گزارش کار پیاده سازی بخش سوم

تمرین شماره چهارم

کتایون کبرائی

در این تمرین با دیتاسیت سیفار10 کار میشد و با استفاده از ان اوتواینکودری باید پیاده سازی میشد که بتواند عکس های ترکیبی بدست امده را از هم جدا کرده و تمایز بخشد و نهایتا به عنوان خروجی هایی به ما برگرداند. نکته این تمرین با تمرین های دیگر این بود که در این تمرین علاوه بر دقت و خطای بدست امده خروجی مورد نظر ما عکس های بازتولید شده است که که به درستی تمییز داده شود.

ابتدا دیتا را لود میکنیم. میدانیم در لایبرری تورچ ویژن این دیتاست موجود است و ما به اسانی به ان دسترسی داریم. نکته استفاده از این روش لود دیتا این است که میتوانیم بچ سایز به ان داده و از انجایی که در صورت سوال هم گفته کار کردن با 1000 داده این دیتاست موردنیاز است، پس بچ سایز را محدود انتخاب میکنیم و مد شافل را انتخاب میکنیم تا به صورت رندوم دیتای محدود شده ما لود شود.

دیتای مورد نیاز ما مثل هر دیتای دیگر ابتدا نیاز است نرمالایز شود. با استفاده از لایبرری ترنسفرم تمامی دیتا را نرمالایز میکنیم و همزمان به تنسور هم تبدیل شان میکنیم.

بعد از اضافه کردن دیتا ان را به دیتا لودر داده تا بخش محدود دیتا در ان نگه داری شود. توابع ترنسفرم نیز ورودی به این شیی داده می شود.

همانند دیتای ترین دیتای تست نیز هم اضافه و لود میشود.

[3]:

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.485, 0.456, 0.406), (0.485, 0.456, 0.406))])

train_batch_size = 2000

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=train_batch_size,
                                          shuffle=True, num_workers=2)
test_batch_size = 1000
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=test_batch_size,
                                         shuffle=False, num_workers=2)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
Loading widget...
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

حال که دیتای موردنظر به درستی لود شد نیاز داریم دیتا مورد نظر مسئله را تولید کنیم. در مسئله گفته دیتای ورودی باید نتیجه میانگین دو عکس دیتاست اصلی باشند. برای این کار روش های مختلفی را میتوان امتحان کرد تا تعداد دیتای کافی بدست اید. برای مثال میتوان دو دسته 2000 تایی از دیتای اصلی انتخاب کرد و برای مثال نظیر به نظیر ان ها را باهم جمع و تقسیم بر دو کرد. یا کاری که در این پیاده سازی انجام شد این بود که دو دسته کوچک تر 50 تایی مثلا انتخاب شد و به شکلی ایتریشن انجام شد که نتیجه میانگین هر عکس با تمامی 50 عضو دسته دیگر ذخیره شد.

```
for i in range(50):
    tar1 = train_images[i:i+1]
    tar1 = tar1.cuda()
    for j in range(600,650):
        tar2 = train_images[j:j+1]
        inp = torch.FloatTensor([((tar1.cpu()[0].numpy()+tar2.cpu()[0].numpy())/2).tolist()])
        inp = inp.cuda()
```

چون با جی پی یو کار میشد نیاز بود تمام دیتای ورودی مدل در کودا هم اضافه شوند.

برای دیدن مثال های بصری از این مجموعه تولید شده نیاز بود ابتدا هر ورودی از 32*32*3 به 3*32*32 تغییر شکل یابد تا بتوان ان را به تابع رسم شکل داد.

```
for i in range(len(avg_input)):
    avg_input[i] = np.transpose(avg_input[i],[2, 1, 0])

for j in range(len(labels1)):
    labels1[j] = np.transpose(labels1[j],[2, 1, 0])
    labels2[j] = np.transpose(labels2[j],[2, 1, 0])

print(len(avg_input))
type(avg_input)
```

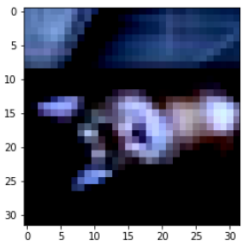
```
45150
[10]: list
```

در ادامه برخی از نمونه های تولید شده از میانگین گیری دو عکس دیتا ست اصلی را میبینیم.

عکس های اصلی:


```
[12]: plt.imshow(labels2[10])
      plt.show()
```

onsole

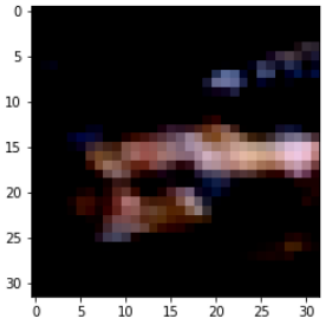



```
[12]: plt.imshow(labels2[10])
      plt.show()
```

onsole

عکس میانگین تولید شده:

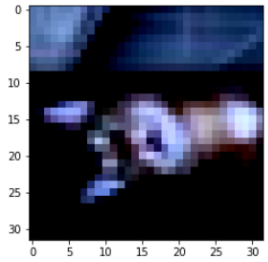
```
[13]: plt.imshow(avg_input[10])
      plt.show()
```



onsole

عکس های اصلی :

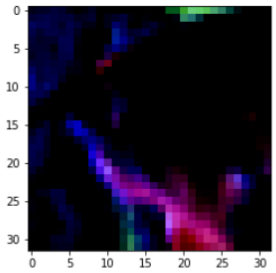
[12]:
```
plt.imshow(labels2[10])
plt.show()
```



[14]:

onsole

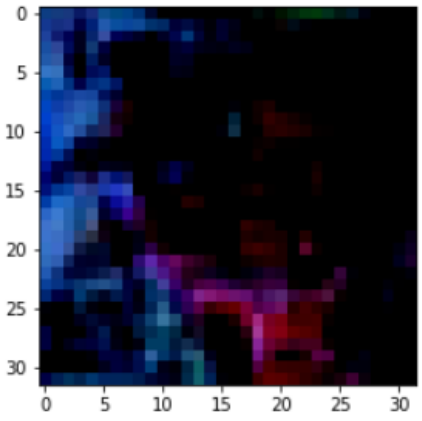
[13]:
```
plt.imshow(labels1[10])
plt.show()
```



onsole

عکس میانگین تولید شده :

[14]:
```
plt.imshow(avg_input[10])
plt.show()
```



[15]:

onsole

حال به بخش طراحی مدل میرسیم. کد ما شئی است که که سه پارامتر اصلی مسئله یعنی اینکودر و دو دیکودر موازی را در خود دارد. هر کدام از این ها را مثل مدل های قبلی طراحی میکنیم. یعنی به انتخاب یک سری لایه ی لینیر یا کانولوشن میذاریم و بعد از ان اکتیویشن فانکشن مورد نظر ان را قرار میدهیم. نکته مهم این است که شیئ دو دوکودر موازی با هم دارد که هردو ورودی خروجی دیکودر را میگیرند و ان را دوباره کدگزاری میکنند تا عکس مورد نظر را رمزگشایی کنند و دوباره بسازند. برحسب تجربه میدانیم اگر به جای لایه لینیر، لایه کانولوشن بگذاریم نتیجه بهتری خواهیم گرفت. ابتدا اینکودر ورودی عکس ما را میگیرد و انرا با توجه به گام دوتایی به لایه بعدی که دو برابر ان است میدهد. میتوانستیم تا جایی که بخواهیم کار را ادامه دهیم ولی بر حسب تجربه تعداد زیاد لایه هم در اینکودر نتیجه خوبی نمیدهد. سپس این لایه های بسیط را دوباره به دیکودر میدهیم تا ان را منبسط کرده (در اینجا هر بار نصف در هر لایه میشود) و به عکس برگرداند که نهاتا هر کدام از دیکودر ها یک خروجی و عکس را به ما برمیگردانند.

```
class Autoencoder(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.encoder = torch.nn.Sequential(
            nn.Conv2d(3, 12, 4, stride=2, padding=1),
            nn.ReLU(),
#             nn.MaxPool2d((2, 2)),
            nn.Conv2d(12, 24, 4, stride=2, padding=1),
            nn.ReLU(),
#             nn.MaxPool2d((2, 2)),
            nn.Conv2d(24, 48, 4, stride=2, padding=1),
            nn.ReLU(),
#             nn.MaxPool2d((2, 2)),
            nn.Conv2d(48, 96, 4, stride=2, padding=1),
            nn.ReLU(),
        )
```

```
        self.decoder1 = torch.nn.Sequential(
            nn.ConvTranspose2d(96, 48, 4, stride=2, padding=1),
#             nn.Conv2d(96, 48, 4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(48, 24, 4, stride=2, padding=1),
#             nn.Conv2d(48, 24, 4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(24, 12, 4, stride=2, padding=1),
#             nn.Conv2d(24, 12, 4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(12, 3, 4, stride=2, padding=1),
#             nn.Conv2d(12, 6, 4, stride=2, padding=1),
            nn.Sigmoid(),
        )

        self.decoder2 = torch.nn.Sequential(
            nn.ConvTranspose2d(96, 48, 4, stride=2, padding=1),
#             nn.Conv2d(96, 48, 4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(48, 24, 4, stride=2, padding=1),
#             nn.Conv2d(48, 24, 4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(24, 12, 4, stride=2, padding=1),
#             nn.Conv2d(24, 12, 4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(12, 3, 4, stride=2, padding=1),
#             nn.Conv2d(12, 6, 4, stride=2, padding=1),
            nn.Sigmoid(),
        )
```

بعد از ان همانند ورودی ها مدل را هم به جی پی یو میفرستیم تا از ان استفاده کنیم.

برای مدل از اپتیمایزر آدام و تابع لاس ام اس ای استفاده میکنیم که بهترین نتیجه ممکن را به ما بازگرداند.

```
model = create_model()
print(model)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters())
```

در نهایت کار شروع به ترین مدل میکنیم.

```
epochs = 15
losses = []
for epoch in range(epochs):
    t0 = datetime.now()
    running_loss = 0.0
    print(f"Running epoch {epoch+1} out of {epochs}")

    for i in range(50):
        tar1 = train_images[i:i+1]
        tar1 = tar1.cuda()
        for j in range(600,650):
            tar2 = train_images[j:j+1]
            inp = torch.FloatTensor([((tar1.cpu()[0].numpy()+tar2.cpu()[0].numpy())/2).tolist()])
            inp = inp.cuda()
            tar2 = tar2.cuda()


            out1, out2 = model(inp)
            loss_1 = criterion(out1, tar1)
            loss_2 = criterion(out2, tar2)
            loss = loss_1 + loss_2
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.data
    losses.append(running_loss/40000)

    dt = datetime.now() - t0
    print(f'Train_Loss: {losses[-1]:.4f}, Duration: {dt} ')
```

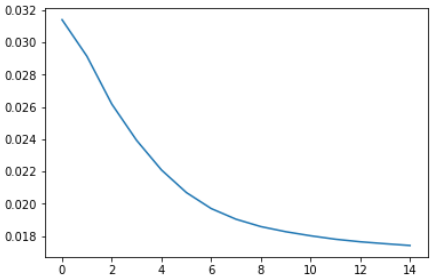
لاس به دست امده از این نمونه مدل پایین و مناسب خواهد شد:

```
Running epoch 1 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.522898
Running epoch 2 out of 15
Train_Loss: 0.0167, Duration: 0:00:14.064400
Running epoch 3 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.384632
Running epoch 4 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.554401
Running epoch 5 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.891970
Running epoch 6 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.576412
Running epoch 7 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.707251
Running epoch 8 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.231113
Running epoch 9 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.823837
Running epoch 10 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.707560
Running epoch 11 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.905666
Running epoch 12 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.455461
Running epoch 13 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.356130
Running epoch 14 out of 15
Train_Loss: 0.0167, Duration: 0:00:14.459154
Running epoch 15 out of 15
Train_Loss: 0.0167, Duration: 0:00:13.691871
```

اگر نمودار لاس مورد نظر را بکشیم میبینیم تا حد پایینی کم میشود که این نشان میدهد مدل ما درست کار کرده است و مدل توانسته با خطای کمی این دو عکس را از عکس میانگین بیرون بکشد.

```python
plot_losses = []
for l in losses:
    plot_losses.append(float(l.cpu()))

plt.plot(plot_losses)
plt.show()
```



حال برای تست مدل دیتای تست را به مدل میدهیم تا دو خروجی عکس مورد نظر را به ما برگرداند. برای اینکار مقدار دلخواه که ما تجربا 100 دیتا از دیتای تست برمیداریم و انرا به مدل میدهیم تا لاس های ان بدست اید. بعد از تشکیل دیتا و تولید ارایه لاس ها میتوانیم میانگین این ارایخ را برای بیان لاس کلی تست مدل بیان کنیم.

```
test_losses = []
total_loss = 0.0

for i in range(100):
    Tinp1 = test_images[i:i+1]
    Tinp1 = Tinp1.cuda()
    for j in range(100,200):
        Tinp2 = test_images[j:j+1]
        Tinp2 = Tinp2.cuda()
        test_inp = torch.FloatTensor([((Tinp1.cpu()[0].numpy()+Tinp2.cpu()[0].numpy())/2).tolist()])
        test_inp = test_inp.cuda()
        test_out1, test_out2 = model(test_inp)

        test_loss_1 = criterion(test_out1, Tinp1)
        test_loss_2 = criterion(test_out2, Tinp2)
        test_loss = test_loss_1 + test_loss_2
        running_loss += test_loss.data
test_losses.append(running_loss/10000)



inp1 = test_images[10:11]
inp2 = test_images[19:20]

test_input = torch.FloatTensor([((inp1.cpu()[0].numpy()+inp2.cpu()[0].numpy())/2).tolist()])
test_input = test_input.cuda()

test_output1, test_output2 = model(test_input)
```

```
print(test_loss)
```

```
tensor(0.5274, device='cuda:0', grad_fn=<AddBackward0>)
```

+ Code + Markdown

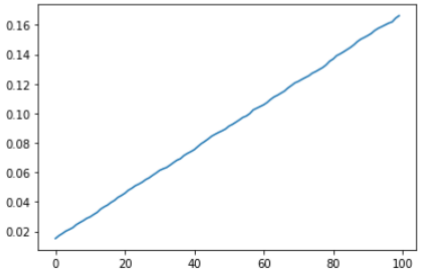
```
print(test_losses)
```

```
[tensor(0.0189, device='cuda:0')]
```

میتوانیم برای کمک بیشتر نمودار لاس هارا هم رسم کنیم.

```
20]:
plot_test_losses = []
for l in test_losses:
    plot_test_losses.append(float(l.cpu()))

plt.plot(plot_test_losses)
plt.show()
```

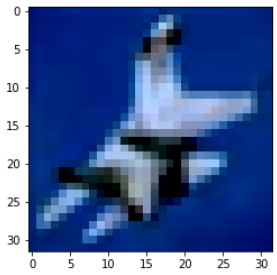


سپس این خروجی ها را به شکل بصری چاپ میکنیم تا چک کنیم و ببینیم مدل چگونه عمل میکند.

```
print("real image - input 1")

inp1 = np.transpose(inp1[0],[2, 1, 0])
plt.imshow(torchvision.utils.make_grid(inp1.data))
```

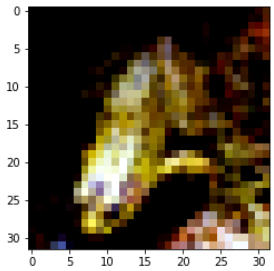
```
real image - input 1
<matplotlib.image.AxesImage at 0x7f7bfc359ad0>
```



+ Code + Markdown

```
print("real image - input 1")
print("it is a ", str(test_labels[10:11]))
inp2 = np.transpose(inp2[0],[2, 1, 0])
plt.imshow(torchvision.utils.make_grid(inp2.data))
```

```
real image - input 1
it is a  tensor([0])
<matplotlib.image.AxesImage at 0x7f7bfbcd53d0>
```

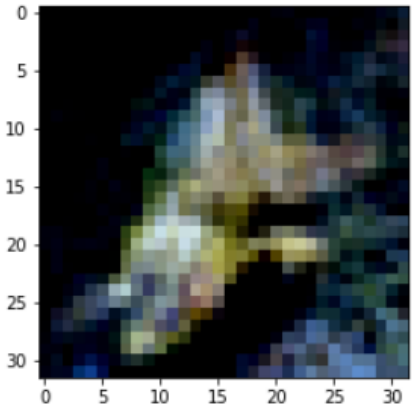


دیتای ترکیبی ساخته شده:

```
print("avarage image - input")

avg = torch.FloatTensor([((inp2.cpu().numpy()+inp1.cpu().numpy())/2).tolist()])
avg = np.array(avg)
plt.imshow(avg[0])
```

```
avarage image - input
<matplotlib.image.AxesImage at 0x7efc3efe7b50>
```

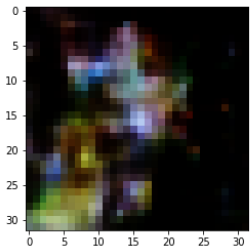


خروجی های مدل:

```
print("predected image - output 1")

type(test_output2[0])
index = test_output2[0].cpu()
index = index.detach().numpy()
# output1 = np.array(index)
type(index)
index.shape
test_output2 = np.transpose(index,[2, 1, 0])
plt.imshow(test_output2)
```

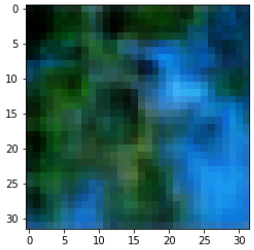
```
<matplotlib.image.AxesImage at 0x7f7bfc2c5390>
```



```
print("predected image - output 1")

index = test_output1[0].cpu()
index = index.detach().numpy()
test_output1 = np.transpose(index,[2, 1, 0])
plt.imshow(test_output1)
```

```
<matplotlib.image.AxesImage at 0x7f7bfc286c90>
```

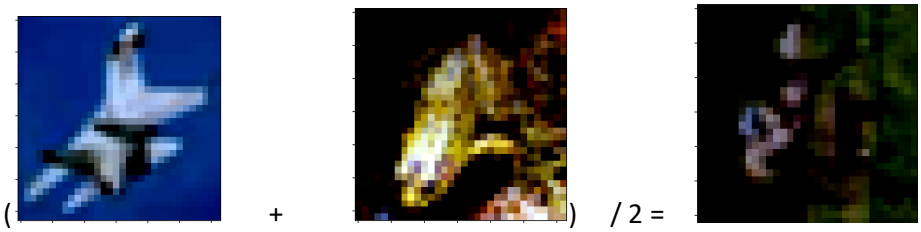


در این تمرین با ساختار های دیگر و تعداد داده های بیشتری هم کار شد.

برای مثال اگه دسته های داده را 100 تایی یا 200 تایی قرار میدادیم هم به نتایجی میرسیدیم که بهترین نتیجه نبود. برای مثال لاس ان 0.0673 میشد.

```
Running epoch 1 out of 20
Train_Loss: 0.0678, Duration: 0:00:54.968518
Running epoch 2 out of 20
Train_Loss: 0.0677, Duration: 0:00:55.110824
Running epoch 3 out of 20
Train_Loss: 0.0676, Duration: 0:00:54.897178
Running epoch 4 out of 20
Train_Loss: 0.0676, Duration: 0:00:55.145834
Running epoch 5 out of 20
Train_Loss: 0.0676, Duration: 0:00:55.057226
Running epoch 6 out of 20
Train_Loss: 0.0676, Duration: 0:00:55.244662
Running epoch 7 out of 20
Train_Loss: 0.0675, Duration: 0:00:54.640514
Running epoch 8 out of 20
Train_Loss: 0.0675, Duration: 0:00:55.070483
Running epoch 9 out of 20
Train_Loss: 0.0675, Duration: 0:00:55.224217
Running epoch 10 out of 20
Train_Loss: 0.0674, Duration: 0:00:55.243265
Running epoch 11 out of 20
Train_Loss: 0.0674, Duration: 0:00:54.732648
Running epoch 12 out of 20
Train_Loss: 0.0674, Duration: 0:00:55.171618
Running epoch 13 out of 20
Train_Loss: 0.0673, Duration: 0:00:55.543859
Running epoch 14 out of 20
Train_Loss: 0.0674, Duration: 0:00:55.367001
Running epoch 15 out of 20
Train_Loss: 0.0673, Duration: 0:00:54.837852
Running epoch 16 out of 20
Train_Loss: 0.0673, Duration: 0:00:55.792784
Running epoch 17 out of 20
Train_Loss: 0.0674, Duration: 0:00:55.161610
Running epoch 18 out of 20
Train_Loss: 0.0672, Duration: 0:00:55.225406
Running epoch 19 out of 20
Train_Loss: 0.0673, Duration: 0:00:55.333582
Running epoch 20 out of 20
Train_Loss: 0.0673, Duration: 0:00:55.327516
```

و عکس هارا به این صورت تشخیص میداد.



که خروجی های ما میشدند: