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
```

```python
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.