گزارش رمزگذار-رمزگشا با Pytorch

این کد پیاده‌سازی یک مدل انکودر-دیکودر (Encoder-Decoder) برای پردازش تصاویر دیتاست MNIST است. در این گزارش، هر کلاس و بخش از این پیاده‌سازی به تفصیل توضیح داده شده است.

کتابخانه‌ها

```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
```

ابتدا کتابخانه‌های مورد نیاز وارد می‌شوند torch برای کار با تنسورها و ساخت مدل‌های شبکه عصبی استفاده می‌شود torchvisionبرای بارگذاری و پردازش داده‌های تصویری، DataLoaderبرای مدیریت داده‌ها، nnبرای تعریف مدل‌های شبکه عصبی و optim برای بهینه‌سازی مدل مورد استفاده قرار می‌گیرند. همچنین، matplotlibبرای نمایش تصاویر به کار می‌رود.

تبدیل داده‌ها به تنسور

```
transform = transforms.Compose([
    transforms.ToTensor(),
])
```

در این قسمت، داده‌های تصویری به تنسور تبدیل می‌شوند تا بتوانند توسط PyTorch پردازش شوند.

بارگذاری دیتاستMNIST

```
train_dataset = datasets.MNIST(root='./data', train=True,
transform=transform, download=True)
batch_size = 32
train_loader = DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True)
```

دیتاست MNIST که شامل تصاویر دست‌نویس ارقام است، بارگذاری می‌شود DataLoader داده‌ها را به صورتbatch های کوچک بارگذاری می‌کند تا پردازش آن‌ها ساده‌تر شود.

کلاس رمزگذار (Encoder)

```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=2,
padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=2,
padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, stride=2,
padding=1)
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.relu = nn.ReLU()

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

کلاس Encoder یک شبکه عصبی کانولوشنی (Convolutional Neural Network) است که تصاویر ورودی را فشرده می‌کند. این کلاس شامل سه لایه کانولوشنی (Conv2d) است که هر کدام تعداد فیلترها را افزایش می‌دهند. همچنین یک لایه AdaptiveAvgPool2dبرای کاهش ابعاد خروجی و یک لایه ReLU برای فعال‌سازی به کار رفته‌اند.

کلاس رمزگشا(Decoder)

```
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.conv_transpose1 = nn.ConvTranspose2d(64, 32,
kernel_size=4, stride=1, padding=0)
        self.conv_transpose2 = nn.ConvTranspose2d(32, 16,
kernel_size=3, stride=2, padding=1, output_padding=0)
        self.conv_transpose3 = nn.ConvTranspose2d(16, 8,
kernel_size=4, stride=2, padding=1, output_padding=0)
        self.conv_transpose4 = nn.ConvTranspose2d(8, 1,
kernel_size=4, stride=2, padding=1, output_padding=0)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv_transpose1(x))
        x = self.relu(self.conv_transpose2(x))
        x = self.relu(self.conv_transpose3(x))
        x = self.conv_transpose4(x)
        return x
```

کلاس Decoder نیز یک شبکه عصبی کانولوشنی است که داده‌های فشرده‌شده را بازسازی می‌کند. این کلاس شامل چهار لایه کانولوشنی معکوس (ConvTranspose2d) است که به تدریج ابعاد تصاویر را به اندازه اولیه بازمی‌گردانند. لایه‌های ReLU نیز برای فعال‌سازی به کار رفته‌اند.

کلاس رمزگذار-رمزگشا(Encoder-Decoder)

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

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

کلاس EncoderDecoder مدل اصلی است که شامل دو بخش انکودر و دیکودر است. این کلاس داده‌های ورودی را ابتدا از طریق انکودر فشرده و سپس از طریق دیکودر بازسازی می‌کند.

آموزش مدل

```
encoder_decoder = EnocderDecoder()
criterion = nn.MSELoss()
optimizer = optim.Adam(encoder_decoder.parameters(), lr=0.001)

for epoch in range(10):
    for inputs, _ in train_loader:
        optimizer.zero_grad()
        outputs = encoder_decoder(inputs)
        loss = criterion(outputs, inputs)
        loss.backward()
        optimizer.step()
    print(f'Epoch {epoch+1}, Loss: {loss.item()}')
```

مدل رمزگذار-رمزگشا ساخته شده و با استفاده از MSELoss به عنوان تابع خطا و Adam به عنوان بهینه‌ساز آموزش داده می‌شود. مدل برای ۱۰ دوره آموزشی آموزش داده می‌شود و خطای هر دوره چاپ می‌شود.

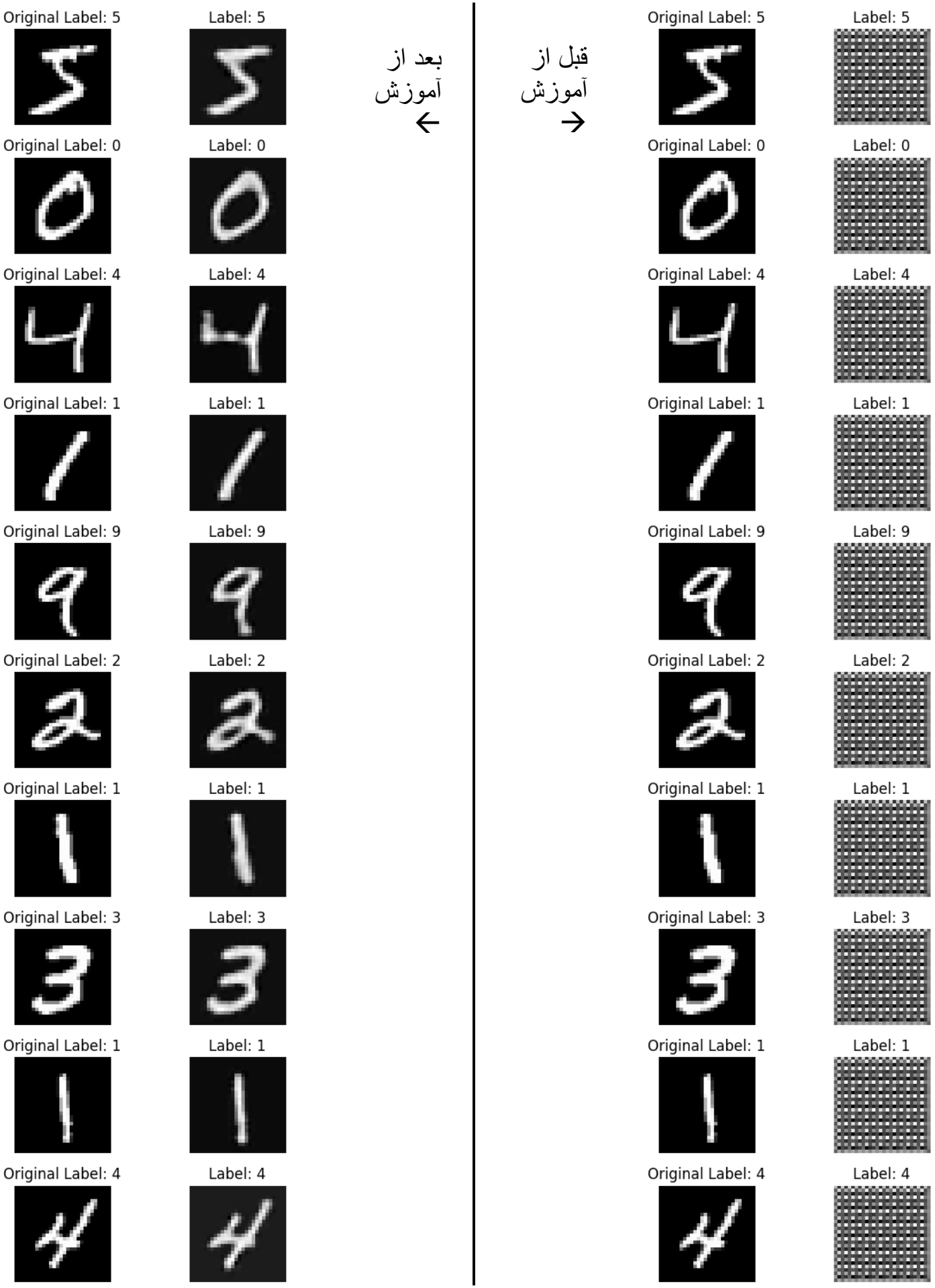
نمایش نتایج قبل و بعد از آموزش

نتایج مدل قبل و بعد از آموزش به صورت تصاویر نمایش داده می‌شوند.

```
fig, axes = plt.subplots(10, 2, figsize=(5, 15))
for i in range(10):
    image, label = train_loader.dataset[i]
    axes[i,0].imshow(image.squeeze(), cmap='gray')
    axes[i,0].set_title(f'Original Label: {label}')
    axes[i,0].axis('off')
    image = encoder_decoder(image.unsqueeze(0))
    axes[i,1].imshow(image.squeeze().detach().numpy(),
cmap='gray')
    axes[i,1].set_title(f'Label: {label}')
    axes[i,1].axis('off')
plt.tight_layout()
plt.show()
```

این بخش ابتدا تصاویری از دیتاست را بدون پردازش نمایش می‌دهد و سپس همان تصاویر را پس از بازسازی توسط مدل انکودر-دیکودر نشان می‌دهد. به این ترتیب می‌توان تاثیر مدل را بر روی بازسازی تصاویر مشاهده کرد.

همانطور که در صفحه بعد مشاهده میکنیم، تمامی برچسب ها بعد از آموزش به درستی توسط مدل تشخیص داده شده اند.


Original Label: 5
Label: 5
بعد از
آموزش
←
قبل از
آموزش
→
Original Label: 0
Label: 0
Original Label: 4
Label: 4
Original Label: 1
Label: 1
Original Label: 9
Label: 9
Original Label: 2
Label: 2
Original Label: 1
Label: 1
Original Label: 3
Label: 3
Original Label: 1
Label: 1
Original Label: 4
Label: 4