Họ tên : Nguyễn Quang Anh

Mã sinh viên : B21DCDT036

CHƯƠNG 11 : Học Sâu trong Xử Lý Hình Ảnh - Phát Hiện Đối Tượng và Các vấn đề khác

Các chủ đề sẽ được trình bày trong chương này bao gồm:

* **Mô hình hoàn toàn tích chập (Fully Convolutional Model) để phát hiện đối tượng: YOLO (v2)**
* **Phân đoạn sâu với DeepLab (v3)**
* **Học chuyển giao (Transfer Learning): là gì và khi nào nên sử dụng**
* **Chuyển phong cách sâu (Deep Style Transfer) với cv2 sử dụng mô hình học sâu đã được huấn luyện sẵn dựa trên torch**

THUẬT TOÁN 1 : use DeepLab V3+ model for semantic segmentation

**Giới thiệu về thuật toán:**

**DeepLab V3+** là một mô hình phân đoạn ngữ nghĩa ảnh tiên tiến sử dụng mạng học sâu. Thuật toán này sử dụng kiến trúc mạng Convolutional Neural Networks (CNNs) với các cải tiến như **Atrous Convolution** và **Spatial Pyramid Pooling** để phân đoạn các đối tượng trong ảnh ở mức pixel. DeepLab V3+ đặc biệt mạnh mẽ trong việc phân loại các đối tượng phức tạp, đặc biệt là trong các ảnh có chứa nhiều lớp và các đối tượng có kích thước thay đổi. Mô hình này được sử dụng rộng rãi trong các ứng dụng như phân đoạn cảnh vật, phân tích video, và nhận dạng đối tượng.

**Các bước để thực hiện thuật toán:**

* + .

**B1: Chuẩn bị ảnh đầu vào**:

* + Đọc ảnh từ thư mục input.
  + Thay đổi kích thước ảnh sao cho chiều dài hoặc chiều rộng không vượt quá 512px.
  + Chuẩn hóa giá trị của ảnh để đảm bảo các giá trị pixel nằm trong phạm vi từ -1 đến 1.

**B2 :Thêm Padding**:

* + Thêm padding để ảnh có kích thước vuông (512x512px) vì mô hình DeepLab V3+ yêu cầu ảnh đầu vào có kích thước cố định.

**B3: Dự đoán phân đoạn**:

* + Sử dụng mô hình Deeplabv3+ để thực hiện phân đoạn ảnh. Mô hình sẽ phân đoạn ảnh thành các vùng tương ứng với các nhãn (lớp) khác nhau.

**B4: Hiển thị kết quả phân đoạn**:

* + Sử dụng matplotlib để hiển thị ảnh phân đoạn với một bảng màu (inferno).
  + Ảnh phân đoạn sẽ có màu sắc khác nhau cho từng đối tượng hoặc lớp trong ảnh.

**B5 :Lưu kết quả phân đoạn**:

* + Lưu kết quả phân đoạn dưới dạng ảnh mới trong thư mục output.

**Giải thích các câu lệnh mã nguồn:**

import sys  
import os  
from matplotlib import pyplot as pylab  
import cv2 # Dùng để thay đổi kích thước ảnh  
import numpy as np  
  
# Thêm thư mục chứa 'model.py' vào sys.path  
path\_to\_model\_dir = os.path.join(os.path.dirname(\_\_file\_\_), 'keras-deeplab-v3-plus-master')  
sys.path.append(path\_to\_model\_dir)  
  
# Kiểm tra lại sys.path để xác nhận thư mục đã được thêm đúng cách  
print("sys.path:", sys.path)  
  
# Import từ model.py trong thư mục 'keras-deeplab-v3-plus-master'  
try:  
 from model import Deeplabv3  
except ImportError as e:  
 print("Lỗi khi import: ", e)  
 sys.exit(1)  
  
# Khởi tạo mô hình Deeplabv3+  
deeplab\_model = Deeplabv3()  
  
# Định nghĩa đường dẫn cho ảnh đầu vào và đầu ra  
pathIn = 'input' # Đường dẫn thư mục chứa ảnh đầu vào  
pathOut = 'output' # Đường dẫn thư mục lưu ảnh phân đoạn  
  
# Đọc ảnh đầu vào từ thư mục 'input'  
img = pylab.imread(pathIn + "/cycle.jpg")  
  
# Thay đổi kích thước ảnh sao cho chiều dài hoặc chiều rộng không vượt quá 512px  
w, h, \_ = img.shape  
ratio = 512. / np.max([w, h])  
resized = cv2.resize(img, (int(ratio \* h), int(ratio \* w)))  
  
# Chuẩn hóa ảnh (đưa giá trị vào khoảng [-1, 1])  
resized = resized / 127.5 - 1.  
  
# Thêm padding để ảnh có kích thước vuông (512x512)  
pad\_x = int(512 - resized.shape[0])  
resized2 = np.pad(resized, ((0, pad\_x), (0, 0), (0, 0)), mode='constant')  
  
# Dự đoán phân đoạn ảnh với mô hình Deeplabv3+  
res = deeplab\_model.predict(np.expand\_dims(resized2, 0))  
  
# Lấy nhãn có xác suất cao nhất  
labels = np.argmax(res.squeeze(), -1)  
  
# Hiển thị ảnh phân đoạn với cmap 'inferno'  
pylab.figure(figsize=(20, 20))  
pylab.imshow(labels[:-pad\_x], cmap='inferno')  
pylab.axis('off') # Tắt trục  
pylab.colorbar() # Hiển thị thanh màu  
pylab.show()  
  
# Lưu kết quả phân đoạn vào thư mục output  
pylab.savefig(pathOut + "\\segmented.jpg", bbox\_inches='tight', pad\_inches=0)  
pylab.close()

**Nhập các thư viện cần thiết:**

import sys

import os

from matplotlib import pyplot as pylab

import cv2 # Dùng để thay đổi kích thước ảnh

import numpy as np

* **sys**: Thư viện dùng để thao tác với các tham số và cấu hình của môi trường Python. Cụ thể, ở đây sys.path được dùng để thêm thư mục chứa các mô-đun cần thiết vào đường dẫn tìm kiếm của Python.
* **os**: Thư viện làm việc với các thao tác hệ điều hành, như xử lý đường dẫn tệp.
* **matplotlib.pyplot as pylab**: Thư viện vẽ đồ họa, được dùng để hiển thị ảnh và biểu đồ. pylab là một alias của pyplot giúp dễ dàng truy cập các hàm vẽ.
* **cv2**: Thư viện OpenCV (Open Source Computer Vision Library), được dùng để thực hiện các thao tác xử lý ảnh, chẳng hạn như thay đổi kích thước ảnh.
* **numpy**: Thư viện dùng cho các phép toán số học trên mảng và ma trận, rất hữu ích trong việc xử lý ảnh.

**. Thêm thư mục chứa model.py vào sys.path:**

path\_to\_model\_dir = os.path.join(os.path.dirname(\_\_file\_\_), 'keras-deeplab-v3-plus-master')

sys.path.append(path\_to\_model\_dir)

* **os.path.dirname(\_\_file\_\_)**: Lấy đường dẫn của tệp script hiện tại.
* **os.path.join()**: Kết hợp các thành phần đường dẫn lại với nhau, tạo thành một đường dẫn hợp lệ đến thư mục keras-deeplab-v3-plus-master.
* **sys.path.append()**: Thêm thư mục path\_to\_model\_dir vào danh sách các thư mục mà Python tìm kiếm mô-đun. Điều này giúp Python có thể tìm thấy tệp model.py trong thư mục này khi bạn gọi from model import Deeplabv3.

**. Kiểm tra lại sys.path:**

print("sys.path:", sys.path)

* **sys.path** là một danh sách chứa các thư mục mà Python sử dụng để tìm kiếm các mô-đun. Câu lệnh này in ra giá trị hiện tại của sys.path, giúp xác nhận rằng thư mục chứa mô-đun model.py đã được thêm vào đúng cách.

**Nhập mô-đun Deeplabv3 từ model.py:**

try:

from model import Deeplabv3

except ImportError as e:

print("Lỗi khi import: ", e)

sys.exit(1)

* **from model import Deeplabv3**: Cố gắng nhập lớp Deeplabv3 từ tệp model.py trong thư mục keras-deeplab-v3-plus-master.
* **try-except**: Nếu không thể nhập mô-đun (do lỗi như không tìm thấy tệp model.py), sẽ in ra thông báo lỗi và dừng chương trình với sys.exit(1).

**. Khởi tạo mô hình Deeplabv3+:**

deeplab\_model = Deeplabv3()

* **Deeplabv3()**: Tạo một đối tượng mô hình Deeplabv3+. Mô hình này sẽ được sử dụng để dự đoán phân đoạn ngữ nghĩa cho ảnh.

**. Định nghĩa đường dẫn cho ảnh đầu vào và đầu ra:**

pathIn = 'input' # Đường dẫn thư mục chứa ảnh đầu vào

pathOut = 'output' # Đường dẫn thư mục lưu ảnh phân đoạn

* **pathIn**: Đường dẫn đến thư mục chứa ảnh đầu vào (input), nơi bạn sẽ lưu ảnh cần phân đoạn.
* **pathOut**: Đường dẫn đến thư mục nơi lưu ảnh phân đoạn (output).

**Đọc ảnh đầu vào từ thư mục input:**

img = pylab.imread(pathIn + "/cycle.jpg")

* **pylab.imread()**: Đọc ảnh từ tệp cycle.jpg trong thư mục input. Ảnh này sẽ được đưa vào mô hình Deeplabv3+ để phân đoạn.

**Thay đổi kích thước ảnh sao cho chiều dài hoặc chiều rộng không vượt quá 512px:**

w, h, \_ = img.shape

ratio = 512. / np.max([w, h])

resized = cv2.resize(img, (int(ratio \* h), int(ratio \* w)))

* **img.shape**: Lấy kích thước (chiều dài, chiều rộng) của ảnh.
* **np.max([w, h])**: Tìm chiều rộng hoặc chiều dài lớn nhất của ảnh.
* **ratio = 512. / np.max([w, h])**: Tính tỷ lệ thay đổi kích thước sao cho chiều dài hoặc chiều rộng không vượt quá 512px.
* **cv2.resize()**: Thay đổi kích thước ảnh theo tỷ lệ tính được.

**Chuẩn hóa ảnh (đưa giá trị vào khoảng [-1, 1]):**

resized = resized / 127.5 - 1.

* **Chuẩn hóa**: Chia giá trị pixel của ảnh cho 127.5 và trừ đi 1, giúp đưa giá trị của các pixel vào khoảng từ -1 đến 1. Đây là bước cần thiết để mô hình Deeplabv3+ có thể xử lý hiệu quả.

**. Thêm padding để ảnh có kích thước vuông (512x512):**

pad\_x = int(512 - resized.shape[0])

resized2 = np.pad(resized, ((0, pad\_x), (0, 0), (0, 0)), mode='constant')

* **np.pad()**: Thêm padding vào ảnh để chiều cao (hoặc chiều rộng) của ảnh trở thành 512px, tạo ảnh vuông. Nếu ảnh không đủ kích thước 512px, nó sẽ được bổ sung thêm các pixel có giá trị 0 (padding).

**Dự đoán phân đoạn ảnh với mô hình Deeplabv3+:**

res = deeplab\_model.predict(np.expand\_dims(resized2, 0))

* **np.expand\_dims(resized2, 0)**: Thêm một chiều vào đầu mảng ảnh, chuyển ảnh từ định dạng (H, W, C) thành (1, H, W, C) để phù hợp với đầu vào của mô hình.
* **deeplab\_model.predict()**: Sử dụng mô hình Deeplabv3+ để dự đoán phân đoạn ảnh.

**. Lấy nhãn có xác suất cao nhất:**

labels = np.argmax(res.squeeze(), -1)

* **res.squeeze()**: Loại bỏ các chiều dư thừa của mảng kết quả dự đoán.
* **np.argmax(..., -1)**: Lấy nhãn có xác suất cao nhất cho mỗi pixel trong ảnh phân đoạn.

**. Hiển thị ảnh phân đoạn với colormap 'inferno':**

pylab.figure(figsize=(20, 20))

pylab.imshow(labels[:-pad\_x], cmap='inferno')

pylab.axis('off')

pylab.colorbar()

pylab.show()

* **pylab.figure()**: Tạo một hình vẽ với kích thước lớn.
* **pylab.imshow()**: Hiển thị ảnh phân đoạn với colormap inferno, giúp các lớp phân đoạn được hiển thị với màu sắc khác nhau.
* **pylab.axis('off')**: Tắt các trục của ảnh.
* **pylab.colorbar()**: Hiển thị thanh màu bên cạnh ảnh để chỉ định các giá trị nhãn.
* **pylab.show()**: Hiển thị hình vẽ.

**. Lưu kết quả phân đoạn vào thư mục output:**

pylab.savefig(pathOut + "\\segmented.jpg", bbox\_inches='tight', pad\_inches=0)

pylab.close()

* **pylab.savefig()**: Lưu ảnh phân đoạn vào thư mục output dưới tên segmented.jpg.
* **bbox\_inches='tight'**: Cắt bớt không gian thừa xung quanh ảnh.
* **pad\_inches=0**: Loại bỏ padding thừa

THUẬT TOÁN 2 : Neural Style Transfer

1,Giới thiệu thuật toán :

**Neural Style Transfer (NST)** là một kỹ thuật học sâu dùng để chuyển giao phong cách giữa hai bức ảnh. Kỹ thuật này tìm cách tối ưu hóa một ảnh đầu vào sao cho ảnh đó có **nội dung** giống với một ảnh nội dung và **phong cách** giống với một ảnh phong cách.

2, Các bước thực hiện :

B1: **Chọn ảnh nội dung và ảnh phong cách**:

* **Ảnh nội dung**: Là bức ảnh bạn muốn giữ nguyên nội dung (ví dụ: bức tranh Mona Lisa).
* **Ảnh phong cách**: Là bức ảnh có phong cách mà bạn muốn áp dụng cho ảnh nội dung (ví dụ: bức tranh "Starry Night" của Van Gogh).

B2: **Chuyển ảnh thành Tensor và chuẩn hóa**:

* Ảnh được chuyển thành tensor (mảng dữ liệu), có thể được xử lý bởi mạng nơ-ron.
* Ảnh sẽ được chuẩn hóa (giảm giá trị của các pixel xuống một phạm vi chuẩn) để giúp mạng học dễ dàng hơn.

B3: **Tải mô hình VGG19**:

* Mô hình **VGG19** đã được huấn luyện trước trên bộ dữ liệu ImageNet. Nó sẽ được sử dụng để trích xuất các đặc trưng từ ảnh.

B4: **Tính toán Loss (mất mát):**

* **Content Loss**: Đo lường sự khác biệt giữa các đặc trưng của ảnh nội dung và ảnh đầu ra tại các lớp nhất định của mạng.
* **Style Loss**: Đo lường sự khác biệt giữa các đặc trưng phong cách (được đo bằng ma trận Gram) của ảnh phong cách và ảnh đầu ra.

B5: **Tối ưu hóa ảnh đầu vào**:

* Sử dụng thuật toán tối ưu hóa (như LBFGS) để tối ưu hóa ảnh đầu vào sao cho giảm thiểu cả **content loss** và **style loss**.

B6: **Kết quả đầu ra**:

* Sau khi quá trình tối ưu hóa hoàn thành, bạn có ảnh đầu ra kết hợp phong cách của ảnh phong cách và nội dung của ảnh nội dung.

3, Giải thích đoạn mã nguồn :

import torch  
import torch.optim as optim  
import torchvision.transforms as transforms  
from torch.autograd import Variable  
from torch import nn  
from torchvision import models  
from PIL import Image  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Tải ảnh content và style  
content\_image\_path = r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\monalisa.jpg"  
style\_image\_path = r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\starry\_night.jpg"  
  
  
# Chuyển ảnh thành Tensor và chuẩn hóa  
def image\_loader(image\_name, imsize=(512, 512)):  
 loader = transforms.Compose([  
 transforms.Resize(imsize),  
 transforms.ToTensor(),  
 transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),  
 ])  
 image = Image.open(image\_name)  
 image = loader(image).unsqueeze(0)  
 return image  
  
  
content\_image = image\_loader(content\_image\_path)  
style\_image = image\_loader(style\_image\_path)  
  
  
# Hiển thị ảnh content và style  
def imshow(tensor):  
 image = tensor.cpu().clone() # clone the tensor to avoid changing the original tensor  
 image = image.squeeze(0) # remove the batch dimension  
 image = transforms.ToPILImage()(image)  
 plt.imshow(image)  
 plt.axis('off')  
 plt.show()  
  
  
# Hiển thị ảnh  
imshow(content\_image)  
imshow(style\_image)  
  
# Load pre-trained VGG19 model  
cnn = models.vgg19(pretrained=True).features  
  
# Chuyển model sang chế độ evaluation  
for param in cnn.parameters():  
 param.requires\_grad = False  
  
# Di chuyển model và ảnh vào GPU nếu có  
device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  
cnn = cnn.to(device)  
content\_image = content\_image.to(device)  
style\_image = style\_image.to(device)  
  
  
# Define các loss layers và các trọng số cho các loss (content và style loss)  
class ContentLoss(nn.Module):  
 def \_\_init\_\_(self, target):  
 super(ContentLoss, self).\_\_init\_\_()  
 self.target = target.detach()  
  
 def forward(self, x):  
 loss = nn.functional.mse\_loss(x, self.target)  
 return loss  
  
  
class StyleLoss(nn.Module):  
 def \_\_init\_\_(self, target):  
 super(StyleLoss, self).\_\_init\_\_()  
 self.target = self.gram\_matrix(target).detach()  
  
 def gram\_matrix(self, x):  
 batch\_size, h, w, f\_map\_num = x.size()  
 features = x.view(batch\_size, f\_map\_num, h \* w)  
 G = torch.bmm(features, features.transpose(1, 2))  
 return G.div(f\_map\_num \* h \* w)  
  
 def forward(self, x):  
 G = self.gram\_matrix(x)  
 loss = nn.functional.mse\_loss(G, self.target)  
 return loss  
  
  
# Tạo model với các lớp chứa content loss và style loss  
def get\_style\_model\_and\_losses(cnn, content\_img, style\_img):  
 cnn = cnn.clone()  
  
 content\_losses = []  
 style\_losses = []  
  
 # Thêm content loss và style loss vào các lớp  
 model = nn.Sequential()  
 i = 0  
 for layer in cnn.children():  
 model.add\_module(str(i), layer)  
 if isinstance(layer, nn.Conv2d):  
 i += 1  
 if i == 4:  
 content\_loss = ContentLoss(content\_img)  
 model.add\_module(str(i), content\_loss)  
 content\_losses.append(content\_loss)  
 if i == 2:  
 style\_loss = StyleLoss(style\_img)  
 model.add\_module(str(i), style\_loss)  
 style\_losses.append(style\_loss)  
 return model, content\_losses, style\_losses  
  
  
# Lưu model  
model, content\_losses, style\_losses = get\_style\_model\_and\_losses(cnn, content\_image, style\_image)  
  
# Cập nhật các tham số  
input\_img = content\_image.clone()  
  
# Tối ưu hóa input image  
optimizer = optim.LBFGS(input\_img)  
  
# Chu trình huấn luyện  
run = [0]  
while run[0] <= 300:  
 def closure():  
 input\_img.data.clamp\_(0, 1)  
  
 optimizer.zero\_grad()  
 model(input\_img)  
  
 style\_score = 0  
 content\_score = 0  
  
 for sl in style\_losses:  
 style\_score += sl.forward(input\_img)  
 for cl in content\_losses:  
 content\_score += cl.forward(input\_img)  
  
 loss = style\_score \* 1000000 + content\_score \* 1  
 loss.backward()  
  
 run[0] += 1  
 return loss  
  
  
 optimizer.step(closure)  
  
# Hiển thị kết quả  
imshow(input\_img)

**Cài đặt các thư viện cần thiết:**

import torch

import torch.optim as optim

import torchvision.transforms as transforms

from torch.autograd import Variable

from torch import nn

from torchvision import models

from PIL import Image

import matplotlib.pyplot as plt

import numpy as np

* **torch, torch.optim**: Dùng để thực hiện các phép toán trên tensor và tối ưu hóa.
* **torchvision.models**: Cung cấp các mô hình học sâu đã được huấn luyện trước (ví dụ: VGG19).
* **PIL (Python Imaging Library)**: Dùng để xử lý ảnh.
* **matplotlib.pyplot**: Dùng để hiển thị ảnh.
* **numpy**: Thư viện xử lý mảng.

**2. Tải ảnh nội dung và phong cách và chuyển thành Tensor:**

content\_image\_path = r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\monalisa.jpg"

style\_image\_path = r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\starry\_night.jpg"

* Định nghĩa đường dẫn đến các ảnh nội dung và phong cách.

def image\_loader(image\_name, imsize=(512, 512)):

loader = transforms.Compose([

transforms.Resize(imsize),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

])

image = Image.open(image\_name)

image = loader(image).unsqueeze(0)

return image

* **image\_loader()**: Hàm này tải ảnh từ đường dẫn và chuyển ảnh thành tensor, đồng thời chuẩn hóa giá trị pixel theo trung bình và độ lệch chuẩn của bộ dữ liệu ImageNet (vì mô hình VGG19 đã được huấn luyện trên ImageNet).

content\_image = image\_loader(content\_image\_path)

style\_image = image\_loader(style\_image\_path)

* Tải ảnh nội dung và phong cách bằng cách gọi hàm image\_loader.

**3. Hiển thị ảnh nội dung và phong cách:**

def imshow(tensor):

image = tensor.cpu().clone() # clone the tensor to avoid changing the original tensor

image = image.squeeze(0) # remove the batch dimension

image = transforms.ToPILImage()(image)

plt.imshow(image)

plt.axis('off')

plt.show()

* **imshow()**: Hàm này chuyển đổi tensor thành ảnh và hiển thị ảnh sử dụng matplotlib.

imshow(content\_image)

imshow(style\_image)

* Hiển thị ảnh nội dung và phong cách để chúng ta kiểm tra trước khi áp dụng style transfer.

**4. Chọn mô hình VGG19 và chuyển sang chế độ evaluation:**

cnn = models.vgg19(pretrained=True).features

for param in cnn.parameters():

param.requires\_grad = False

* **models.vgg19(pretrained=True)**: Tải mô hình VGG19 đã huấn luyện trước.
* **param.requires\_grad = False**: Đóng băng tất cả các tham số trong mô hình, vì chúng ta chỉ cần sử dụng mô hình để trích xuất đặc trưng mà không cần huấn luyện lại.

**5. Tạo các lớp loss:**

class ContentLoss(nn.Module):

def \_\_init\_\_(self, target):

super(ContentLoss, self).\_\_init\_\_()

self.target = target.detach()

def forward(self, x):

loss = nn.functional.mse\_loss(x, self.target)

return loss

* **ContentLoss**: Tính toán mất mát giữa đặc trưng của ảnh nội dung và ảnh đầu ra.

class StyleLoss(nn.Module):

def \_\_init\_\_(self, target):

super(StyleLoss, self).\_\_init\_\_()

self.target = self.gram\_matrix(target).detach()

def gram\_matrix(self, x):

batch\_size, h, w, f\_map\_num = x.size()

features = x.view(batch\_size, f\_map\_num, h \* w)

G = torch.bmm(features, features.transpose(1, 2))

return G.div(f\_map\_num \* h \* w)

def forward(self, x):

G = self.gram\_matrix(x)

loss = nn.functional.mse\_loss(G, self.target)

return loss

* **StyleLoss**: Tính toán mất mát giữa phong cách của ảnh và ảnh đầu ra, sử dụng ma trận Gram để đo sự khác biệt giữa các đặc trưng.

**6. Quá trình tối ưu hóa và huấn luyện:**

optimizer = optim.LBFGS(input\_img)

* **LBFGS**: Đây là một thuật toán tối ưu hóa sử dụng trong huấn luyện mô hình học sâu, giúp tối ưu hóa ảnh đầu vào sao cho giảm thiểu các mất mát content và style.

run = [0]

while run[0] <= 300:

def closure():

input\_img.data.clamp\_(0, 1)

optimizer.zero\_grad()

model(input\_img)

style\_score = 0

content\_score = 0

for sl in style\_losses:

style\_score += sl.forward(input\_img)

for cl in content\_losses:

content\_score += cl.forward(input\_img)

loss = style\_score \* 1000000 + content\_score \* 1

loss.backward()

run[0] += 1

return loss

optimizer.step(closure)

* Đây là vòng lặp tối ưu hóa, tính toán mất mát và cập nhật ảnh đầu vào qua từng lần lặp.

**7. Hiển thị kết quả đầu ra:**

imshow(input\_img)

* Sau khi tối ưu hóa xong, ảnh đầu ra được hiển thị với phong cách của ảnh phong cách và nội dung của ảnh nội dung.