yagna_kaasaragadda_deep_learning_assignment_4_problem2

October 17, 2023

[ ]:
```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot  as plt
import torch.nn.functional as F
from torchvision.datasets import DatasetFolder
import cv2

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

[ ]:
```
!git clone https://github.com/skokalj/RowanDLclassNEA.git
```

```
fatal: destination path 'RowanDLclassNEA' already exists and is not an empty
directory.
```

[ ]:
```
from torchvision import transforms
import PIL.Image


def load_image(img_path:str):
        np_img = cv2.imread(img_path)
        return PIL.Image.fromarray(np_img)

dset = DatasetFolder(root=r'RowanDLclassNEA/NEUdata', loader = load_image,␣
 ↪extensions = ('.bmp',))

transform = transforms.Compose([
  transforms.Resize(256),
  transforms.CenterCrop(224),
  transforms.ToTensor(),
  transforms.Normalize(
      mean=[0.485, 0.456, 0.406],
      std=[0.229, 0.224, 0.225])])

dset = DatasetFolder(root=r'RowanDLclassNEA/NEUdata', loader = load_image,␣
 ↪extensions = ('.bmp',), transform = transform)
```

[ ]:
```
from torch.utils.data import random_split

train_set, val_set = random_split(
                      dset,
                      [1200, 600])

trainloader = torch.utils.data.DataLoader(
                    train_set,
                    batch_size=16,
                    shuffle=True)

valloader = torch.utils.data.DataLoader(
                    val_set,
                    batch_size=16,
                    shuffle=True)
```

[ ]:
```
data, label = next(iter(trainloader))

data.shape
```

[ ]:
```
torch.Size([16, 3, 224, 224])
```

[ ]:
```
import torch
import torch.nn as nn

class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(8, 16, 3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(16, 32, 3, stride=2, padding=1)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = torch.relu(self.conv3(x))
        return x

class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.deconv1 = nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1)
        self.deconv2 = nn.ConvTranspose2d(16, 8, 3, stride=2, padding=1)
        self.deconv3 = nn.ConvTranspose2d(8, 3, 3, stride=2, padding=1)

    def forward(self, x):
        x = torch.relu(self.deconv1(x))
```

```
        x = torch.relu(self.deconv2(x))
        x = torch.sigmoid(self.deconv3(x))
        return x

class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

[ ]:
```
import torch
import torch.optim as optim
import torch.nn as nn

device = "cuda" if torch.cuda.is_available() else "cpu"
autoencoder = Autoencoder().to(device=device)
criterion = nn.MSELoss()
optimizer = optim.SGD(autoencoder.parameters(), lr=0.01, momentum=0.9)

N_EPOCHS = 50
tr_loss_hist = []
val_loss_hist = []

for epoch in range(N_EPOCHS):

    train_loss = 0.0
    autoencoder.train()

    for inputs, _ in trainloader:
        inputs = inputs.to(device)

        optimizer.zero_grad()

        outputs = autoencoder(inputs)
        resize_transform = transforms.Resize((224, 224))
        outputs = resize_transform(outputs)
        loss = criterion(outputs, inputs)

        loss.backward()
        optimizer.step()

        train_loss += loss.item()
```

```
    val_loss = 0.0
    autoencoder.eval()

    for inputs, _ in valloader:
        inputs = inputs.to(device)

        outputs = autoencoder(inputs)
        resize_transform = transforms.Resize((224, 224))
        outputs = resize_transform(outputs)
        loss = criterion(outputs, inputs)

        val_loss += loss.item()

    print("Epoch: {} Train Loss: {} Val Loss: {}".format(
        epoch, train_loss / len(trainloader), val_loss / len(valloader)))
    tr_loss_hist.append(train_loss / len(trainloader))
    val_loss_hist.append(val_loss / len(valloader))
```

```
c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-
packages\torchvision\transforms\functional.py:1603: UserWarning: The default
value of the antialias parameter of all the resizing transforms (Resize(),
RandomResizedCrop(), etc.) will change from None to True in v0.17, in order to
be consistent across the PIL and Tensor backends. To suppress this warning,
directly pass antialias=True (recommended, future default), antialias=None
(current default, which means False for Tensors and True for PIL), or
antialias=False (only works on Tensors - PIL will still use antialiasing). This
also applies if you are using the inference transforms from the models weights:
update the call to weights.transforms(antialias=True).
  warnings.warn(
```

```
Epoch: 0 Train Loss: 0.9448308054606119 Val Loss: 0.9056618637160251
Epoch: 1 Train Loss: 0.9136919665336609 Val Loss: 0.8787455190169183
Epoch: 2 Train Loss: 0.8980738095442454 Val Loss: 0.863718024209926
Epoch: 3 Train Loss: 0.8902680500348409 Val Loss: 0.859459292731787
Epoch: 4 Train Loss: 0.8857079108556112 Val Loss: 0.8540434586374384
Epoch: 5 Train Loss: 0.8829828910032909 Val Loss: 0.8567193330902803
Epoch: 6 Train Loss: 0.8812591584523519 Val Loss: 0.8592999789275622
Epoch: 7 Train Loss: 0.8800425227483114 Val Loss: 0.846468365506122
Epoch: 8 Train Loss: 0.8791759626070659 Val Loss: 0.8502309604694969
Epoch: 9 Train Loss: 0.8786619873841603 Val Loss: 0.8559394086662092
Epoch: 10 Train Loss: 0.878176064491272 Val Loss: 0.8529743097330394
Epoch: 11 Train Loss: 0.8778266421953838 Val Loss: 0.8530622846201846
Epoch: 12 Train Loss: 0.877572299639384 Val Loss: 0.8474722386975038
Epoch: 13 Train Loss: 0.8773988183339437 Val Loss: 0.8448863923549652
Epoch: 14 Train Loss: 0.8772234718004862 Val Loss: 0.8491889301099276
Epoch: 15 Train Loss: 0.8770340156555175 Val Loss: 0.8457779131437603
Epoch: 16 Train Loss: 0.8769327433904012 Val Loss: 0.8476189798430392
```

```
Epoch: 17 Train Loss: 0.8768842800458272 Val Loss: 0.8498212326514093
Epoch: 18 Train Loss: 0.8767496713002523 Val Loss: 0.8425291878612418
Epoch: 19 Train Loss: 0.8766906726360321 Val Loss: 0.8450826320209002
Epoch: 20 Train Loss: 0.87659938454628 Val Loss: 0.8511998143635298
Epoch: 21 Train Loss: 0.8765528543790182 Val Loss: 0.8456992216800389
Epoch: 22 Train Loss: 0.8764846885204315 Val Loss: 0.8485586588319979
Epoch: 23 Train Loss: 0.8764226766427358 Val Loss: 0.8466203126468157
Epoch: 24 Train Loss: 0.8763035563627879 Val Loss: 0.8516024162894801
Epoch: 25 Train Loss: 0.8762430834770203 Val Loss: 0.8461345214592783
Epoch: 26 Train Loss: 0.8761699418226878 Val Loss: 0.8462095135136655
Epoch: 27 Train Loss: 0.8759487164020539 Val Loss: 0.8474605499129546
Epoch: 28 Train Loss: 0.8756444303194681 Val Loss: 0.8401525648016679
Epoch: 29 Train Loss: 0.8750477313995362 Val Loss: 0.8396681264827126
Epoch: 30 Train Loss: 0.8737404239177704 Val Loss: 0.8422199699439501
Epoch: 31 Train Loss: 0.8704095602035522 Val Loss: 0.8353573303473624
Epoch: 32 Train Loss: 0.8458011170228322 Val Loss: 0.7374401531721416
Epoch: 33 Train Loss: 0.5706716910998026 Val Loss: 0.49534866213798523
Epoch: 34 Train Loss: 0.47591678440570834 Val Loss: 0.4480230918056087
Epoch: 35 Train Loss: 0.44512454489866893 Val Loss: 0.4269769066258481
Epoch: 36 Train Loss: 0.42811627984046935 Val Loss: 0.4115311915152951
Epoch: 37 Train Loss: 0.4176080362002055 Val Loss: 0.40547987034446314
Epoch: 38 Train Loss: 0.4107160985469818 Val Loss: 0.4026633830446946
Epoch: 39 Train Loss: 0.40584597071011863 Val Loss: 0.3943340970497382
Epoch: 40 Train Loss: 0.4022965604066849 Val Loss: 0.39468685145440857
Epoch: 41 Train Loss: 0.399549320936203 Val Loss: 0.3918251457967256
Epoch: 42 Train Loss: 0.39738854865233103 Val Loss: 0.3896137828889646
Epoch: 43 Train Loss: 0.39564051608244577 Val Loss: 0.38583083686075714
Epoch: 44 Train Loss: 0.3941827529668808 Val Loss: 0.3829561040589684
Epoch: 45 Train Loss: 0.3929565227031708 Val Loss: 0.3834725002709188
Epoch: 46 Train Loss: 0.39191037992636363 Val Loss: 0.38233362648047897
Epoch: 47 Train Loss: 0.3910048166910807 Val Loss: 0.38364142728479284
Epoch: 48 Train Loss: 0.3902007194360097 Val Loss: 0.38167898984331833
Epoch: 49 Train Loss: 0.38950171768665315 Val Loss: 0.3794572655307619
```

[ ]:
```
import matplotlib.pyplot as plt

autoencoder.eval()
originals = []
reconstructed = []

tset = DatasetFolder(root=r'RowanDLclassNEA/NEUdata_split/Test',␣
 ↪loader=load_image, extensions=('.bmp',), transform=transform)

test_loader = torch.utils.data.DataLoader(
    tset,
    batch_size=50,
    shuffle=True)
```

```
for inputs, _ in test_loader:
    inputs = inputs.to(device)
    outputs = autoencoder(inputs)

    inputs_np = inputs.cpu().detach().numpy()
    outputs_np = outputs.cpu().detach().numpy()

    originals.extend(inputs_np)
    reconstructed.extend(outputs_np)

originals = np.array(originals)
reconstructed = np.array(reconstructed)

#print(reconstructed[0])

num_images = len(originals[:50])


fig = plt.figure(figsize=(4,100))

print(num_images)

for i in range(50):
    fig.add_subplot(50,2,2*i+1)
    #print(originals[0].shape)
    plt.imshow(originals[i].transpose((1, 2, 0)))
    fig.add_subplot(50,2,2*i+2)
    #print(reconstructed[0].shape)
    plt.imshow(reconstructed[i].transpose((1, 2, 0)))


plt.show()
```

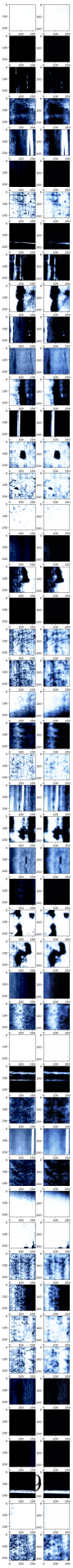
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
50
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



[ ]:
```
torch.save(autoencoder.state_dict(),"./AE_model.pt")
```

[ ]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
autoencoder = Autoencoder().to(device=device)
autoencoder_s = autoencoder.load_state_dict(torch.load("./AE_model.
 ↪pt",map_location=torch.device('cpu')))
```

[ ]:
```
import matplotlib.pyplot as plt

autoencoder_s.eval()
originals = []
reconstructed = []

tset = DatasetFolder(root=r'RowanDLclassNEA/NEUdata_split/Test',␣
 ↪loader=load_image, extensions=('.bmp',), transform=transform)

test_loader = torch.utils.data.DataLoader(
    tset,
    batch_size=50,
    shuffle=True)

for inputs, _ in test_loader:
    inputs = inputs.to(device)
    outputs = autoencoder_s(inputs)

    inputs_np = inputs.cpu().detach().numpy()
    outputs_np = outputs.cpu().detach().numpy()

    originals.extend(inputs_np)
    reconstructed.extend(outputs_np)

originals = np.array(originals)
reconstructed = np.array(reconstructed)

#print(reconstructed[0])

num_images = len(originals[:50])


fig = plt.figure(figsize=(4,100))

print(num_images)

for i in range(1):
```

```
    fig.add_subplot(50,2,2*i+1)
    print(originals[0].shape)
    plt.imshow(originals[i].transpose((1, 2, 0)))
    fig.add_subplot(50,2,2*i+2)
    print(reconstructed[0].shape)
    plt.imshow(reconstructed[i].transpose((1, 2, 0)))


plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
c:\Users\kaasa\Documents\GitHub\DeepLearning_class\Assignment 4\Problem2.ipynb␣
 ↪Cell 11 line 3
      <a href='vscode-notebook-cell:/c%3A/Users/kaasa/Documents/GitHub/
 ↪DeepLearning_class/Assignment%204/Problem2.ipynb#X13sZmlsZQ%3D%3D?line=0'>1</
 ↪a> import matplotlib.pyplot as plt
----> <a href='vscode-notebook-cell:/c%3A/Users/kaasa/Documents/GitHub/
 ↪DeepLearning_class/Assignment%204/Problem2.ipynb#X13sZmlsZQ%3D%3D?line=2'>3</
 ↪a> autoencoder_s.eval()
      <a href='vscode-notebook-cell:/c%3A/Users/kaasa/Documents/GitHub/
 ↪DeepLearning_class/Assignment%204/Problem2.ipynb#X13sZmlsZQ%3D%3D?line=3'>4</
 ↪a> originals = []
      <a href='vscode-notebook-cell:/c%3A/Users/kaasa/Documents/GitHub/
 ↪DeepLearning_class/Assignment%204/Problem2.ipynb#X13sZmlsZQ%3D%3D?line=4'>5</
 ↪a> reconstructed = []

AttributeError: '_IncompatibleKeys' object has no attribute 'eval'
```

[ ]: