SW마에스트로 실습

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

성무열

목차

코드: NeRF.ipynb - Colaboratory (google.com)

1 Settings

Imports

```
[ ]  try:
       import google.colab
       IN_COLAB = True
     except:
       IN_COLAB = False

     if IN_COLAB:
       %tensorflow_version 1.x

     import os, sys
     import tensorflow as tf
     tf.compat.v1.enable_eager_execution()

     from tqdm import tqdm_notebook as tqdm
     import numpy as np
     import matplotlib.pyplot as plt
```

- Tensorflow 사용

- render된 결과 출력 위해 matplotlib 사용

- tqdm은 진행 상태 막대그래프로 표현하는 module, 360 video rendering할 때 사용(중요하지 않음)

1 Settings

Mount

```
[ ]  from google.colab import auth
     auth.authenticate_user()

     from google.colab import drive
     drive.mount('/content/gdrive', force_remount=False)
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remou
```

```
[ ]  import os
     from pathlib import Path

     folder = "Colab Notebooks"
     project_dir = "NeRF"

     base_path = Path("/content/gdrive/My Drive/")
     project_path = base_path / folder / project_dir
     os.chdir(project_path)
     for x in list(project_path.glob("*")):
       if x.is_dir():
         dir_name = str(x.relative_to(project_path))
         os.rename(dir_name, dir_name.split(" ", 1)[0])
     print(f"현재 디렉토리 위치: {os.getcwd()}")
```

```
현재 디렉토리 위치: /content/gdrive/My Drive/Colab Notebooks/NeRF
```

- Google 드라이브 내의 폴더로 mount

- 원래 없는 코드, 데이터 직접 다루기 위해 추가함

1 Settings

Download Dataset

```
[ ]  if not os.path.exists('tiny_nerf_data.npz'):
         !wget http://cseweb.ucsd.edu/~viscomp/projects/LF/papers/ECCV20/nerf/tiny_nerf_data.npz
```

- 공식 제공해주는 dataset 다운로드

1 Settings

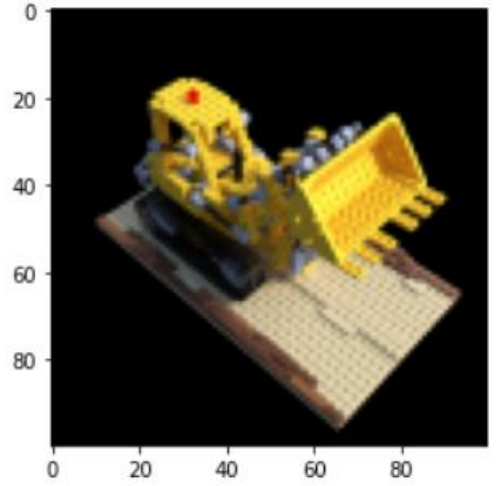
Load Input Images and Poses

```
data = np.load('tiny_nerf_data.npz')
images = data['images']
print(images.shape) # 106개의 사진, 100*100, RGB
poses = data['poses']
print(poses.shape) # 106개의 사진, ??
focal = data['focal']
print(focal) # ??
H, W = images.shape[1:3] # 100*100

testimg = images[101] # 102번째 사진으로 test
images = images[:100,...,:3] # 100개의 사진 선정, 100*100, RGB
poses = poses[:100] # 100개의 사진 선정

plt.imshow(testimg)
plt.show()
```

```
(106, 100, 100, 3)
(106, 4, 4)
138.88887889922103
```



- 106개의 각도에서 찍은 사진, 100*100 픽셀
- pose와 focal은 무엇을 의미하는지 모르겠습니다..
- 100개의 이미지만 선정, train set과 test set 분리(하지만 여기에서는 test 진행하지 않음)

2 View Synthesis

```
import time
t = time.time()

N_samples = 64
N_iters = 1000
i_plot = 25
psnrs = []
iternums = []

model = init_model()
optimizer = tf.keras.optimizers.Adam(5e-4)
```

- 학습 시간 측정하기 위해 time 사용

- 샘플링은 64개만 진행
(hierarchical sampling 없음)

- 1000번 반복, 25번마다 logging

- PSNR 평가 지표 사용, 높을수록 좋음

2 View Synthesis

```
for i in range(N_iters+1):
  # 랜덤으로 하나의 학습에 사용할 이미지 가져오기
  img_rand = np.random.randint(images.shape[0])
  target = images[img_rand]
  pose = poses[img_rand]

  # 학습
  with tf.GradientTape() as tape:
    # 선택한 각도에서, 각 픽셀에 대한 광선 구하기(r(t) = o + td)
    rays_o, rays_d = get_rays(H, W, focal, pose)
    # 선택한 이미지에서의 각도에 대한 렌더링(hierarchical sampling은 구현하지 않음)
    rgb, depth, acc = render_rays(model, rays_o, rays_d, near=2., far=6., N_samples=N_samples, rand=True)
    # 렌더링된 이미지와 실제 이미지와의 오차 계산
    loss = tf.reduce_mean(tf.square(rgb - target))
    # 계산한 오차를 가지고 학습
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

-랜덤으로 100개의 이미지 중 하나를 가져와 학습에 사용
-선택한 이미지의 각도에 맞춰 렌더링 후 실제 이미지와 색상 오차 계산하여 학습

2 View Synthesis

```
    # 학습 횟수가 25의 배수면 logging
    if i%i_plot==0:

        # 학습당 평균 시간 측정
        print('Epoch:', '%d'%i, ', secs per iter:', '%0.9f'%((time.time() - t) / i_plot))
        t = time.time()

        # 현재까지의 학습 결과를 가지고 렌더링
        rays_o, rays_d = get_rays(H, W, focal, testpose)
        rgb, depth, acc = render_rays(model, rays_o, rays_d, near=2., far=6., N_samples=N_samples)
        loss = tf.reduce_mean(tf.square(rgb - testimg))

        #그래프 출력
        plt.figure(figsize=(10,4)) #가로 10인치, 세로 4인치

        #그래프 출력 - 현재까지 학습 결과로 렌더링한 이미지
        plt.subplot(121) #rows = 1, cols = 2의 index = 1
        plt.imshow(rgb)
        plt.title(f'Iteration: {i}')

        #그래프 출력 - epoch당 PSNR(평가 지표, 높을수록 좋음)
        plt.subplot(122) #rows = 1, cols = 2의 index = 2
        iternums.append(i)
        psnr = -10. * tf.math.log(loss) / tf.math.log(10.)
        psnrs.append(psnr.numpy())
        plt.plot(iternums, psnrs)
        plt.title('PSNR')

        plt.show()

print('Done')
```

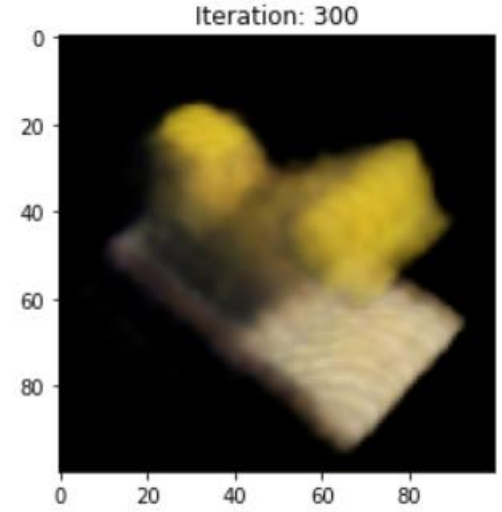
1. 학습당 평균 시간

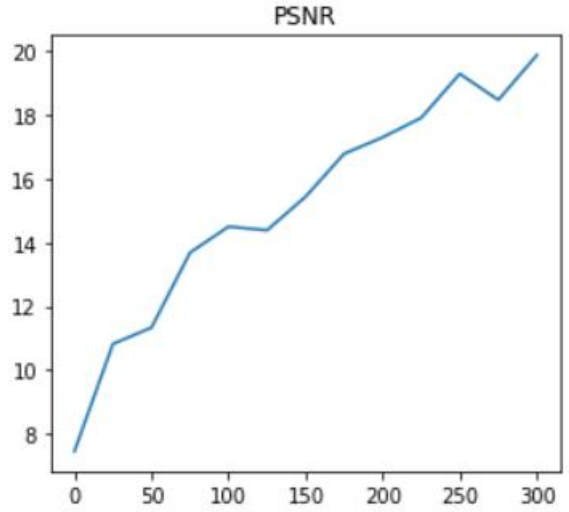
2. 현재까지의 모델로 렌더링한 결과

3. PSNR

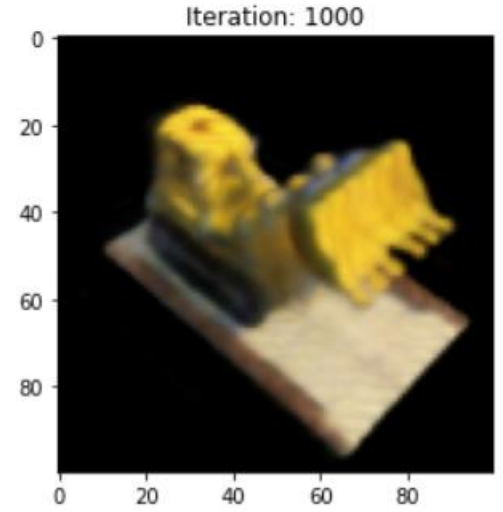
2 View Synthesis

Epoch: 300 , secs per iter: 0.713823843

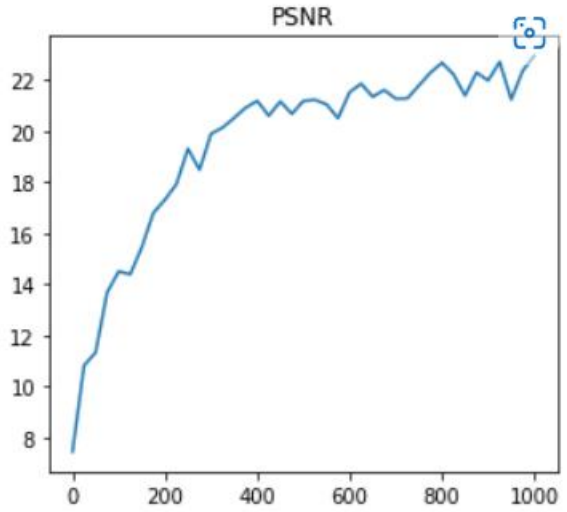




Epoch: 1000 , secs per iter: 0.717308865



Done



training이 진행될 수록 더욱 선명한 렌더링 결과를 얻을 수 있음

2 View Synthesis

```
for i in range(N_iters+1):
  # 랜덤으로 하나의 학습에 사용할 이미지 가져오기
  img_rand = np.random.randint(images.shape[0])
  target = images[img_rand]
  pose = poses[img_rand]

  # 학습
  with tf.GradientTape() as tape:
    # 선택한 각도에서, 각 픽셀에 대한 광선 구하기(r(t) = o + td)
    rays_o, rays_d = get_rays(H, W, focal, pose)
    # 선택한 이미지에서의 각도에 대한 렌더링(hierarchical sampling은 구현하지 않음)
    rgb, depth, acc = render_rays(model, rays_o, rays_d, near=2., far=6., N_samples=N_samples, rand=True)
    # 렌더링된 이미지와 실제 이미지와의 오차 계산
    loss = tf.reduce_mean(tf.square(rgb - target))
    # 계산한 오차를 가지고 학습
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

1. get_rays로 각 픽셀에 대한 광선 구하기

2. 각 광선에 대해 64개씩 샘플링하기(100 * 100 * 64)

3. 샘플링한 점들에 대해 positional encoding

4. MLP를 통해 xyz -> RGBa

5. 색상과 밀도를 가지고 볼륨 렌더링

6. 렌더링된 이미지와 실제 이미지를 비교하여 학습

2 View Synthesis

```
# 특정 각도에서, 각 픽셀에 대한 광선 구하기(r(t) = o + td)
# 함수 내용은 모르겠습니다.. 각 픽셀마다의 r(t) 계산 위한 o, d를 output으로 함
# output shape: 100*100(픽셀) * 2(o, d) * 3(xyz)

def get_rays(H, W, focal, c2w):
    i, j = tf.meshgrid(tf.range(W, dtype=tf.float32), tf.range(H, dtype=tf.float32), indexing='xy')
    dirs = tf.stack([(i-W*.5)/focal, -(j-H*.5)/focal, -tf.ones_like(i)], -1)
    rays_d = tf.reduce_sum(dirs[..., np.newaxis, :] * c2w[:3,:3], -1)
    rays_o = tf.broadcast_to(c2w[:3,-1], tf.shape(rays_d))
    return rays_o, rays_d
```

1. get_rays로 각 픽셀에 대한 광선 구하기

2. 각 광선에 대해 64개씩 샘플링하기(100 * 100 * 64)

3. 샘플링한 점들에 대해 positional encoding

4. MLP를 통해 xyz -> RGBa

5. 색상과 밀도를 가지고 볼륨 렌더링

6. 렌더링된 이미지와 실제 이미지를 비교하여 학습

2 View Synthesis

```
[ ]  # 렌더링
     def render_rays(network_fn, rays_o, rays_d, near, far, N_samples, rand=False):

       # batch normalization??
       def batchify(fn, chunk=1024*32):
         return lambda inputs : tf.concat([fn(inputs[i:i+chunk]) for i in range(0, inputs.shape[0], chunk)], 0)

       # 각 t(z_vals)에 대한 r(t) 구하기 => uniform sampling
       z_vals = tf.linspace(near, far, N_samples)
       if rand:
         z_vals += tf.random.uniform(list(rays_o.shape[:-1]) + [N_samples]) * (far-near)/N_samples
       pts = rays_o[...,None,:] + rays_d[...,None,:] * z_vals[...,:,None]
```

1. get_rays로 각 픽셀에 대한 광선 구하기

2. 각 광선에 대해 64개씩 샘플링하기(100 * 100 * 64)

3. 샘플링한 점들에 대해 positional encoding

4. MLP를 통해 xyz -> RGBa

5. 색상과 밀도를 가지고 볼륨 렌더링

6. 렌더링된 이미지와 실제 이미지를 비교하여 학습

2 View Synthesis

```
# MLP를 통해 각 샘플에 대한 RGBa 구하기
# (100, 100, 64, 3)
pts_flat = tf.reshape(pts, [-1,3]) #100 * 100 * 64(sample) -> 640000
# (640000, 3)
pts_flat = embed_fn(pts_flat) # positional encoding 적용
# (640000, 39)
raw = batchify(network_fn)(pts_flat) # model 적용(xyz * (2 * L_embed + 1) -> RGBa)
# (640000, 4)
raw = tf.reshape(raw, list(pts.shape[:-1]) + [4]) #640000 -> 100 * 100 * 64(sample)
# (100, 100, 64, 4)

# RGB(색상)와 a(밀도) 분리
rgb = tf.math.sigmoid(raw[...,:3])
sigma_a = tf.nn.relu(raw[...,3])
```

```
[ ]  # positional encoding
     # tf.Tensor를 input으로 하고, 2^i *pi * input만큼 sin, cos해서 ret에 추가
     # shape: (data, 3) => (data, 3 * (2 * L_embed + 1))

     def posenc(x):
      rets = [x]
      for i in range(L_embed):
        for fn in [tf.sin, tf.cos]:
          rets.append(fn(2.**i * x))
      return tf.concat(rets, -1)

     #parameters
     L_embed = 6
     embed_fn = posenc

     # parameters_no positional encoding
     # L_embed = 0
     # embed_fn = tf.identity
```

1. get_rays로 각 픽셀에 대한 광선 구하기

2. 각 광선에 대해 64개씩 샘플링하기(100 * 100 * 64)

3. 샘플링한 점들에 대해 positional encoding

4. MLP를 통해 xyz -> RGBa

5. 색상과 밀도를 가지고 볼륨 렌더링

6. 렌더링된 이미지와 실제 이미지를 비교하여 학습

2 View Synthesis

```
# MLP를 통해 각 샘플에 대한 RGBa 구하기
# (100, 100, 64, 3)
pts_flat = tf.reshape(pts, [-1,3]) #100 * 100 * 64(sample) -> 640000
# (640000, 3)
pts_flat = embed_fn(pts_flat) # positional encoding 적용
# (640000, 39)
raw = batchify(network_fn)(pts_flat) # model 적용(xyz * (2 * L_embed + 1) -> RGBa)
# (640000, 4)
raw = tf.reshape(raw, list(pts.shape[:-1]) + [4]) #640000 -> 100 * 100 * 64(sample)
# (100, 100, 64, 4)

# RGB(색상)와 a(밀도) 분리
rgb = tf.math.sigmoid(raw[...,:3])
sigma_a = tf.nn.relu(raw[...,3])
```

```
# 모델 정의(논문에서의 설명보다는 간소화된 상태)
# MLP로 구성, xyz -> RGBa 변환
# D = depth(몇 층으로 구성되어 있는가), W = weight(중간 층에서의 dimension)

def init_model(D=8, W=256):
    relu = tf.keras.layers.ReLU()
    dense = lambda W=W, act=relu : tf.keras.layers.Dense(W, activation=act)

    inputs = tf.keras.Input(shape=(3 + 3 * 2 * L_embed))  # 3(xyz)) + positional encoding
    outputs = inputs
    for i in range(D):
        outputs = dense()(outputs)
        if i%4==0 and i>0:
            outputs = tf.concat([outputs, inputs], -1) #네트워크 중간에 input을 다시 넣어줌
    outputs = dense(4, act=None)(outputs) #최종 output은 4차원

    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model
```

1. get_rays로 각 픽셀에 대한 광선 구하기

2. 각 광선에 대해 64개씩 샘플링하기(100 * 100 * 64)

3. 샘플링한 점들에 대해 positional encoding

4. MLP를 통해 xyz -> RGBa

5. 색상과 밀도를 가지고 볼륨 렌더링

6. 렌더링된 이미지와 실제 이미지를 비교하여 학습

2 View Synthesis

```
# 볼륨 렌더링
dists = tf.concat([z_vals[..., 1:] - z_vals[..., :-1], tf.broadcast_to([1e10], z_vals[...,:1].shape)], -1)
alpha = 1.-tf.exp(-sigma_a * dists)
weights = alpha * tf.math.cumprod(1.-alpha + 1e-10, -1, exclusive=True)

rgb_map = tf.reduce_sum(weights[...,None] * rgb, -2)
depth_map = tf.reduce_sum(weights * z_vals, -1)
acc_map = tf.reduce_sum(weights, -1)

return rgb_map, depth_map, acc_map
```

1. get_rays로 각 픽셀에 대한 광선 구하기

2. 각 광선에 대해 64개씩 샘플링하기(100 * 100 * 64)

3. 샘플링한 점들에 대해 positional encoding

4. MLP를 통해 xyz -> RGBa

5. 색상과 밀도를 가지고 볼륨 렌더링

6. 렌더링된 이미지와 실제 이미지를 비교하여 학습

3 360 video rendering

```
[ ]  frames = []
     for th in tqdm(np.linspace(0., 360., 120, endpoint=False)):
       c2w = pose_spherical(th, -30., 4.)
       rays_o, rays_d = get_rays(H, W, focal, c2w[:3,:4])
       rgb, depth, acc = render_rays(model, rays_o, rays_d, near=2., far=6., N_samples=N_samples)
       frames.append((255*np.clip(rgb,0,1)).astype(np.uint8))

     import imageio
     f = 'video.mp4'
     imageio.mimwrite(f, frames, fps=30, quality=7)
```

```
[ ]  from IPython.display import HTML
     from base64 import b64encode
     mp4 = open('video.mp4','rb').read()
     data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
     HTML("""
     <video width=400 controls autoplay loop>
         <source src="%s" type="video/mp4">
     </video>
     """ % data_url)
```