هانا اله یار پارسا

تمرین 3 بینایی

لینک کولب: لینک کولب

سوال1-

الف)

1. **تفاوت اصلی بین یک اتوانکودر استاندارد با یک وریشنال اتوانکودر:(VAE)**

- **اتوانکودر استاندارد** :(AE)یک مدل یادگیری عمیق است که برای فشرده‌سازی داده‌ها و یادگیری ویژگی‌های پنهان آن‌ها طراحی شده است. این مدل شامل دو بخش است :**انکودر** که ورودی‌ها را به یک نمای فشرده (فضای نهان) تبدیل می‌کند و **دیکودر** که نمای فشرده را دوباره به ورودی اصلی تبدیل می‌کند. در این مدل، فضای نهان معمولاً یک نمایش خاص از داده‌هاست که نیاز به هیچ توزیعی ندارد.
- **وریشنال اتوانکودر** :**(VAE)** این مدل شباهت‌هایی به AE دارد، اما با یک تفاوت اساسی: در فضای نهان، VAEها از توزیع‌های احتمالی (مثل توزیع نرمال) استفاده می‌کنند، نه تنها یک نقطه ثابت. این یعنی به جای اینکه تنها یک نمای خاص از داده‌ها داشته باشیم، برای هر داده یک توزیع در فضای نهان ساخته می‌شود که می‌تواند نمونه‌گیری‌های مختلف از آن داده را ایجاد کند. این ویژگی باعث می‌شود که مدل بتواند ویژگی‌های پیچیده‌تر و بیشتری از داده‌ها را یاد بگیرد و از آن‌ها برای تولید داده‌های جدید استفاده کند.

2. **چراVAE ها از توزیع احتمال در فضای نهان استفاده می‌کنند؟**

- استفاده از توزیع‌های احتمال در فضای نهان به مدل این امکان را می‌دهد که **تولید داده‌های جدید** (مانند تصاویری مشابه با داده‌های آموزش) را به راحتی انجام دهد. درVAE ، انکودر به جای تولید یک نقطه خاص در فضای نهان، دو پارامتر برای هر ورودی تولید می‌کند :**میانگین** و **انحراف معیار** که به یک توزیع نرمال منجر می‌شوند. این ویژگی به مدل کمک می‌کند که به جای تکیه بر ویژگی‌های خاص یک نقطه، از **مجموعه‌ای از احتمالات** برای تولید داده‌ها استفاده کند. این توزیع‌های احتمالی باعث می‌شوند که مدل به‌طور مؤثرتری داده‌های جدید و متنوع بسازد، زیرا می‌تواند نمونه‌گیری‌هایی از توزیع‌ها انجام دهد و تنوع در داده‌های تولیدی ایجاد کند.

به عبارت دیگر، استفاده از توزیع‌های احتمالی باعث می‌شودVAE ها نه تنها مدل‌های قدرتمندتری برای فشرده‌سازی باشند بلکه قابلیت **تولید داده‌های جدید و متنوع** را نیز داشته باشند.

ب)

درVAE ، تابع خطا شامل دو جزء اصلی است که به نام‌های **خطای بازسازی** و **خطای KL Divergence** شناخته می‌شوند. این دو جزء به طور همزمان در فرایند آموزش مدل به کار می‌روند و هرکدام نقش خاص خود را دارند:

**1.خطای بازسازی:(Reconstruction Error)**

- این جزء از تابع خطا مربوط به تفاوت بین ورودی اصلی و داده‌های بازسازی شده توسط دیکودر است. به طور معمول، این خطا با استفاده از معیارهایی مثل **میانگین مربعات خطا (MSE)** یا **آنتروپی متقابل** (Cross-Entropy)محاسبه می‌شود.

- هدف این است که مدل به‌طور دقیق‌تری ورودی‌ها را بازسازی کند. این خطا به‌طور غیرمستقیم به انکودر کمک می‌کند که ویژگی‌های پنهانی از داده‌ها را یاد بگیرد که برای بازسازی مجدد داده‌ها ضروری هستند.

**چرا ضروری است؟**

- این بخش از تابع خطا به مدل کمک می‌کند که ورودی‌ها را به صورت صحیح بازسازی کند و از این طریق یاد بگیرد که ویژگی‌های مهم داده‌ها را در فضای نهان ذخیره کند. بدون این جزء، مدل قادر به یادگیری ویژگی‌های مفید داده‌ها برای بازسازی نخواهد بود و عملکرد آن پایین خواهد بود.

**2.خطای KL Divergence (Divergence) یا تفاوت:(KL (Kullback-Leibler Divergence**

- این جزء از تابع خطا به مقایسه توزیع فضای نهان واقعی مدل (که توسط انکودر تولید می‌شود) و توزیع فرضی پیش‌فرض (معمولاً یک توزیع نرمال استاندارد) می‌پردازد. به عبارت دیگر، این بخش مدل را مجاب می‌کند که توزیع‌های احتمالی فضای نهانش را به توزیع نرمال نزدیک کند.

- KL Divergenceمیزان تفاوت بین توزیع احتمالی فضای نهان و توزیع استاندارد را اندازه‌گیری می‌کند و هدف این است که این تفاوت را به حداقل برسانیم.

**چرا ضروری است؟**

- هدف استفاده از KL Divergence این است که مدل فضای نهان را به‌طور منظم و هموار نگه دارد، به‌طوری که بتوان از آن برای تولید داده‌های جدید استفاده کرد. این جزء از تابع خطا کمک می‌کند تا فضای نهان مدل به شکل معقولی شکل بگیرد و توزیع‌های پیچیده‌تری از داده‌ها ایجاد نکند که نتوان از آن‌ها برای تولید داده‌های جدید استفاده کرد.

- همچنین این بخش از خطا باعث می‌شود که مدل از **توزیع پیش‌فرض** (معمولاً نرمال استاندارد) پیروی کند و از تولید داده‌های بی‌معنی یا غیرمنطقی در فضای نهان جلوگیری کند.

**ترکیب این دو جزء:**

تابع خطای کلی در VAE به صورت زیر ترکیب می‌شود:

$$\text{KL Divergence} \cdot \lambda + \text{Reconstruction Error} = \mathcal{L}$$

در اینجا، lambda( λ) معمولاً یک هایپرپارامتر است که تأثیر وزن KL Divergence را تعیین می‌کند.

**چرا ترکیب این دو جزء ضروری است؟**

- اگر تنها از خطای بازسازی استفاده کنیم، مدل ممکن است فضای نهانی بی‌نظم و پیچیده تولید کند که توانایی تولید داده‌های جدید از آن سخت باشد. از طرف دیگر، اگر فقط از KL Divergence استفاده کنیم، ممکن است مدل نتواند ویژگی‌های مهم داده‌ها را به‌خوبی یاد بگیرد. بنابراین، ترکیب این دو جزء باعث می‌شود که مدل هم بتواند داده‌های جدید و متنوع تولید کند و هم ویژگی‌های مهم ورودی‌ها را حفظ کند.

در نهایت، استفاده از هر دو جزء **خطای بازسازی** و **KL Divergence**به مدل کمک می‌کند که یک تعادل بین فشرده‌سازی داده‌ها و تولید داده‌های جدید به‌دست آورد.

ج)

VAEها به جای نگاشت داده‌ها به نقاط ثابت در فضای نهان، آن‌ها را به **توزیع‌های احتمالی** مانند **توزیع گاوسی** نگاشت می‌کنند به چند دلیل مهم:

**1.توانایی تولید داده‌های جدید:**

- زمانی که یک داده به یک **توزیع احتمالی** (مانند توزیع گاوسی) نگاشت می‌شود، می‌توان از این توزیع برای نمونه‌گیری داده‌های جدید استفاده کرد. به عبارت دیگر، به جای اینکه یک نقطه خاص در فضای نهان برای هر ورودی تعریف کنیم، با استفاده از توزیع‌های احتمالی می‌توانیم به راحتی از فضای نهان نمونه‌برداری کنیم و داده‌های جدید تولید کنیم.
- این ویژگی بهVAE ها این امکان را می‌دهد که **داده‌های جدید و متنوع** تولید کنند که مشابه با داده‌های آموزش باشند، اما در عین حال از داده‌های واقعی نباشند. این امر به ویژه برای **مدل‌های مولد** مانندVAE ها بسیار مفید است.

**2.بررسی و یادگیری ویژگی‌های پیچیده:**

- فضای نهان در VAE به گونه‌ای طراحی می‌شود که ویژگی‌های پیچیده‌تری از داده‌ها را ذخیره کند. استفاده از توزیع‌های احتمالی، به ویژه توزیع گاوسی، این امکان را فراهم می‌آورد که مدل بتواند **تنوع داده‌ها** را در فضای نهان به خوبی یاد بگیرد.
- برای هر داده، به جای یک نقطه خاص در فضای نهان، توزیعی از مقادیر (مثلاً میانگین و انحراف معیار توزیع گاوسی) تعریف می‌شود که این باعث می‌شود مدل ویژگی‌های مختلف داده‌ها را به‌طور کامل‌تری بیان کند.

**3.افزایش تعمیم‌پذیری مدل:**

- استفاده از توزیع‌ها باعث می‌شود که مدل در برابر **داده‌های جدید و ناآشنا** مقاومت بیشتری داشته باشد. به عبارت دیگر، فضای نهانی که از توزیع‌های احتمالی استفاده می‌کند، نسبت به فضای نهانی که تنها شامل نقاط ثابت است، **قابلیت تعمیم بالاتری** دارد. این باعث می‌شود که مدل به جای حفظ ویژگی‌های دقیق یک داده خاص، ویژگی‌های عمومی‌تری را یاد بگیرد که به آن اجازه می‌دهد داده‌های جدیدی تولید کند که مشابه با داده‌های آموزش باشند.

**4.کنترل بیشتر روی فضای نهان:**

- با استفاده از توزیع‌های احتمالی، مانند توزیع گاوسی، می‌توانیم **پارامترهای توزیع** (مانند میانگین و انحراف معیار) را به‌طور پیوسته تغییر دهیم. این به مدل این امکان را می‌دهد که فضای نهان را به‌طور دقیق‌تر و با قابلیت انعطاف‌پذیری بیشتر مدل کند. از طرف دیگر، اگر داده‌ها تنها به نقاط ثابت نگاشت شوند، این تغییرات پیوسته و کنترل دقیق بر روی فضای نهان امکان‌پذیر نخواهد بود.

**5.کمک به منظم‌سازی فضای نهان:**

- استفاده از توزیع‌های احتمالی باعث می‌شود که مدل به‌طور خودکار فضای نهان را به یک **فضای منظم و هموار** تبدیل کند که بتواند از آن برای تولید داده‌های جدید استفاده کند. به‌ویژه استفاده از **KL Divergence**در VAE به کمک مدل می‌آید تا فضای نهان به توزیع استاندارد (مثلاً نرمال گاوسی) نزدیک شود. این ویژگی به جلوگیری از تولید داده‌های بی‌معنی یا غیرمنطقی کمک می‌کند.

**نتیجه‌گیری:**

استفاده از توزیع‌های احتمالی مانند توزیع گاوسی درVAE ها باعث می‌شود که این مدل‌ها توانایی **تولید داده‌های جدید، یادگیری ویژگی‌های پیچیده، بهبود تعمیم‌پذیری و کنترل بهتر بر فضای نهان** را داشته باشند. این ویژگی‌ها بهVAE ها امکان می‌دهند که به عنوان مدل‌های مولد موفق عمل کنند و داده‌های متنوع و مشابه با داده‌های آموزشی تولید کنند.

بخش پیاده‌سازی:

برای پیاده‌سازی یک **Variational Autoencoder (VAE)**ساده برای بازسازی تصاویر از مجموعه داده **MNIST**، ابتدا نیاز داریم که یک مدل VAE بسازیم که از یک شبکه عصبی برای تولید توزیع احتمال (معمولاً یک توزیع نرمال) برای هر نمونه استفاده کند. سپس، از آن توزیع برای تولید یک نمونه جدید و بازسازی تصویر استفاده می‌کنیم.

در اینجا مراحل مختلف را شرح می‌دهم:

**.1طراحی مدل:VAE**

مدل VAE معمولاً از دو بخش اصلی تشکیل می‌شود:

- **Encoder**:این بخش ورودی (تصویر) را به یک بردار نهان (latent vector) تبدیل می‌کند.
- **Decoder**:این بخش بردار نهان را به تصویر بازسازی شده تبدیل می‌کند.

**.2اجزای تابع خطا:**

تابع خطای VAE از دو بخش اصلی تشکیل می‌شود:

1. **Reconstruction Loss**:که معمولاً از **خطای میانگین مربعات (MSE)** استفاده می‌شود تا تفاوت بین تصویر اصلی و بازسازی شده را اندازه‌گیری کند.
2. **KL Divergence Loss**:که فاصله بین توزیع احتمال پیش‌بینی شده توسط encoder و توزیع نرمال استاندارد را اندازه‌گیری می‌کند. این مقدار از **Kullback-Leibler Divergence**محاسبه می‌شود.

**.3ساختار مدل:**

- **Latent Space Dimensionality**:بعد بردار نهان را ۲ در نظر می‌گیریم، همانطور که درخواست کرده‌اید.
- **معیار ارزیابی** :برای ارزیابی کیفیت تصاویر بازسازی شده می‌توانیم از **MSE** (Mean Squared Error)یا **SSIM** (Structural Similarity Index)استفاده کنیم.
- **4 .توضیحات کد:**
- **Encoder**:ورودی را از طریق لایه‌های کانولوشنی پردازش کرده و به دو بردار (میانگین و لگاریتم واریانس) برای توزیع احتمال تبدیل می‌کند.
- **Decoder**:از بردار نهان (latent vector) برای بازسازی تصویر استفاده می‌کند.
- **VAE Loss**:تابع خطا شامل دو بخش است :
  1. **Reconstruction Loss**:که از **MSE**برای ارزیابی کیفیت بازسازی تصویر استفاده می‌کند.
  2. **KL Divergence Loss**:که توزیع نهان را با توزیع نرمال استاندارد مقایسه می‌کند.
- **reparameterize**:برای اعمال روش بازنمونه‌گیری در VAE استفاده می‌شود.

- **Training**: مدل برای ۱۰ دوره آموزش می‌بیند و در پایان بازسازی تصاویر را نمایش می‌دهد.

**6.ارزیابی کیفیت:**

برای ارزیابی کیفیت تصاویر بازسازی‌شده، می‌توان از معیار **MSE** یا **SSIM** استفاده کرد. در این کد از MSE به‌عنوان تابع خطا استفاده شده است، اما اگر بخواهید از SSIM برای ارزیابی کیفیت استفاده کنید، باید از کتابخانه‌هایی مانند **scikit-image** استفاده کنید.

Encoder:

```
# Hyperparameters
batch_size = 128
latent_dim = 2  # Dimensionality of the latent space
epochs = 10
learning_rate = 1e-3

# Load MNIST dataset
transform = transforms.ToTensor()
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Encoder
class Encoder(nn.Module):
    def __init__(self, latent_dim):
        super(Encoder, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, stride=2, padding=1)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, latent_dim)  # Mean of the latent space
        self.fc3 = nn.Linear(128, latent_dim)  # Log variance of the latent space

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        mean = self.fc2(x)
        log_var = self.fc3(x)
        return mean, log_var
```

Decoder:

```python
# Decoder
class Decoder(nn.Module):
    def __init__(self, latent_dim):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(latent_dim, 128)
        self.fc2 = nn.Linear(128, 64 * 7 * 7)
        self.deconv1 = nn.ConvTranspose2d(64, 32, 3, stride=2, padding=1, output_padding=1)
        self.deconv2 = nn.ConvTranspose2d(32, 1, 3, stride=2, padding=1, output_padding=1)

    def forward(self, z):
        z = torch.relu(self.fc1(z))
        z = torch.relu(self.fc2(z))
        z = z.view(z.size(0), 64, 7, 7)
        z = torch.relu(self.deconv1(z))
        z = torch.sigmoid(self.deconv2(z))
        return z
```

Model - loss:

```python
# VAE Model
class VAE(nn.Module):
    def __init__(self, latent_dim):
        super(VAE, self).__init__()
        self.encoder = Encoder(latent_dim)
        self.decoder = Decoder(latent_dim)

    def reparameterize(self, mean, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mean + eps * std

    def forward(self, x):
        mean, log_var = self.encoder(x)
        z = self.reparameterize(mean, log_var)
        x_reconstructed = self.decoder(z)
        return x_reconstructed, mean, log_var

# Loss function
def vae_loss(x, x_reconstructed, mean, log_var):
    # Reconstruction loss (MSE)
    reconstruction_loss = nn.functional.mse_loss(x_reconstructed, x, reduction='sum')

    # KL divergence loss
    kl_loss = -0.5 * torch.sum(1 + log_var - mean.pow(2) - log_var.exp())

    return reconstruction_loss + kl_loss
```

Training:

```
# Training loop
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
vae = VAE(latent_dim).to(device)
optimizer = torch.optim.Adam(vae.parameters(), lr=learning_rate)

for epoch in range(epochs):
    vae.train()
    train_loss = 0
    for data, _ in train_loader:
        data = data.to(device)
        optimizer.zero_grad()
        x_reconstructed, mean, log_var = vae(data)
        loss = vae_loss(data, x_reconstructed, mean, log_var)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

    print(f'Epoch {epoch+1}/{epochs}, Loss: {train_loss / len(train_loader.dataset)}')
```

Show result:

```
# Testing and visualizing results
vae.eval()
with torch.no_grad():
    data, _ = next(iter(train_loader))
    data = data.to(device)
    x_reconstructed, _, _ = vae(data)
    x_reconstructed = x_reconstructed.cpu()

    # Display original and reconstructed images
    for i in range(5):
        plt.subplot(2, 5, i+1)
        plt.imshow(data[i].cpu().squeeze(), cmap='gray')
        plt.axis('off')
        plt.subplot(2, 5, i+6)
        plt.imshow(x_reconstructed[i].squeeze(), cmap='gray')
        plt.axis('off')
    plt.show()
```

Result:


سوال 2-

الف)

یک **Generative Adversarial Network (GAN)**شامل دو شبکه عصبی است که به طور همزمان آموزش می‌بینند : **Generator**و .**Discriminator**این دو شبکه با یکدیگر به صورت **رقابتی** (Adversarial)عمل می‌کنند، به این معنی که هرکدام سعی می‌کند دیگری را شکست دهد. در نتیجه، GANقادر است داده‌هایی مشابه داده‌های واقعی تولید کند.

**معماری:GAN**

1. **Generator ( مولد:(**

- وظیفه **Generator**تولید داده‌هایی است که به نظر واقعی می‌آیند. این شبکه ورودی تصادفی (مانند **Noise**یا نویز) را دریافت کرده و سعی می‌کند داده‌ای مشابه با داده‌های واقعی تولید کند.
- **ورودی** :**Generator** ورودی معمولاً یک **بردار تصادفی** (معمولاً از توزیع نرمال یا یکنواخت) است که به عنوان نمونه‌ای از فضای نهان (latent space) در نظر گرفته می‌شود.

- **خروجی** :**Generator** خروجی Generator معمولاً یک داده مصنوعی است، مثلاً یک تصویر که از نظر ساختاری مشابه با داده‌های واقعی است.
- **نقش** :**Generator** هدف Generator این است که با تولید داده‌های مصنوعی به‌گونه‌ای عمل کند که Discriminatorنتواند آن‌ها را از داده‌های واقعی تشخیص دهد .Generator سعی می‌کند به مرور زمان داده‌هایی تولید کند که ویژگی‌های واقعی‌تری داشته باشند.

2. **) Discriminatorتشخیص‌دهنده:(**

- وظیفه **Discriminator**ارزیابی این است که آیا داده‌ای که دریافت می‌کند واقعی است یا تولید شده توسط.Generator
- **ورودی** Discriminator :**Discriminator** داده‌هایی دریافت می‌کند که می‌توانند از دو منبع مختلف باشند: داده‌های واقعی از مجموعه داده‌های آموزشی یا داده‌های تولید شده توسط.Generator
- **خروجی** Discriminator :**Discriminator** معمولاً یک مقدار احتمال (در بازه 0 تا 1) تولید می‌کند که نشان می‌دهد داده ورودی **واقعی** است (نزدیک به 1) یا **جعلی** است (نزدیک به 0).
- **نقش** :**Discriminator** هدف Discriminator این است که توانایی شناسایی داده‌های واقعی از داده‌های تولید شده را پیدا کند .Discriminator سعی می‌کند یاد بگیرد که داده‌های مصنوعی را از داده‌های واقعی تمیز دهد.

**فرآیند آموزش:GAN**

- **Generator**و **Discriminator**به طور همزمان و در یک رقابت بهینه می‌شوند. این فرآیند به این صورت است :

1. Generatorیک نمونه از داده‌های مصنوعی تولید می‌کند.
2. Discriminatorداده‌ها را ارزیابی می‌کند و پیش‌بینی می‌کند که آیا این داده واقعی است یا مصنوعی.
3. Generatorاز اشتباهات Discriminator استفاده می‌کند تا داده‌های بهتری تولید کند.
4. Discriminatorنیز از اشتباهات خود برای بهتر شناسایی داده‌های مصنوعی استفاده می‌کند.

هدف نهایی این است که Generator داده‌هایی تولید کند که از نظر Discriminator به حدی واقعی به نظر برسند که تشخیص واقعی یا جعلی بودن آن‌ها سخت شود.

**نقش و وظایف هر جزء:**

- **Generator**:
  - تولید داده‌های مصنوعی که شبیه به داده‌های واقعی هستند.
  - با توجه به بازخورد ازDiscriminator ، به مرور زمان داده‌هایی تولید می‌کند که شبیه‌تر به داده‌های واقعی می‌شوند.
- **Discriminator**:
  - تشخیص داده‌های واقعی از داده‌های تولید شده توسط.Generator

- به مرور زمان یاد می‌گیرد که ویژگی‌های داده‌های واقعی و مصنوعی را از یکدیگر تمیز دهد.

**هدف نهایی:GAN**

- هدف کلی در GAN این است که Generator به تولید داده‌هایی برسد که تا حد امکان **غیرقابل تمایز** از داده‌های واقعی باشند، به طوری که Discriminator نتواند تفاوت آن‌ها را تشخیص دهد. در این حالت، Generatorبه یک مدل تولیدی بسیار قدرتمند تبدیل می‌شود که قادر به تولید داده‌های واقع‌نما است.

**خلاصه:**

- **Generator**سعی می‌کند داده‌هایی تولید کند که شبیه داده‌های واقعی باشند.
- **Discriminator**تلاش می‌کند داده‌های واقعی را از داده‌های مصنوعی تمیز دهد.
- این دو شبکه به صورت رقابتی آموزش می‌بینند، به طوری که هرکدام به طور مداوم سعی می‌کند دیگری را به چالش بکشد تا به تدریج هر دو به بهترین عملکرد خود برسند.

ب)

تابع زیان در **Generative Adversarial Network (GAN)**به صورت **رقابتی** طراحی می‌شود به دلیل رقابت بین دو شبکه اصلی مدل **Generator** :و .**Discriminator**این دو شبکه به گونه‌ای آموزش می‌بینند که هرکدام تلاش می‌کند عملکرد دیگری را بهبود دهد، که این فرآیند به یادگیری بهتر هر دو مدل منجر می‌شود.

**تعریف تابع زیان:GAN**

تابع زیان GAN به طور کلی از دو بخش اصلی تشکیل می‌شود:

1. **زیان :(D) Discriminator** که هدف آن تشخیص این است که آیا داده واقعی است یا تولید شده توسط Generator.
2. **زیان :(G) Generator** که هدف آن فریب دادن Discriminator است تا داده‌های تولید شده را واقعی شبیه‌سازی کند.

فرض کنید D(x)D(x)احتمال واقعی بودن داده ورودی xxاست که توسط Discriminator پیش‌بینی می‌شود و G(z)داده‌ای است که توسط Generator از ورودی تصادفی z تولید می‌شود.

**تابع زیان:Discriminator**

زیان Discriminator را می‌توان به صورت زیر نوشت:

$$[(D(G(z)) - \mathbb{E}_{z\sim p}[\log(1 - \mathbb{E}_{\text{data}x\sim P}[\log D(x)]- = {}_{D}L$$

- در اینجا $x$ داده واقعی است که از توزیع داده‌های واقعی ${}_{\text{data}}P$ نمونه‌برداری شده است.
- $G(z)$ داده‌ای است که توسط Generator از ورودی تصادفی $z$ تولید شده است.
- $D(x)$ احتمال واقعی بودن داده $x$ است که Discriminator پیش‌بینی می‌کند.
- هدف Discriminator این است که:
- $D(x)$ برای داده‌های واقعی به 1 نزدیک باشد.
- $D(G(z))$ برای داده‌های تولید شده به 0 نزدیک باشد.

**تابع زیان Generator:**

زیان Generator را می‌توان به صورت زیر نوشت:

$$\mathbb{E}_{z\sim p}[\log D(G(z))]- = {}_{G}L$$

- هدف Generator این است که $D(G(z))$ را به 1 نزدیک کند، یعنی Generator سعی می‌کند Discriminator را فریب دهد تا فکر کند داده‌های تولیدی آن واقعی هستند.

**چرا تابع زیان به صورت رقابتی طراحی شده است؟**

تابع زیان GAN به طور رقابتی طراحی شده است زیرا هدف اصلی GAN ایجاد یک **رقابت بین Generator و Discriminator**است. این رقابت باعث می‌شود که هرکدام از این دو شبکه به طور مداوم در تلاش برای بهبود عملکرد خود باشند:

1. **Generator**سعی می‌کند داده‌های مصنوعی تولید کند که به‌قدری شبیه به داده‌های واقعی باشند که Discriminatorنتواند آن‌ها را تشخیص دهد. این امر باعث می‌شود Generator به تولید داده‌هایی با ویژگی‌های واقعی‌تر بپردازد.
2. **Discriminator**وظیفه دارد تا داده‌های واقعی را از داده‌های تولیدی تمیز دهد Discriminator .تلاش می‌کند ویژگی‌های داده‌های واقعی را یاد بگیرد و از آن‌ها برای شناسایی داده‌های مصنوعی استفاده کند.

به دلیل این رقابت، GANبه یک بازی دوطرفه (two-player game) تبدیل می‌شود که در آن:

- **Generator**در تلاش است تا Discriminator را فریب دهد.
- **Discriminator**در تلاش است تا داده‌های واقعی را از داده‌های مصنوعی تمیز دهد.

**نتیجه رقابت:**

این رقابت باعث می‌شود که هرکدام از شبکه‌ها به طور پیوسته در حال بهبود باشند. در نهایت، وقتی که این رقابت به تعادل برسد، Generatorقادر خواهد بود داده‌هایی تولید کند که **غیرقابل تمایز** از داده‌های واقعی باشند) به عبارت دیگر، Discriminatorنتواند تفاوت بین داده‌های واقعی و تولید شده را تشخیص دهد.(

**خلاصه:**

- **تابع زیان GAN** شامل دو جزء است: زیان Discriminator و زیان.Generator
- **زیان Discriminator** تلاش می‌کند داده‌های واقعی را به 1 و داده‌های تولید شده را به 0 نزدیک کند.
- **زیان Generator** تلاش می‌کند تا داده‌های تولیدی را طوری تولید کند که Discriminator آن‌ها را واقعی تشخیص دهد.
- این تابع به صورت **رقابتی** طراحی شده است چون Generator و Discriminator در حال رقابت برای بهبود عملکرد خود هستند، که منجر به یادگیری بهتر هر دو شبکه می‌شود.

ج)

در سال‌های اخیر، معماری‌های مختلفی از **Generative Adversarial Networks (GANs)** معرفی شده‌اند که بهبودهای قابل توجهی در تولید داده‌های مصنوعی به‌ویژه در زمینه تولید تصاویر، ترجمه تصویر به تصویر، و تولید تصاویر با جزئیات بالا فراهم کرده‌اند. در اینجا، پیشرفت‌های برجسته‌ای مانند **DCGAN**، **StyleGAN**، و **CycleGAN** مورد بررسی قرار می‌گیرند و تفاوت‌های این مدل‌ها با **GAN استاندارد** توضیح داده می‌شود.

**1. DCGAN (Deep Convolutional GAN):**

- **پیشرفت DCGAN** :از شبکه‌های عصبی کانولوشنی (CNN) برای بهبود کیفیت تولید تصاویر استفاده می‌کند. این مدل از لایه‌های کانولوشنی در Generator و Discriminator برای یادگیری ویژگی‌های پیچیده‌تر تصاویر و تولید تصاویر با کیفیت بهتر استفاده می‌کند.
- **تفاوت‌ها** با **GAN استاندارد**:
  - در **GAN استاندارد** معمولاً از **شبکه‌های پرسپترون چند لایه (MLP)** استفاده می‌شود که برای پردازش داده‌های غیر ساختاری مناسب‌تر است. اما **DCGAN** به طور خاص برای پردازش داده‌های تصویری طراحی شده و از **لایه‌های کانولوشنی** برای هر دو بخش Generator و Discriminator استفاده می‌کند.
  - این مدل به‌ویژه برای تولید **تصاویر طبیعی** کاربرد دارد، به‌طوری که ساختارهای پیچیده‌تری از داده‌ها مانند لبه‌ها، بافت‌ها، و ویژگی‌های هندسی تصاویر را بهتر یاد می‌گیرد.
  - همچنین در DCGAN از **Batch Normalization** و **Leaky ReLU** به‌عنوان توابع فعال‌سازی استفاده می‌شود که به پایداری بیشتر آموزش کمک می‌کند.

**2. StyleGAN:**

- **پیشرفت StyleGAN** :توسط **NVIDIA** معرفی شد و به‌طور خاص برای تولید تصاویر با **جزئیات و کیفیت بسیار بالا** (به‌ویژه تصاویر چهره) طراحی شده است. در این معماری، از یک تکنیک خاص به نام **Style-based Generator Architecture** استفاده می‌شود که به Generator این امکان را می‌دهد که ویژگی‌های مختلف تصویر (مثل نورپردازی، حالت چهره، پس‌زمینه، و جزئیات دیگر) را به‌طور مستقل از یکدیگر کنترل کند.
- **تفاوت‌ها** با **GAN استاندارد**:

- در **GANاستاندارد،** Generatorداده‌های تصادفی را به یک فضای نهان معمولی تبدیل می‌کند و سپس آن‌ها را به داده‌های مصنوعی تبدیل می‌کند. در حالی که در **StyleGAN،** Generatorورودی تصادفی را به چندین سطح از ویژگی‌های مختلف (که به‌طور جداگانه کنترل می‌شوند) تبدیل می‌کند.
- این معماری با استفاده از **مقیاس‌بندی ویژگی‌ها** (Style Mixing)و **شبکه‌های عصبی کانونی** (Adaptive Instance Normalization)به Generator اجازه می‌دهد که ویژگی‌های مختلف تصویر را به‌صورت دقیق‌تر کنترل کند. این باعث می‌شود که **تصاویر تولیدی** دارای **جزئیات بیشتری** و بیشتر شبیه به داده‌های واقعی باشند.
- StyleGANبه‌ویژه در **تولید چهره‌های انسانی** با ویژگی‌هایی مانند **عمر، جنسیت، و وضعیت صورت** کنترل‌پذیر موفقیت‌های بزرگی به‌دست آورده است.

**3. CycleGAN:**

- **پیشرفت CycleGAN** :به‌ویژه برای **ترجمه تصویر به تصویر** (Image-to-Image Translation)طراحی شده است که قادر است بدون نیاز به جفت داده‌های همسان، تصاویر را از یک دامنه به دامنه دیگر تبدیل کند. این مدل به صورت **غیرمزدوج** عمل می‌کند، یعنی نیازی به داشتن داده‌های همتای واقعی و مصنوعی نیست.
- **تفاوت‌ها** با **GANاستاندارد**:

- در **GANاستاندارد،** Generatorتلاش می‌کند داده‌های تصادفی را به داده‌های واقعی مشابه تولید کند. در حالی که در **CycleGAN،** مدل تلاش می‌کند تا تصاویر یک دامنه (مثلاً تصاویر تابلو نقاشی) را به تصاویر دامنه دیگر (مثلاً تصاویر واقعی) تبدیل کند، بدون اینکه داده‌های جفت‌شده‌ای از هر دو دامنه داشته باشد.
- CycleGANاز **دوره‌های حفظ (Cycle Consistency Loss)** استفاده می‌کند تا اطمینان حاصل کند که تصویر بازسازی‌شده از دامنه هدف، مشابه تصویر اصلی در دامنه اولیه است. به عبارت دیگر، وقتی یک تصویر از دامنه XXبه دامنه YYتبدیل می‌شود و سپس دوباره به دامنه XXبرمی‌گردد، باید به تصویر اصلی نزدیک باشد.
- CycleGANبه‌طور خاص برای **ترجمه تصاویر غیرمزدوج** استفاده می‌شود که در کاربردهایی مانند تبدیل سبک‌های هنری، تبدیل تابلو به عکس، و حتی در تصاویر پزشکی کاربرد دارد.

**تفاوت‌های کلی بین این مدل‌ها و GANاستاندارد:**

- **DCGAN**از لایه‌های کانولوشنی برای تولید تصاویر با کیفیت بالا و ساختارهای پیچیده‌تر استفاده می‌کند، در حالی که GAN استاندارد بیشتر برای داده‌های غیر ساختاری مناسب است.
- **StyleGAN**از معماری خاصی برای کنترل دقیق ویژگی‌های مختلف تصویر به‌طور مستقل استفاده می‌کند، که باعث تولید تصاویر با جزئیات بسیار بالا و با قابلیت تنظیم ویژگی‌های مختلف (مانند نورپردازی، حالت چهره، پس‌زمینه و غیره) می‌شود. این ویژگی در GAN استاندارد وجود ندارد.
- **CycleGAN**به‌طور خاص برای ترجمه‌های تصویر به تصویر بدون نیاز به داده‌های همسان طراحی شده است و از **Lossحفظ چرخه** استفاده می‌کند. این در حالی است که GAN استاندارد معمولاً فقط برای تولید داده‌های مشابه از ورودی‌های تصادفی استفاده می‌شود.

**نتیجه‌گیری:**

این معماری‌ها هرکدام ویژگی‌های خاص خود را دارند که به حل مشکلات مختلف GAN استاندارد می‌پردازند:

- **DCGAN**کیفیت و ساختار تصاویر را بهبود می‌بخشد.
- **StyleGAN**جزئیات تصاویر را به دقت کنترل می‌کند و امکان تولید تصاویر بسیار واقعی را فراهم می‌آورد.
- **CycleGAN**به شما اجازه می‌دهد که تصاویر را بدون جفت داده‌های همسان از یک دامنه به دامنه دیگر ترجمه کنید.

این پیشرفت‌ها به‌طور چشمگیری توانایی‌های GAN را در زمینه‌های مختلف مانند **تولید تصاویر واقعی، ترجمه تصویر به تصویر و کنترل دقیق ویژگی‌های تصویر** افزایش داده‌اند.

د)

**Variational Autoencoders (VAE)**و **Generative Adversarial Networks (GAN)**دو معماری معروف در زمینه **مدل‌های تولیدی** هستند که به‌طور مشابه برای **تولید داده‌های جدید** استفاده می‌شوند، اما هرکدام ویژگی‌های خاص خود را دارند. در اینجا، به **شباهت‌ها و تفاوت‌ها** میان این دو مدل و **شرایط استفاده از هرکدام** می‌پردازیم.

**شباهت‌ها بین VAE و:GAN**

1. **هدف مشترک**:
   - هر دو مدل، یعنی **VAE**و **GAN**، هدفشان تولید داده‌های جدید مشابه داده‌های واقعی است. این داده‌های تولیدی می‌توانند تصاویر، صدا، یا دیگر نوع داده‌ها باشند.
2. **مدل‌های تولیدی**:
   - هر دو مدل از **شبکه‌های عصبی** برای یادگیری توزیع داده‌ها و تولید داده‌های جدید استفاده می‌کنند. در VAEها، این فرایند از طریق یادگیری توزیع‌های احتمالی و درGAN ها از طریق رقابت بین Generatorو Discriminator انجام می‌شود.
3. **آموزش بر اساس داده‌های واقعی**:
   - هر دو مدل از داده‌های واقعی برای آموزش استفاده می‌کنند و در نهایت هدفشان این است که تولیداتی مشابه با داده‌های واقعی ایجاد کنند.

**تفاوت‌ها بین VAE و GAN:**

| ویژگی | VAE | GAN |
|---|---|---|
| **ساختار اصلی** | شامل **انکودر** (Encoder)و **دیکودر** (Decoder)است که داده‌ها را به فضای نهان تبدیل می‌کنند و سپس آن‌ها را بازسازی می‌کنند. | شامل دو شبکه **Generator** :(مولد) و **Discriminator**(تشخیص‌دهنده) که به طور رقابتی کار می‌کنند. |
| **توزیع فضای نهان** | از **توزیع احتمالی** (مثل توزیع نرمال) در فضای نهان استفاده می‌کند. | فضای نهان معمولاً شامل نقاط مشخص است، نه توزیع احتمالی. |
| **فرآیند تولید داده** | Generatorداده‌های جدید را از **نمونه‌گیری از توزیع احتمالی** در فضای نهان تولید می‌کند. | Generatorبه‌طور مستقیم داده‌های جدید را از **نویز تصادفی** تولید می‌کند. |
| **پایداری در آموزش** | معمولاً آموزش **پایدارتر** است و از ویژگی‌هایی مانند **KL Divergence**برای تنظیم فضای نهان استفاده می‌کند. | ممکن است در آموزش **ناپایدار** باشد و نیاز به تنظیم دقیق و **رقابت بین دو شبکه** دارد. |
| **نوع مدل تولیدی** | **مدل احتمالی** است و توزیع‌های احتمالی را می‌آموزد. | **مدل تعیینی** است و به‌طور مستقیم داده‌های خاص را تولید می‌کند. |
| **دقت در تولید داده‌ها** | معمولاً داده‌های تولیدی به اندازهGAN ها واقعی به نظر نمی‌رسند، اما برای داده‌های **محتوای مختلف** (نه فقط تصاویر) مناسب است. | توانایی تولید **تصاویر با جزئیات و واقع‌گرایانه‌تر** را دارد، مخصوصاً در مواردی که تولید داده‌های پیچیده مانند تصاویر واقعی مدنظر باشد. |
| **ساده بودن پیاده‌سازی** | ساده‌تر است و آموزش آن معمولاً کمتر نیاز به تنظیمات پیچیده دارد. | پیچیده‌تر است و آموزش آن ممکن است زمان‌بر و دشوارتر باشد. |

**شرایطی که یکی نسبت به دیگری ترجیح داده می‌شود:**

1. **VAE (Variational Autoencoders)**ترجیح داده می‌شود زمانی که:
   - **پایداری در آموزش** مهم استVAE :ها به دلیل استفاده از **KL Divergence**و مدل‌های احتمالی، آموزش پایدارتری دارند.
   - **مدل‌های احتمالی** مورد نیاز هستندVAE :ها به‌طور طبیعی توزیع‌های احتمالی یاد می‌گیرند و برای استفاده در **دسته‌بندی** یا **کلاس‌بندی** داده‌ها مناسب هستند.
   - **تولید داده‌های متنوع و کاربردی** مهم است: در صورتی که نیاز به تولید داده‌های متنوع از یک توزیع خاص داریم (مثلاً در مدل‌های ترکیب داده‌ها یا یادگیری توزیع‌های پیچیده)، VAEها گزینه مناسبی هستند.

- **نیاز به فشرده‌سازی داده‌ها** وجود دارد: در صورتی که به فشرده‌سازی داده‌ها نیاز باشد، VAEها می‌توانند نمایه‌ای فشرده و منظم از داده‌ها بسازند.

2. **GAN (Generative Adversarial Networks)**ترجیح داده می‌شود زمانی که:

- **تولید داده‌های واقعی و با جزئیات بالا** مهم استGAN :ها به‌ویژه در تولید **تصاویر واقعی و با کیفیت بالا** (مثل چهره‌ها، اشیاء و غیره) بهتر عمل می‌کنند و معمولاً تصاویر تولیدی بسیار واقعی به نظر می‌رسند.
- **رقابت و بهینه‌سازی دقیق** لازم استGAN :ها معمولاً توانایی تولید داده‌های بسیار دقیق و واقع‌گرایانه را دارند، به شرطی که تنظیمات آموزشی به دقت انجام شود.
- **زمان و منابع محاسباتی زیاد** در اختیار است: آموزشGAN ها معمولاً پیچیده‌تر است و نیاز به منابع محاسباتی بیشتری دارد، به‌ویژه در فرآیند **تنظیم**.**Hyperparameters**
- **مدل‌های تعیینی** مورد نیاز استGAN :ها برای مواقعی که باید داده‌های خاص و دقیق تولید شوند (مثلاً تولید تصویر خاص از یک توزیع محدود) بسیار مفید هستند.

**نتیجه‌گیری:**

- **VAEها** به دلیل ساختار احتمالی‌شان، معمولاً برای کاربردهایی که به **پایداری در آموزش** و **توزیع‌های احتمالی** نیاز دارند، مناسب‌تر هستند.
- **GANها** به دلیل توانایی‌های خود در تولید **تصاویر با کیفیت بالا و واقعی**، به‌ویژه در زمینه‌های **تولید تصاویر طبیعی و واقعی**، برVAE ها ارجحیت دارند، اما نیاز به تنظیمات دقیق و رقابت پیچیده‌ای دارند که ممکن است آموزش آن‌ها را دشوارتر کند.

بنابراین، انتخاب بین VAE و GAN به هدف و نیازهای خاص پروژه بستگی دارد.

پیاده سازی:

## گزارش پیاده‌سازی مدل انتشار (Diffusion Model) برای مجموعه داده CIFAR-10

---

**مقدمه**

در این پروژه، هدف پیاده‌سازی یک مدل انتشار (Diffusion Model) برای مجموعه داده CIFAR-10 است. این مدل از فرآیند انتشار (افزودن نویز گوسی) و یک فرآیند معکوس (حذف نویز) برای بازسازی تصاویر استفاده می‌کند. همچنین مدل با استفاده از شرط کلاس، توانایی تولید تصاویر باکیفیت از کلاس‌های خاص (مانند گربه و هواپیما) را دارد.

---

**مراحل پیاده‌سازی**

**1.بارگذاری داده‌ها**

ابتدا مجموعه داده CIFAR-10 بارگذاری شد. این داده‌ها شامل تصاویر 32×32 از 10 کلاس مختلف است. داده‌ها با استفاده از کتابخانه PyTorch و با اعمال نرمال‌سازی آماده‌سازی شدند.

**2.فرآیند انتشار**

برای شبیه‌سازی فرآیند انتشار، نویز گوسی به تصاویر اضافه شد. شدت نویز با گذشت زمان افزایش یافت تا تأثیر مراحل مختلف نویزگذاری مشخص شود. این فرآیند با تابع `add_noise`پیاده‌سازی شد. نویزگذاری روی تصاویر اصلی انجام شده و نتایج برای مراحل مختلف تجسم شدند:

- نویز کم(Step 0)
- نویز متوسط(Step 5)
- نویز زیاد(Step 10)

**3.طراحی مدل شبکه عصبی(UNet)**

یک مدل U-Net برای انجام فرآیند معکوس طراحی شد. این مدل از دو بخش اصلی تشکیل شده است:

- **Encoder**:کاهش ابعاد و استخراج ویژگی‌های نویزی.
- **Decoder**:بازسازی تصویر اصلی از داده‌های نویزی .مدل با شرط کلاس (One-Hot Encoding) ترکیب شد تا قابلیت بازسازی تصاویر مرتبط با کلاس خاص را داشته باشد.

**4.آموزش مدل**

مدل با استفاده از داده‌های نویزی و تصاویر اصلی آموزش داده شد. برای این کار:

- از **تابع خطای MSE** برای کاهش تفاوت بین تصویر بازسازی‌شده و تصویر اصلی استفاده شد.
- از **Adam Optimizer**برای بهینه‌سازی وزن‌های شبکه استفاده شد .مدل در طول 5 دوره آموزشی (`Epoch`) تمرین داده شد.

**5.ارزیابی و تجسم**

برای ارزیابی مدل:

- تصاویر نویزی به مدل داده شدند.
- تصاویر بازسازی‌شده توسط مدل با تصاویر اصلی مقایسه شدند.
- نتایج به‌صورت گرافیکی نمایش داده شدند.

---

**نتایج و تحلیل**

1. **فرآیند انتشار** :تصاویر نویزی در مراحل مختلف نشان دادند که افزایش نویز تأثیر قابل‌توجهی بر کیفیت تصاویر دارد. نویز کم تصاویر را کمتر مخدوش می‌کند، در حالی که نویز زیاد جزئیات تصویر را از بین می‌برد.
2. **بازسازی تصاویر** :مدل توانست نویز را حذف کرده و تصاویر اصلی را با کیفیت خوبی بازسازی کند. نتایج برای کلاس‌های مختلف (مانند گربه و هواپیما) نیز بررسی شدند.
3. **عملکرد مدل شرطی** :مدل با توجه به شرط کلاس توانست تصاویر مربوط به کلاس خاصی را تولید کند. برای مثال:

- کلاس گربه: تصویر بازسازی‌شده شباهت زیادی به گربه داشت.
- کلاس هواپیما: بازسازی تصویر هواپیما با جزئیات مناسب انجام شد.

---

**چالش‌ها و راهکارها**

- **چالش تنظیم ابعاد شرط کلاس** :ترکیب شرط کلاس با داده‌های تصویری نیازمند هماهنگی دقیق ابعاد بود که با استفاده از One-Hot Encoding و گسترش ابعاد حل شد.
- **تأثیر تعداد مراحل انتشار** :افزایش تعداد مراحل انتشار (افزودن نویز) باعث کاهش کیفیت بازسازی شد. این موضوع در تجزیه و تحلیل کیفیت بازسازی در گزارش نموداری مشخص شد.

---

**نتیجه‌گیری**

مدل پیاده‌سازی شده توانست فرآیند انتشار و معکوس را به‌طور موفقیت‌آمیز انجام دهد. استفاده از شرط کلاس قابلیت تولید تصاویر باکیفیت برای کلاس‌های مشخص را بهبود بخشید. این روش می‌تواند در زمینه‌های مختلف مانند تولید تصاویر باکیفیت و حذف نویز از تصاویر واقعی استفاده شود.

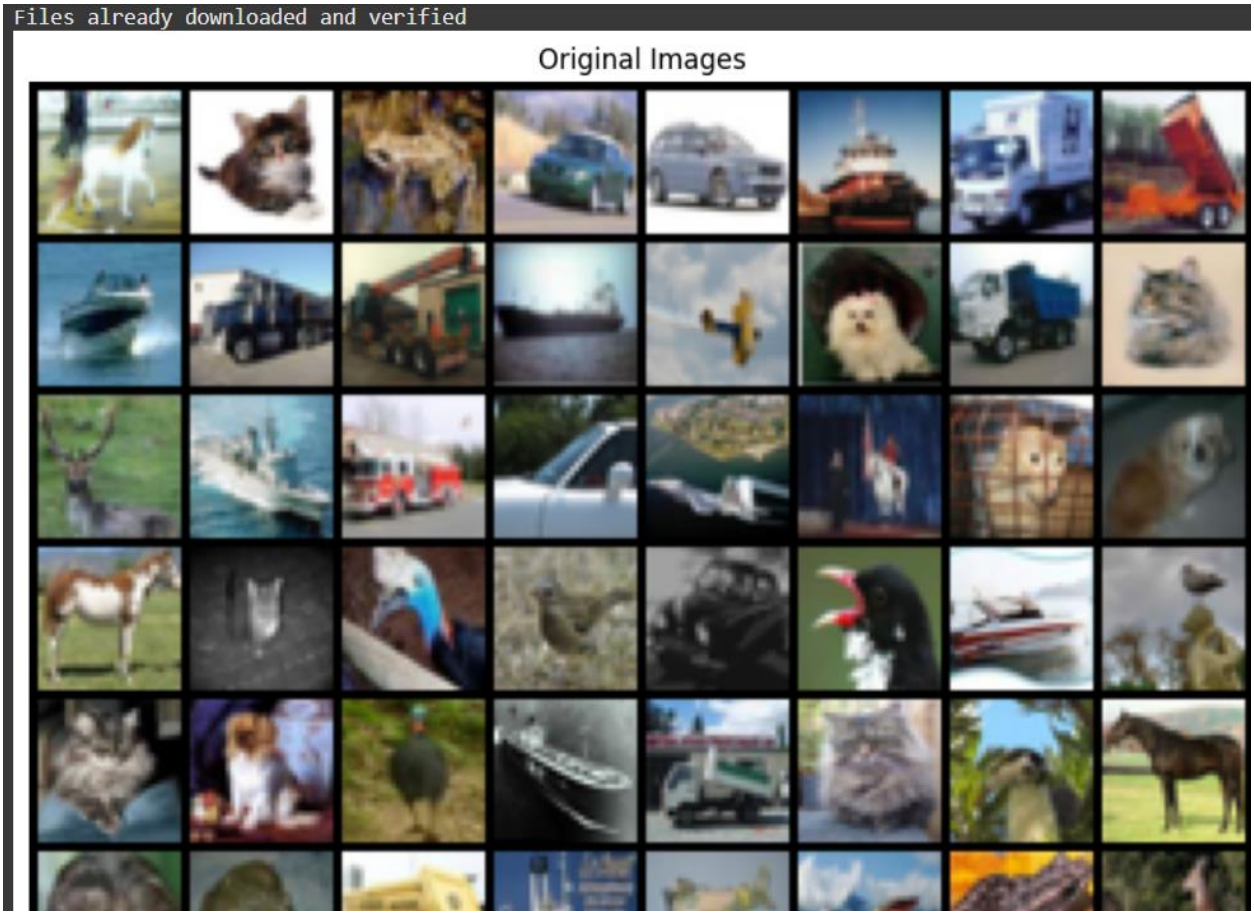
```python
# Initial setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

# Load CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

# Function to display images
def show_images(images, title=""):
    grid = torchvision.utils.make_grid(images, nrow=8, normalize=True)
    plt.figure(figsize=(10, 10))
    plt.title(title)
    plt.imshow(grid.permute(1, 2, 0).cpu().numpy())
    plt.axis('off')
    plt.show()

# Add Gaussian noise to images
def add_noise(images, t, noise_strength=0.1):
    noise = torch.randn_like(images) * noise_strength * (t / 10)
    noisy_images = images + noise
    return torch.clip(noisy_images, -1, 1)

# Display original images
dataiter = iter(trainloader)
images, labels = next(dataiter)
show_images(images, title="Original Images")
```

```python
# Visualize noisy images at different steps
for t in [0, 2, 5, 10]:
    noisy_images = add_noise(images, t)
    show_images(noisy_images, title=f"Step {t}: Noise Level {t/10}")

# Design the U-Net neural network model
class UNet(nn.Module):
    def __init__(self, num_classes):
        super(UNet, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(3 + num_classes, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 3, kernel_size=2, stride=2),
            nn.Tanh()
        )

    def forward(self, x, condition):
        # Convert condition to one-hot
        batch_size = x.shape[0]
        num_classes = 10  # Number of classes
        one_hot_condition = torch.zeros((batch_size, num_classes), device=x.device)
        one_hot_condition[torch.arange(batch_size), condition] = 1

        # Expand dimensions to match input
        one_hot_condition = one_hot_condition.unsqueeze(-1).unsqueeze(-1)
        one_hot_condition = one_hot_condition.repeat(1, 1, x.shape[-2], x.shape[-1])
```

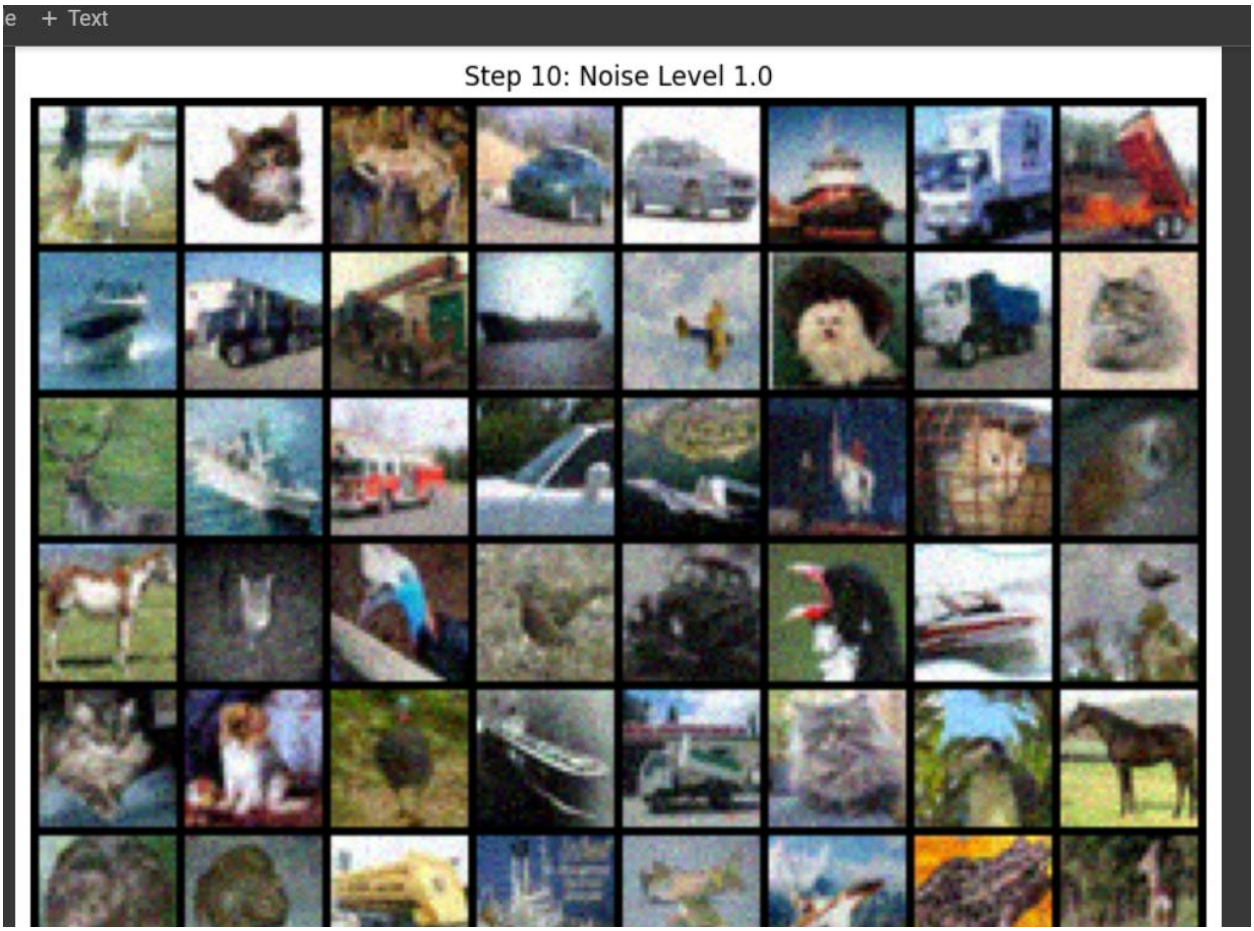
```
        # Combine with input
        x = torch.cat((x, one_hot_condition), dim=1)
        x = self.encoder(x)
        x = self.decoder(x)
        return x

# Function to train the model
def train_model(model, dataloader, num_epochs=10):
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.MSELoss()

    model.train()
    for epoch in range(num_epochs):
        epoch_loss = 0
        for images, labels in dataloader:
            images = images.to(device)
            labels = labels.to(device)

            # Add noise to images
            noisy_images = add_noise(images, t=5)

            # Reconstruct images
            outputs = model(noisy_images, labels)
            loss = criterion(outputs, images)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()
```

```
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()

        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}")

# Example usage of the model
unet_model = UNet(num_classes=10).to(device)
train_model(unet_model, trainloader, num_epochs=5)

# Test the reconstruction of images
def test_model(model, dataloader):
    model.eval()
    with torch.no_grad():
        for images, labels in dataloader:
            images = images.to(device)
            labels = labels.to(device)
            noisy_images = add_noise(images, t=5)
            outputs = model(noisy_images, labels)
            show_images(noisy_images, title="Noisy Images")
            show_images(outputs, title="Reconstructed Images")
            break

test_model(unet_model, trainloader)
```

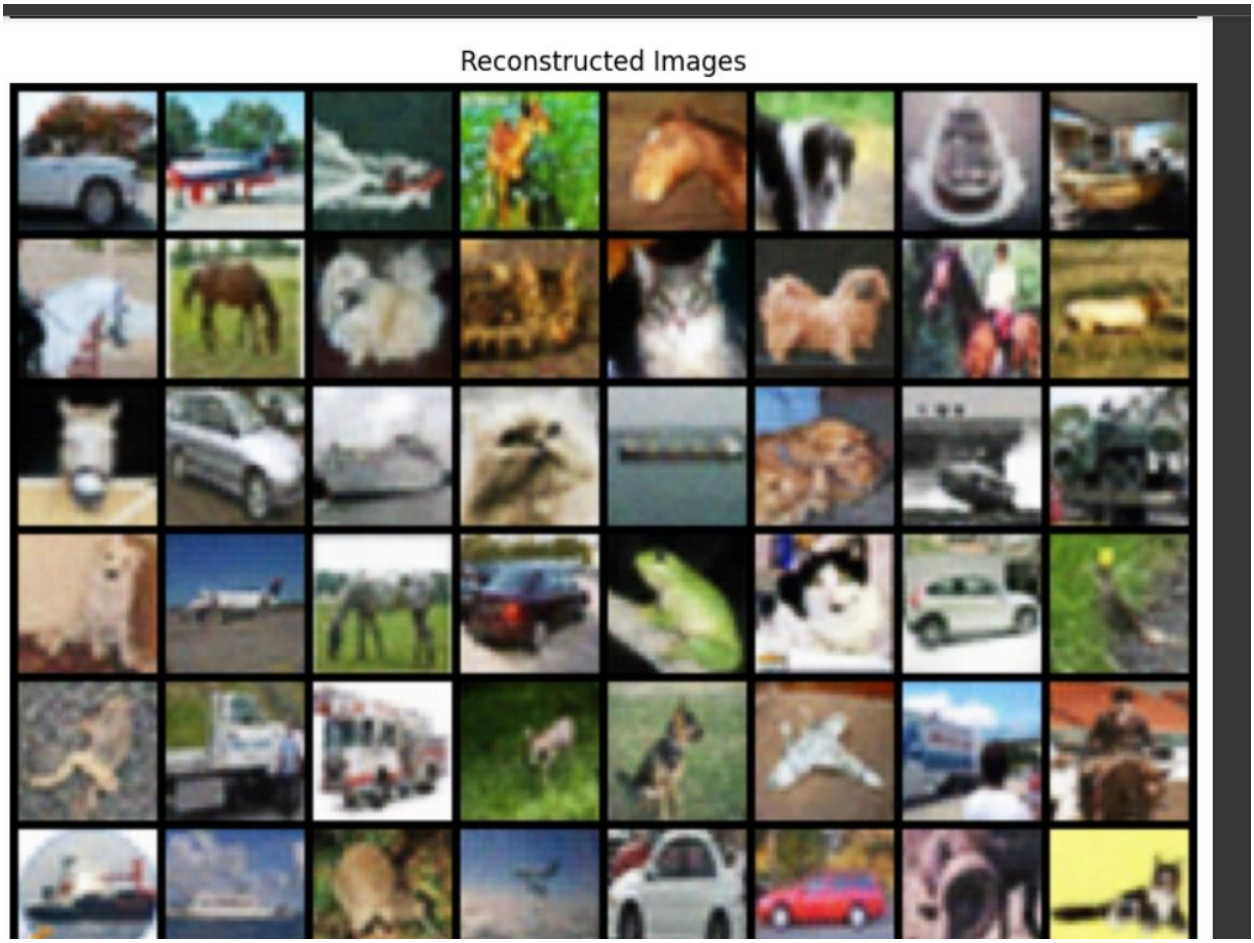
تصویر اصلی:




تصویر نویزی:




تصویری که گرفتیم و رفع نویز شده:


Reconstructed Images