**building a CNN to transfer the style of one image onto another (e.g., turning a photograph into a painting)**

**Code:**

import tensorflow as tf

from tensorflow.keras.applications import vgg19

from tensorflow.keras.models import Model

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

# Helper Functions

def load\_and\_process\_image(image\_path, target\_size=(224, 224)):

    img = Image.open()

    img = img.resize(target\_size)

    img = tf.keras.preprocessing.image.img\_to\_array(img)

    img = np.expand\_dims(img, axis=0)

    img = vgg19.preprocess\_input(img)

    return img

def deprocess\_image(processed\_img):

    img = processed\_img.copy()

    img = img.reshape((img.shape[1], img.shape[2], 3))

    img[:, :, 0] += 103.939

    img[:, :, 1] += 116.779

    img[:, :, 2] += 123.68

    img = img[:, :, ::-1]

    img = np.clip(img, 0, 255).astype('uint8')

    return img

# Compute Content Loss

def compute\_content\_loss(base\_content, target):

    return tf.reduce\_mean(tf.square(base\_content - target))

# Compute Style Loss

def gram\_matrix(input\_tensor):

    result = tf.linalg.einsum('bijc,bijd->bcd', input\_tensor, input\_tensor)

    input\_shape = tf.shape(input\_tensor)

    num\_locations = tf.cast(input\_shape[1] \* input\_shape[2], tf.float32)

    return result / num\_locations

def compute\_style\_loss(base\_style, gram\_target):

    gram\_style = gram\_matrix(base\_style)

    return tf.reduce\_mean(tf.square(gram\_style - gram\_target))

# Load the VGG19 Model

def get\_model():

    vgg = vgg19.VGG19(weights="imagenet", include\_top=False)

    vgg.trainable = False

    content\_layers = ["block5\_conv2"]

    style\_layers = [

        "block1\_conv1",

        "block2\_conv1",

        "block3\_conv1",

        "block4\_conv1",

"block5\_conv1",

    ]

    content\_outputs = [vgg.get\_layer(name).output for name in content\_layers]

    style\_outputs = [vgg.get\_layer(name).output for name in style\_layers]

    model\_outputs = style\_outputs + content\_outputs

    return Model(vgg.input, model\_outputs)

# Style Transfer Class

class StyleTransfer:

    def \_\_init\_\_(self, content\_path, style\_path, content\_weight=1e4, style\_weight=1e-2):

        self.content\_image = load\_and\_process\_image(content\_path)

        self.style\_image = load\_and\_process\_image(style\_path)

        self.generated\_image = tf.Variable(self.content\_image, dtype=tf.float32)

        self.model = get\_model()

        self.content\_weight = content\_weight

        self.style\_weight = style\_weight

        self.optimizer = tf.optimizers.Adam(learning\_rate=5.0)

        self.style\_layers = 5

        self.content\_layers = 1

    def compute\_loss(self):

        model\_outputs = self.model(tf.concat([self.style\_image, self.content\_image, self.generated\_image], axis=0))

        style\_features = model\_outputs[:self.style\_layers]

        content\_features = model\_outputs[self.style\_layers:]

        # Style loss

        style\_loss = tf.add\_n([compute\_style\_loss(style\_features[i], gram\_matrix(style\_features[i]))

                               for i in range(self.style\_layers)])

        style\_loss \*= self.style\_weight / self.style\_layers

        # Content loss

        content\_loss = compute\_content\_loss(content\_features[-1], content\_features[-1])

        content\_loss \*= self.content\_weight / self.content\_layers

        # Total loss

        total\_loss = style\_loss + content\_loss

        return total\_loss

    def train\_step(self):

        with tf.GradientTape() as tape:

            loss = self.compute\_loss()

        grad = tape.gradient(loss, self.generated\_image)

        self.optimizer.apply\_gradients([(grad, self.generated\_image)])

        self.generated\_image.assign(tf.clip\_by\_value(self.generated\_image, -1.0, 1.0))

    def transfer\_style(self, epochs=1000):

        for epoch in range(epochs):

            self.train\_step()

            if epoch % 100 == 0:

                print(f"Epoch {epoch}: Loss = {self.compute\_loss().numpy()}")

        return deprocess\_image(self.generated\_image.numpy())

# Main Code to Execute Style Transfer

if \_\_name\_\_ == "\_\_main\_\_":

    content\_image\_path = "content.jpg"  # Path to your content image

    style\_image\_path = "style.jpg"    # Path to your style image

    style\_transfer = StyleTransfer(content\_image\_path, style\_ima**ge\_path)**

result = style\_transfer.transfer\_style(epochs=1000)

    plt.imshow(result)

    plt.axis('off')

    plt.show()

**Output:**

