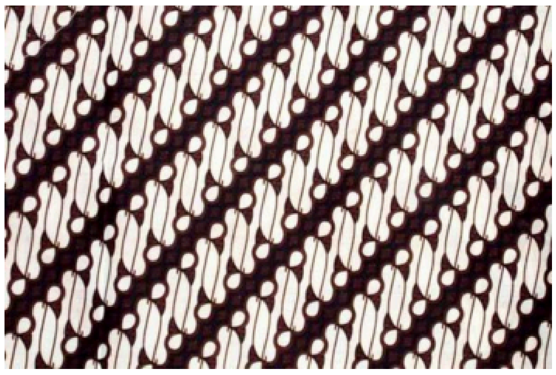
```
# ===============================
#  Natural Style Transfer - IKMI
#  Content  : Kelinci
#  Style    : Batik
# ===============================

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import PIL.Image

# -----------------------------------
# 1. Fungsi untuk load & resize image
# -----------------------------------
def load_img(path_to_img):
    max_dim = 512
    img = PIL.Image.open(path_to_img)
    long = max(img.size)
    scale = max_dim / long
    img = img.resize((round(img.size[0]*scale), round(img.size[1]*scale)))
    img = np.array(img)

    img = tf.convert_to_tensor(img)
    img = tf.image.convert_image_dtype(img, tf.float32)
    img = img[tf.newaxis, :]
    return img

def imshow(image, title=None):
    if len(image.shape) > 3:
        image = tf.squeeze(image, axis=0)
    plt.imshow(image)
    if title:
        plt.title(title)
    plt.axis("off")

# -----------------------------------
# 2. Upload gambar content & style
# -----------------------------------

# ==== GANTI path jika perlu ====
content_path = "/content/drive/MyDrive/Deep learning smstr 6/gambar/kelinci.jpg"   # upload gambar kelinc
style_path   = "/content/drive/MyDrive/Deep learning smstr 6/gambar/motif.jpg"     # upload gambar batik

content_image = load_img(content_path)
style_image = load_img(style_path)

plt.figure(figsize=(10,5))
plt.subplot(1,2,1); imshow(content_image, "Content (Kelinci)")
plt.subplot(1,2,2); imshow(style_image, "Style (Batik)")
plt.show()

# -----------------------------------
# 3. Load VGG19 untuk NST
# -----------------------------------
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
vgg.trainable = False

# Layer style & content
content_layers = ['block5_conv2']
style_layers = ['block1_conv1','block2_conv1','block3_conv1','block4_conv1','block5_conv1']

num_content_layers = len(content_layers)
num_style_layers = len(style_layers)

# -----------------------------------
# 4. Model extractor fitur
# -----------------------------------
def vgg_layers(layer_names):
    outputs = [vgg.get_layer(name).output for name in layer_names]
    model = tf.keras.Model([vgg.input], outputs)
    return model

style_extractor = vgg_layers(style_layers)
content_extractor = vgg_layers(content_layers)

# Preprocess
def preprocess(image):
    return tf.keras.applications.vgg19.preprocess_input(image*255.0)

# ----------------------------------
# 5. Model NST
# ----------------------------------
class StyleContentModel(tf.keras.Model):
    def __init__(self, style_layers, content_layers):
        super().__init__()
        self.vgg = vgg_layers(style_layers + content_layers)
        self.style_layers = style_layers
        self.content_layers = content_layers
        self.num_style_layers = len(style_layers)
        self.vgg.trainable = False

    def call(self, inputs):
        x = preprocess(inputs)
        outputs = self.vgg(x)
        style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                          outputs[self.num_style_layers:])
        style_outputs = [gram_matrix(style_output)
                         for style_output in style_outputs]

        content_dict = {content_name: value
                        for content_name, value
                        in zip(self.content_layers, content_outputs)}

        style_dict = {style_name: value
                      for style_name, value
                      in zip(self.style_layers, style_outputs)}

        return {'content': content_dict, 'style': style_dict}

# Gram matrix
def gram_matrix(input_tensor):
    result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
    num_locations = tf.cast(tf.shape(input_tensor)[1]*tf.shape(input_tensor)[2], tf.float32)
    return result/(num_locations)

extractor = StyleContentModel(style_layers, content_layers)

style_targets = extractor(style_image)['style']
content_targets = extractor(content_image)['content']

# ----------------------------------
# 6. Training NST
# ----------------------------------
image = tf.Variable(content_image)

optimizer = tf.optimizers.Adam(learning_rate=0.02)
style_weight = 1e-2
content_weight = 1e4

@tf.function
def train_step(image):
    with tf.GradientTape() as tape:
        outputs = extractor(image)
        style_outputs = outputs['style']
        content_outputs = outputs['content']

        style_loss = tf.add_n([tf.reduce_mean((style_outputs[name]-style_targets[name])**2)
                               for name in style_outputs.keys()])
        style_loss *= style_weight / num_style_layers
        content_loss = tf.add_n([tf.reduce_mean((content_outputs[name]-content_targets[name])**2)
                                 for name in content_outputs.keys()])
        content_loss *= content_weight / num_content_layers

        loss = style_loss + content_loss

    grad = tape.gradient(loss, image)
    optimizer.apply_gradients([(grad, image)])
    image.assign(tf.clip_by_value(image, 0.0, 1.0))

# ==================================
# 7. Jalankan proses NST
# ==================================
import time
start = time.time()

epochs = 3
steps_per_epoch = 200

for n in range(epochs):
    for m in range(steps_per_epoch):
        train_step(image)
    print(f"Epoch {n+1} selesai")

end = time.time()
print("Total waktu:", end-start, "detik")

# ----------------------------------
# 8. Tampilkan hasil akhir
# ----------------------------------
plt.figure(figsize=(8,8))
imshow(image, "Hasil Style Transfer (Kelinci + Batik)")
plt.show()

# Simpan file
final_path = "/content/hasil_style_transfer.jpg"
PIL.Image.fromarray((image.numpy()[0]*255).astype(np.uint8)).save(final_path)
print("Gambar disimpan:", final_path)
```

Content (Kelinci)

Style (Batik)



```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_t
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1 selesai
Epoch 2 selesai
Epoch 3 selesai
Total waktu: 5585.118232011795 detik
```

Hasil Style Transfer (Kelinci + Batik)