CPSC 8430 Homework#4

Email: xiaofey@clemson.edu

Name: Xiaofeng Yang

CUID: C73220300

Github Link: https://github.com/Xiaoo112/CPSC-8430_HW4

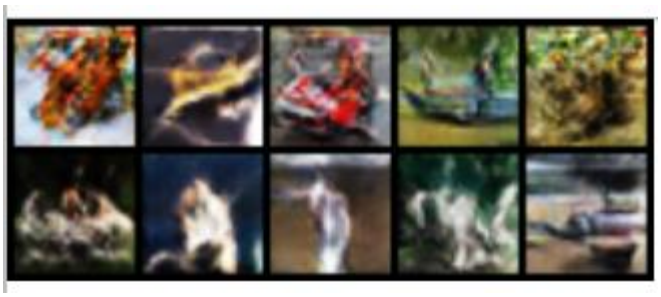
April 17, 2024

Homework#4 DCGAN WGAN and ACGAN

1. DCGAN
    a. DCGAN Architecture

```
class Generator_DCGAN(nn.Module):
    def __init__(self):
        super(Generator_DCGAN, self).__init__()
        self.network = nn.Sequential(
            nn.ConvTranspose2d(in_channels=100, out_channels=1024, kernel_size=4, stride=1, padding=0),
            nn.BatchNorm2d(num_features=1024),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.ConvTranspose2d(in_channels=1024, out_channels=512, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(num_features=512),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(num_features=256),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.ConvTranspose2d(in_channels=256, out_channels=3, kernel_size=4, stride=2, padding=1),
            nn.Tanh())

    def forward(self, input_tensor):
        return self.network(input_tensor)

class Discriminator_DCGAN(nn.Module):
    def __init__(self):
        super(Discriminator_DCGAN, self).__init__()
        self.network = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=256, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(num_features=512),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(num_features=1024),
            nn.LeakyReLU(negative_slope=0.2, inplace=True),
            nn.Conv2d(in_channels=1024, out_channels=1, kernel_size=4, stride=1, padding=0),
            nn.Sigmoid())

    def forward(self, input_tensor):
        return self.network(input_tensor)
```

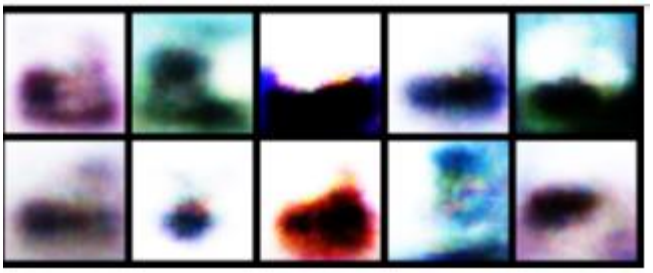
    b. Best 10 generated pictures for DCGAN:



2. WGAN
    a. WGAN architecture

```
class GeneratorWGAN(nn.Module):
    def __init__(self):
        super(GeneratorWGAN, self).__init__()

        def create_block(input_features, output_features, normalize=True):
            layers = [nn.Linear(input_features, output_features)]
            if normalize:
                layers.append(nn.BatchNorm1d(output_features, 0.8))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.network = nn.Sequential(
            *create_block(100, 128, normalize=False),
            *create_block(128, 256),
            *create_block(256, 512),
            *create_block(512, 1024),
            nn.Linear(1024, 3*32*32),
            nn.Tanh()
        )

    def forward(self, noise_vector):
        image = self.network(noise_vector)
        image = image.view(image.shape[0], 3, 32, 32)
        return image

class DiscriminatorWGAN(nn.Module):
    def __init__(self):
        super(DiscriminatorWGAN, self).__init__()

        self.network = nn.Sequential(
            nn.Linear(3*32*32, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
        )

    def forward(self, image):
        image_flat = image.view(image.shape[0], -1)
        validity = self.network(image_flat)
        return validity
```

    b. Best 10 generated pictures for WGAN:



3. ACGAN
    a. ACGAN architecture

```
class GeneratorACGAN(nn.Module):
    def __init__(self):
        super(GeneratorACGAN, self).__init__()
        self.embedding = nn.Embedding(10, 100)
        self.fully_connected = nn.Linear(100, 128 * 64)  # Simplified expression for 128 * 8 * 8
        self.generator_network = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 128, 3, stride=1, padding=1),
            nn.BatchNorm2d(128, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 3, 3, stride=1, padding=1),
            nn.Tanh()
        )

    def forward(self, noise, labels):
        combined_input = torch.mul(self.embedding(labels), noise)
        transformed_input = self.fully_connected(combined_input)
        reshaped_input = transformed_input.view(transformed_input.shape[0], 128, 8, 8)
        output_image = self.generator_network(reshaped_input)
        return output_image

class DiscriminatorACGAN(nn.Module):
    def __init__(self):
        super(DiscriminatorACGAN, self).__init__()

        def discriminator_block(input_channels, output_channels, use_batchnorm=True):
            layers = [nn.Conv2d(input_channels, output_channels, 3, 2, 1), nn.LeakyReLU(0.2, inplace=True), nn.Dropout2d(0.25)]
            if use_batchnorm:
                layers.append(nn.BatchNorm2d(output_channels, 0.8))
            return layers

        self.discriminator_network = nn.Sequential(
            *discriminator_block(3, 16, use_batchnorm=False),
            *discriminator_block(16, 32),
            *discriminator_block(32, 64),
            *discriminator_block(64, 128),
        )

        self.adversarial_layer = nn.Sequential(nn.Linear(128 * 4, 1), nn.Sigmoid())
        self.classification_layer = nn.Sequential(nn.Linear(128 * 4, 10), nn.Softmax(dim=1))

    def forward(self, image):
        processed_image = self.discriminator_network(image)
        flattened_output = processed_image.view(processed_image.shape[0], -1)
        validity_output = self.adversarial_layer(flattened_output)
        label_output = self.classification_layer(flattened_output)
        return validity_output, label_output
```

    b. Best 10 generated pictures for ACGAN:

4. Performance comparison among DCGAN, WGAN, and ACGAN that trained from scratch