**Hadi Seyed – Lab 7**

**Dec 4, 2023**

**Part 1:**

import json

# Opening JSON file

f = open('sa\_255904.json',)

# returns JSON object as

# a dictionary

data = json.load(f)

# print(data)

import numpy as np

import matplotlib.pyplot as plt

im = plt.imread("sa\_255904.jpg")

plt.imshow(im)

plt.show()

from pycocotools import mask as mask\_utils

mask = mask\_utils.decode(data["annotations"][85]["segmentation"])

#Number of masks

print(len(data["annotations"]))

plt.imshow(mask,cmap="gray")

plt.show()

# Create an alpha channel from the inverted mask

alpha\_channel = mask.astype(float)

print(np.amax(alpha\_channel))

im\_f = im/255

print(np.amax(im\_f))

# Create a black background

black\_background = np.zeros\_like(im)

print(alpha\_channel[...,None].shape)

# Alpha blend the image with the black background

blended\_image = (im\_f \* alpha\_channel[...,None] + black\_background \* (1 - alpha\_channel[...,None]))

print(np.amax(blended\_image))

#Display the blended image

plt.imshow(blended\_image)

plt.show()

import torch

import utils

from libs.Matrix import MulLayer

from libs.models import encoder4, decoder4

# We can use both the following command

#content\_im = utils.toTensor(im/255)

content\_im = utils.toTensor(blended\_image)

style\_fn = "data/style/27.jpg"

style\_im = utils.loadImage(style\_fn)

enc\_ref = encoder4()

dec\_ref = decoder4()

matrix\_ref = MulLayer('r41')

enc\_ref.load\_state\_dict(torch.load('models/vgg\_r41.pth'))

dec\_ref.load\_state\_dict(torch.load('models/dec\_r41.pth'))

matrix\_ref.load\_state\_dict(torch.load('models/r41.pth',map\_location=torch.device('cpu')))

with torch.no\_grad():

# Reference comparison

cF\_ref = enc\_ref(content\_im)

sF\_ref = enc\_ref(style\_im)

feature\_ref,transmatrix\_ref = matrix\_ref(cF\_ref['r41'],sF\_ref['r41'])

result = dec\_ref(feature\_ref)

result = utils.toNumpy(result)

result = np.clip(result,0,1)

plt.imshow(result)

plt.show()

# Initialize an empty canvas with the same dimensions as the original image

canvas = np.zeros\_like(result)

import cv2

alpha\_channel = cv2.resize(alpha\_channel,(result.shape[1],result.shape[0]))

canvas += result\*alpha\_channel[...,None]

# Ensure that pixel values are within the valid range

#canvas = np.clip(canvas, 0, 255).astype(np.uint8)

canvas = np.clip(canvas, 0, 1)

# Display the final recombined image

plt.imshow(canvas)

plt.show()

**Part 2:**

from torchvision.io.image import read\_image

from torchvision.models.segmentation import fcn\_resnet50, FCN\_ResNet50\_Weights

from torchvision.transforms.functional import to\_pil\_image

img = read\_image("dog.jpg")

#img = read\_image("dog1.PNG")[:,:,:,:3]

# Step 1: Initialize model with the best available weights

weights = FCN\_ResNet50\_Weights.DEFAULT

model = fcn\_resnet50(weights=weights)

model.eval()

# Step 2: Initialize the inference transforms

preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms

batch = preprocess(img).unsqueeze(0)

# Step 4: Use the model and visualize the prediction

prediction = model(batch)["out"]

normalized\_masks = prediction.softmax(dim=1)

class\_to\_idx = {cls: idx for (idx, cls) in enumerate(weights.meta["categories"])}

mask = normalized\_masks[0, class\_to\_idx["dog"]]

to\_pil\_image(mask).show()

**Part 3:**

Type 1:

import torch

import utils

import numpy as np

import matplotlib.pyplot as plt

from libs.Matrix import MulLayer

from libs.models import encoder4, decoder4

content\_fn = "data/content/chicago.png"

content\_im = utils.loadImage(content\_fn)

style\_fn = "data/style/27.jpg"

style\_im = utils.loadImage(style\_fn)

save\_fn = "output.png"

#device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

enc\_ref = encoder4()

dec\_ref = decoder4()

matrix\_ref = MulLayer('r41')

enc\_ref.load\_state\_dict(torch.load('models/vgg\_r41.pth'))

dec\_ref.load\_state\_dict(torch.load('models/dec\_r41.pth'))

matrix\_ref.load\_state\_dict(torch.load('models/r41.pth',map\_location=torch.device('cpu')))

with torch.no\_grad():

# Reference comparison

cF\_ref = enc\_ref(content\_im)

sF\_ref = enc\_ref(style\_im)

feature\_ref,transmatrix\_ref = matrix\_ref(cF\_ref['r41'],sF\_ref['r41'])

result = dec\_ref(feature\_ref)

result = utils.toNumpy(result)

result = np.clip(result,0,1)

plt.imshow(result)

plt.show()

plt.imsave(save\_fn,result)

Type 2:

import torch

import utils

import numpy as np

import matplotlib.pyplot as plt

from libs.Matrix import MulLayer

from libs.models import encoder4, decoder4

content\_fn = "data/content/1.jpg"

content\_im = utils.loadImage(content\_fn)

style\_fn = "data/style/in2.jpg"

style\_im = utils.loadImage(style\_fn)

save\_fn = "output.png"

#device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

enc\_ref = encoder4()

dec\_ref = decoder4()

matrix\_ref = MulLayer('r41')

enc\_ref.load\_state\_dict(torch.load('models/vgg\_r41.pth'))

dec\_ref.load\_state\_dict(torch.load('models/dec\_r41.pth'))

matrix\_ref.load\_state\_dict(torch.load('models/r41.pth',map\_location=torch.device('cpu')))

with torch.no\_grad():

# Reference comparison

cF\_ref = enc\_ref(content\_im)

sF\_ref = enc\_ref(style\_im)

feature\_ref,transmatrix\_ref = matrix\_ref(cF\_ref['r41'],sF\_ref['r41'])

result = dec\_ref(feature\_ref)

result = utils.toNumpy(result)

result = np.clip(result,0,1)

plt.imshow(result)

plt.show()

plt.imsave(save\_fn,result)

**Feedback:**

In exploring computer vision tasks through various implementations, I've gained hands-on experience with segmentation, style transfer, Neural Networks, and image generation. The use of online demos and mobile apps has provided a user-friendly entry point, allowing us to interact with cutting-edge technologies without delving into code. I've witnessed the power of AI in tasks such as face swapping, image generation, and novel view synthesis, opening up creative possibilities and demonstrating the broad applications of computer vision.

I've encountered challenges such as handling errors related to array manipulation and learned to troubleshoot issues in image processing pipelines. Additionally, my exposure to different tools and models, like those based on neural networks, has expanded your understanding of the underlying technologies driving computer vision advancements.

**Looking Forward in Computer Vision:**

* Advances in deep learning models will likely lead to improved accuracy and robustness in computer vision tasks.
* The development of faster and more efficient algorithms will enable real-time applications of computer vision, leading to enhanced user experiences in fields like augmented reality, autonomous vehicles, and more.
* Computer vision will increasingly integrate with other disciplines, such as natural language processing and robotics, creating synergies that enhance overall AI capabilities.