Mosaic-Real-Time-Transfer

December 16, 2021

[ ]:
```
!git clone https://github.com/MagicShow1999/CV-Final-Project
```

```
Cloning into 'CV-Final-Project'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 59 (delta 5), reused 9 (delta 2), pack-reused 44
Unpacking objects: 100% (59/59), done.
```

[ ]:
```
%cd CV-Final-Project/
```

```
/content/CV-Final-Project
```

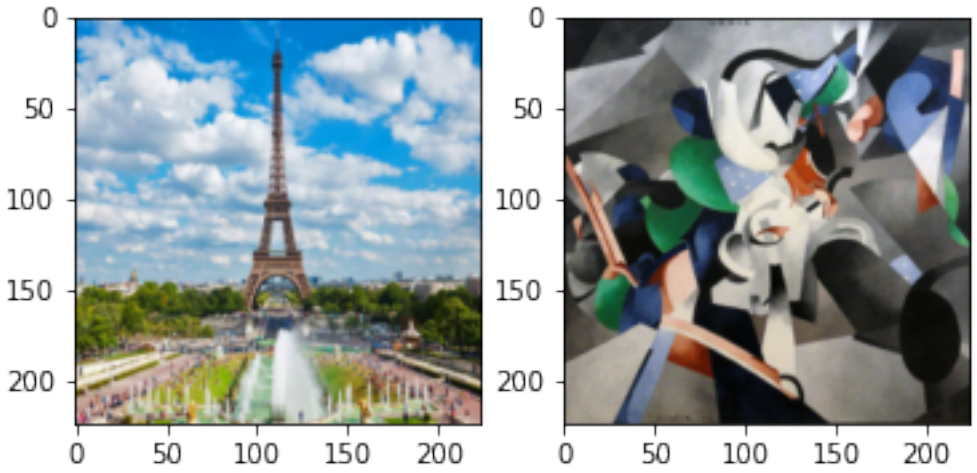
[ ]:
```
import torch
from torchvision import transforms, models, datasets
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import util
import time
import tqdm
import os

device = ("cuda" if torch.cuda.is_available() else "cpu")
```

1 1. Set up

[ ]:
```
# read image
style = util.read_image('./image/udine.jpeg', 224, 224).to(device)
content = util.read_image('./image/content_img.jpg', 224, 224).to(device)
```

[ ]:
```
# show image
fig, (ax1,ax2) = plt.subplots(1,2)
ax1.imshow(util.torchTensorToImage(content),label = "Content")
ax2.imshow(util.torchTensorToImage(style),label = "Style")
```

```
plt.show()
```



1.1 1.1 Hyperparmeter

[121]:
```
# Hyperparmeter
TRAIN_IMAGE_SIZE = 224
EPOCHS = 2
BATCH_SIZE = 4
CONTENT_WEIGHT = 1e0
STYLE_WEIGHTS = {
          "relu1_2":1e5,
          "relu2_2":1e5,
          "relu3_3":1e5,
          "relu4_3":1e5
      }# *(1e3)
LOG_INTERVAL = 500
LR = 0.001
```

1.2 1.2 Loss Network

[122]:
```
import torch
import torch.nn as nn
from torchvision import models

class LossNet(nn.Module):
  """
  The loss network is used to define a feature reconstruction loss and a style␣
 ↪reconstruction loss
  that measure differences in content and style between images
```

```
    """
    def __init__(self):
        super(LossNet, self).__init__()
        self.vgg = models.vgg16(pretrained=True).features
        self.layers = {
            '3': "relu1_2",
            '8': "relu2_2",
            '15': "relu3_3",
            '22': "relu4_3",
        }
        for p in self.vgg.parameters():
            p.requires_grad = False

    def forward(self, x):
      features = {}
      for name,layer in self.vgg._modules.items():
        x = layer(x)
        if name in self.layers:
          features[self.layers[name]] = x
      return features
```

1.3 1.3 Transform Network

1.3.1 Convolution Layer

[123]:
```
class ConvLayer(torch.nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride):
        super(ConvLayer, self).__init__()
        padding_size = kernel_size // 2
        self.reflection_pad = torch.nn.ReflectionPad2d(padding_size)
        self.conv2d = torch.nn.Conv2d(in_channels, out_channels, kernel_size,␣
↪stride)

    def forward(self, x):
        return self.conv2d(self.reflection_pad(x))
```

1.3.2 Residual Block

[124]:
```
class ResidualBlock(torch.nn.Module):
    def __init__(self, channels):
        super(ResidualBlock, self).__init__()
        self.ConvLayers = nn.Sequential(
            ConvLayer(channels, channels, 3, 1),
            torch.nn.InstanceNorm2d(channels, affine=True),
            torch.nn.ReLU(),
            ConvLayer(channels, channels, 3, 1),
            torch.nn.InstanceNorm2d(channels, affine=True)
        ) # ReLU nonlinearity following the addition is removed;
        self.relu = torch.nn.ReLU()
```

```
    def forward(self, x):
        identity = x
        x = self.ConvLayers(x)
        out = x + identity
        out = self.relu(out)
        return out
```

1.3.3 DeConvolution Layer

[125]:
```
class DeConvLayer(torch.nn.Module):

    def __init__(self, in_channels, out_channels, kernel_size, stride,␣
↪upsample=None):
        super(DeConvLayer, self).__init__()
        self.upsample = upsample
        if upsample:
            self.upsample = nn.Upsample(scale_factor=upsample, mode='nearest')
        reflection_padding = kernel_size // 2
        self.reflection_pad = nn.ReflectionPad2d(reflection_padding)
        self.conv2d = nn.Conv2d(in_channels, out_channels, kernel_size, stride)

    def forward(self, x):
        if self.upsample:
            x = self.upsample(x)
        out = self.reflection_pad(x)
        out = self.conv2d(out)
        return out
```

[126]:
```
class TransformNet(torch.nn.Module):
    """
    The image transformation network is trained to minimize a weighted␣
↪combination of loss functions
    The exact architectures of this networks can be found in the supplementary␣
↪material of the paper
    https://cs.stanford.edu/people/jcjohns/papers/fast-style/fast-style-supp.pdf
    """
    def __init__(self):
        super(TransformNet, self).__init__()
        # Convolution layers
        self.ConvLayers = nn.Sequential(
            ConvLayer(3, 32, 9, 1),
            torch.nn.InstanceNorm2d(32),
            torch.nn.ReLU(),
            ConvLayer(32, 64, 3, 2),
            torch.nn.InstanceNorm2d(64),
            torch.nn.ReLU(),
            ConvLayer(64, 128, 3, 2),
```

```
            torch.nn.InstanceNorm2d(128),
            torch.nn.ReLU()
        )

        # Residual blocks
        self.ResidualBlocks = nn.Sequential(
            ResidualBlock(128),
            ResidualBlock(128),
            ResidualBlock(128),
            ResidualBlock(128),
            ResidualBlock(128)
        )

        # Deconvolutional layers
        self.DeConvLayers = nn.Sequential(
            DeConvLayer(128, 64, kernel_size=3, stride=1, upsample=2),
            torch.nn.InstanceNorm2d(64, affine=True),

            DeConvLayer(64, 32, kernel_size=3, stride=1, upsample=2),
            torch.nn.InstanceNorm2d(32, affine=True),

            DeConvLayer(32, 3, kernel_size=9, stride=1),
            torch.nn.InstanceNorm2d(3, affine=True)
        )

    def forward(self, x):
        x = self.ConvLayers(x)
        x = self.ResidualBlocks(x)
        x = self.DeConvLayers(x)
        return x
```

2 Train

2.0.1 2.1 Prepare Training set (COCO) TODO

Currently use val2014 instead because train2014 is too big

[120]:
```
# Download dataset (COCO2017)
# https://cocodataset.org/#download
# TODO change val to train
!wget http://images.cocodataset.org/zips/val2017.zip
```

```
--2021-12-16 07:46:52--  http://images.cocodataset.org/zips/val2017.zip
Resolving images.cocodataset.org (images.cocodataset.org)... 52.217.9.188
Connecting to images.cocodataset.org
(images.cocodataset.org)|52.217.9.188|:80... connected.
HTTP request sent, awaiting response... 200 OK
```

```
Length: 815585330 (778M) [application/zip]
Saving to: val2017.zip

val2017.zip         100%[===================>] 777.80M  16.3MB/s    in 49s

2021-12-16 07:47:41 (15.9 MB/s) - val2017.zip saved [815585330/815585330]
```

[127]:
```
!rm -rf dataset
!rm -rf model_checkpoints
!rm -rf transformed_images
!mkdir dataset
!mkdir model_checkpoints
!mkdir transformed_images
```

[128]:
```
!unzip -qq val2017.zip -d /content/CV-Final-Project/dataset/val2017
!rm val2017.zip
```

[129]:
```
# Get Training Dataset
transform = util.common_transforms(TRAIN_IMAGE_SIZE, TRAIN_IMAGE_SIZE)
# ref: https://pytorch.org/vision/0.8/datasets.html#imagefolder
train_dataset = datasets.ImageFolder("/content/CV-Final-Project/dataset/",␣
 ↪transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset,␣
 ↪batch_size=BATCH_SIZE, shuffle=True)
```

2.0.2 2.2 Style Feature for Reference

[131]:
```
# Style Features
B, C, H, W = style.shape
style_features = vgg(style.expand([BATCH_SIZE, C, H, W]))
style_grams = {}
for layer, feature in style_features.items():
    style_grams[layer] = util.gram_matrix(feature)
```

2.0.3 2.3 Model & Optimizer

[130]:
```
# Model
transform_net = TransformNet().to(device)
vgg = LossNet().to(device)

# Optimizer
optimizer = torch.optim.Adam(transform_net.parameters(),lr=LR)
```

2.0.4 2.4 Training

[132]:
```
content_loss_history = []
style_loss_history = []
total_loss_history = []
begin_time = time.time()
batch_count = 0

for epoch in range(1, EPOCHS + 1):
  content_loss_sum = 0
  style_loss_sum = 0
  total_loss_sum = 0
  for batch_idx, (data, _) in enumerate(train_loader):
    curr_batch_size = data.shape[0]
    batch_count += 1
    if device == "cuda":
      torch.cuda.empty_cache() # Free-up memory

    optimizer.zero_grad()

    # Get transformed images and features generated by transformed images
    content_imgs = data.to(device)
    transformed_imgs = transform_net(content_imgs)
    content_features = vgg(content_imgs)
    transformed_features = vgg(transformed_imgs)

    # Content Loss
    MSELoss = nn.MSELoss().to(device)
    content_loss = CONTENT_WEIGHT * MSELoss(transformed_features['relu3_3'],␣
 ↪content_features['relu3_3'])
    content_loss_sum += content_loss.item()

    # Style Loss
    style_loss = 0
    for layer, feature in transformed_features.items():
      style_gram = style_grams[layer][:curr_batch_size]
      transformed_gram = util.gram_matrix(feature)
      style_loss += MSELoss(transformed_gram, style_gram) * STYLE_WEIGHTS[layer]
    style_loss_sum += style_loss.item()

    # Total Loss
    total_loss = content_loss + style_loss
    total_loss_sum += total_loss.item()

    # Backprop and Weight Update
    total_loss.backward()
    optimizer.step()
```

```
   if batch_idx % LOG_INTERVAL == 0:
       print('\nTrain Epoch: {} [{}/{} ({:.0f}%)]'.format(
           epoch, batch_idx * len(data), len(train_loader.dataset),
             100. * batch_idx / len(train_loader)))
       print('\tContent Loss: {:.6f}\tStyle Loss: {:.6f}\tTotal Loss: {:.6f}'.
↪format(
           content_loss_sum/batch_count, style_loss_sum/batch_count,␣
↪total_loss_sum/batch_count))
       print("Time elapsed:\t{} seconds".format(time.time()-begin_time))
       content_loss_history.append(content_loss_sum/batch_count)
       style_loss_history.append(style_loss_sum/batch_count)
       total_loss_history.append(total_loss_sum/batch_count)
       # Save model checkpoint
       model_file = '/content/CV-Final-Project/model_checkpoints/
↪checkpoint_{}_{}.pth'.format(epoch, batch_idx//LOG_INTERVAL)
       torch.save(transform_net.state_dict(), model_file)
       print('Saved model to ' + model_file + '.')
       # Save image
       transformed_img = transformed_imgs[0].clone().detach().unsqueeze(dim=0)
       image_path = '/content/CV-Final-Project/transformed_images/img_{}_{}.
↪png'.format(epoch, batch_idx*4)
       util.save_image(transformed_img, image_path)
 # Save model weights
 final_path = "/content/CV-Final-Project/model_checkpoints/
↪transformation_network_weight.pth"
 torch.save(transform_net.state_dict(), final_path)
 print('Saved model to ' + final_path + '.')
```

```
Train Epoch: 1 [0/5000 (0%)]
        Content Loss: 31.052502 Style Loss: 1471.257935 Total Loss: 1502.310425
Time elapsed:   0.2166118621826172 seconds
Saved model to /content/CV-Final-Project/model_checkpoints/checkpoint_1_0.pth.
Successfully save the final stylized image to: /content/CV-Final-
Project/transformed_images/img_1_0.png

Train Epoch: 1 [2000/5000 (40%)]
        Content Loss: 16.060682 Style Loss: 23.980723   Total Loss: 40.041405
Time elapsed:   72.9520115852356 seconds
Saved model to /content/CV-Final-Project/model_checkpoints/checkpoint_1_1.pth.
Successfully save the final stylized image to: /content/CV-Final-
Project/transformed_images/img_1_2000.png

Train Epoch: 1 [4000/5000 (80%)]
        Content Loss: 15.210442 Style Loss: 16.709062   Total Loss: 31.919505
Time elapsed:   145.7732515335083 seconds
Saved model to /content/CV-Final-Project/model_checkpoints/checkpoint_1_2.pth.
```

```
Successfully save the final stylized image to: /content/CV-Final-
Project/transformed_images/img_1_4000.png
Saved model to /content/CV-Final-
Project/model_checkpoints/transformation_network_weight.pth.

Train Epoch: 2 [0/5000 (0%)]
        Content Loss: 0.010151  Style Loss: 0.006639    Total Loss: 0.016790
Time elapsed:   181.87240886688232 seconds
Saved model to /content/CV-Final-Project/model_checkpoints/checkpoint_2_0.pth.
Successfully save the final stylized image to: /content/CV-Final-
Project/transformed_images/img_2_0.png

Train Epoch: 2 [2000/5000 (40%)]
        Content Loss: 3.611654  Style Loss: 2.177996    Total Loss: 5.789650
Time elapsed:   254.1175127029419 seconds
Saved model to /content/CV-Final-Project/model_checkpoints/checkpoint_2_1.pth.
Successfully save the final stylized image to: /content/CV-Final-
Project/transformed_images/img_2_2000.png

Train Epoch: 2 [4000/5000 (80%)]
        Content Loss: 5.449342  Style Loss: 3.277000    Total Loss: 8.726342
Time elapsed:   327.0498912334442 seconds
Saved model to /content/CV-Final-Project/model_checkpoints/checkpoint_2_2.pth.
Successfully save the final stylized image to: /content/CV-Final-
Project/transformed_images/img_2_4000.png
Saved model to /content/CV-Final-
Project/model_checkpoints/transformation_network_weight.pth.
```

[133]:
```
!zip -r /content/CV-Final-Project/imgs.zip /content/CV-Final-Project/
↪transformed_images
```

```
updating: content/CV-Final-Project/transformed_images/ (stored 0%)
updating: content/CV-Final-Project/transformed_images/img_1_4000.png (deflated
0%)
updating: content/CV-Final-Project/transformed_images/img_1_2000.png (deflated
0%)
updating: content/CV-Final-Project/transformed_images/img_1_0.png (deflated 0%)
  adding: content/CV-Final-Project/transformed_images/img_2_4000.png (deflated
0%)
  adding: content/CV-Final-Project/transformed_images/img_2_2000.png (deflated
0%)
  adding: content/CV-Final-Project/transformed_images/img_2_0.png (deflated 0%)
```

[134]:
```
!pwd
```

```
/content/CV-Final-Project
```

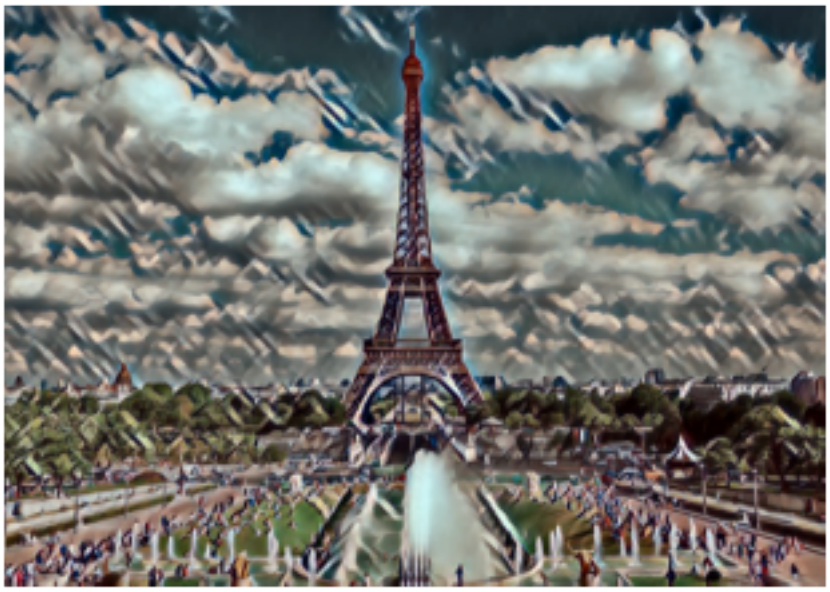
[135]:
```
cnt = len(train_dataset)//(BATCH_SIZE * LOG_INTERVAL)
_, ax = plt.subplots(cnt, 1,figsize = (8,cnt*8))
epoch = EPOCHS
for i in range(1, cnt+1):
  idx = LOG_INTERVAL * i * BATCH_SIZE
  image_name = 'img_{}_{}.png'.format(epoch, idx)
  image_path = './transformed_images/' + image_name
  sampe_image = util.read_image(image_path, TRAIN_IMAGE_SIZE, TRAIN_IMAGE_SIZE).
 ↪to(device)
  ax[i-1].set_title(image_name)
  ax[i-1].imshow(util.torchTensorToImage(sampe_image))
  ax[i-1].axis("off")
plt.show()
```

img_2_2000.png

img_2_4000.png

[139]:
```
begin_time = time.time()
content = util.read_image('/content/CV-Final-Project/image/content_img.jpg').
 ↪to(device)
output_image = transform_net(content).cpu()
print("Time taken: {}".format(time.time()-begin_time))
util.paint_image(output_image, title="output")
util.save_image(output_image, '/content/CV-Final-Project/image/udine_out1.jpg')

begin_time = time.time()
content = util.read_image('/content/CV-Final-Project/image/content_img_2.jpg').
 ↪to(device)
output_image = transform_net(content).cpu()
print("Time taken: {}".format(time.time()-begin_time))
util.paint_image(output_image, title="output")
util.save_image(output_image, '/content/CV-Final-Project/image/udine_out2.jpg')
```

```
Time taken: 0.10517358779907227
```



```
Successfully save the final stylized image to: /content/CV-Final-
Project/image/udine_out1.jpg
Time taken: 0.520117998123169
```

```
Successfully save the final stylized image to: /content/CV-Final-
Project/image/udine_out2.jpg
```

[140]:
```
!zip -r /content/CV-Final-Project/models.zip /content/CV-Final-Project/
↪model_checkpoints
```

```
  adding: content/CV-Final-Project/model_checkpoints/ (stored 0%)
  adding: content/CV-Final-Project/model_checkpoints/checkpoint_2_1.pth
(deflated 8%)
  adding: content/CV-Final-Project/model_checkpoints/checkpoint_1_0.pth
(deflated 9%)
  adding: content/CV-Final-
Project/model_checkpoints/transformation_network_weight.pth (deflated 8%)
  adding: content/CV-Final-Project/model_checkpoints/checkpoint_2_0.pth
(deflated 8%)
  adding: content/CV-Final-Project/model_checkpoints/checkpoint_1_1.pth
(deflated 8%)
  adding: content/CV-Final-Project/model_checkpoints/checkpoint_1_2.pth
(deflated 8%)
  adding: content/CV-Final-Project/model_checkpoints/checkpoint_2_2.pth
(deflated 8%)
```

Saving the notebook as pdf.

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc

from google.colab import drive
drive.mount('/content/drive')
```

[147]:
```
!cp /content/drive/MyDrive/"Colab Notebooks"/Mosaic-Real-Time-Transfer.ipynb ./
!jupyter nbconvert --to PDF "Mosaic-Real-Time-Transfer.ipynb"
```

```
[NbConvertApp] Converting notebook Mosaic-Real-Time-Transfer.ipynb to PDF
[NbConvertApp] Support files will be in Mosaic-Real-Time-Transfer_files/
[NbConvertApp] Making directory ./Mosaic-Real-Time-Transfer_files
[NbConvertApp] Making directory ./Mosaic-Real-Time-Transfer_files
[NbConvertApp] Making directory ./Mosaic-Real-Time-Transfer_files
[NbConvertApp] Making directory ./Mosaic-Real-Time-Transfer_files
[NbConvertApp] Writing 92715 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: [u'xelatex', u'./notebook.tex',
'-quiet']
[NbConvertApp] Running bibtex 1 time: [u'bibtex', u'./notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 978286 bytes to Mosaic-Real-Time-Transfer.pdf
```