(x)
    x = layers.Conv2D(3, 3, padding='same', activation='sigmoid')(x)

```

```

        return tf.squeeze(x, 0)

def enhance_output_image(img):
    # Brightness, contrast, and detail enhancement using CLAHE
    gamma = 1.1
    img_gamma = np.power(img, 1.0 / gamma)
    img_uint8 = np.uint8(np.clip(img_gamma * 255, 0, 255))
    lab = cv2.cvtColor(img_uint8, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    cl = clahe.apply(l)
    lab_clahe = cv2.merge([cl, a, b])
    final = cv2.cvtColor(lab_clahe, cv2.COLOR_LAB2RGB)
    return np.clip(final / 255.0, 0, 1)

def sharpen_image(img):
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
    img_uint8 = np.uint8(np.clip(img * 255, 0, 255))
    sharp = cv2.filter2D(img_uint8, -1, kernel)
    return np.clip(sharp / 255.0, 0, 1)

def derain_image(rainy_path):
    rainy = Image.open(rainy_path).convert('RGB').resize((256, 256))
    rainy_np = np.array(rainy) / 255.0
    rainy_tf = tf.convert_to_tensor(rainy_np, dtype=tf.float32)

    # Frequency domain conversion
    x_freq = to_frequency_domain(rainy_tf, mode='real_imag')
    x_freq = tf.transpose(x_freq, perm=[1, 2, 0, 3])
    x_freq = tf.reshape(x_freq, [1, 256, 256, 6])

    rain_map = model(x_freq, training=False)[0]
    freq_resized = tf.reshape(x_freq[0], [256, 256, 3, 2])
    rainy_reconstructed = from_frequency_domain(freq_resized)

    clean_pred = tf.clip_by_value(rainy_reconstructed - rain_map, 0.0, 1.0)
    #clean_pred = refinement_block(clean_pred)
    clean_pred = sharpen_image(clean_pred)
    #clean_pred = enhance_output_image(clean_pred)
    return clean_pred

# Example usage
test_images = glob(os.path.join(kaggle_dataset_path, "rain-001.png"))
if test_images:
    img_path = test_images[0]
    output = derain_image(img_path)
    original = Image.open(img_path).convert('RGB').resize((256, 256))

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1); plt.imshow(original); plt.title('Original Rainy Image'); plt.axis('off')
    plt.subplot(1, 2, 2); plt.imshow(output); plt.title('Derained Output'); plt.axis('off')
    plt.show()
else:
    print("⚠ No test images found in dataset path for example usage!")

```

Using Colab cache for faster access to the 'rain-dataset' dataset.

✓ Dataset downloaded at: /kaggle/input/rain-dataset

Using TensorFlow version: 2.19.0

Model: "DerainModel"

| Layer (type) | Output Shape | Param # |
|-----------------------------|-----------------------|---------|
| input_layer_12 (InputLayer) | (None, 256, 256, 6) | 0 |
| DNet (Functional) | (None, 256, 256, 256) | 372,544 |
| NNet (Functional) | (None, 256, 256, 3) | 370,563 |

Total params: 743,107 (2.83 MB)

Trainable params: 743,107 (2.83 MB)

Non-trainable params: 0 (0.00 B)

Found 100 rainy images and 100 clean images.

Epoch 1, Loss: 0.3905

Epoch 2, Loss: 0.3505

Epoch 3, Loss: 0.3369

Epoch 4, Loss: 0.3327

Epoch 5, Loss: 0.3287

Original Rainy Image



Derained Output

