APPENDICES

Appendix 1. Synthetic Data for Cat Class

Appendix 2. Synthetic Data for Deer Class

Appendix 3. Synthetic Data for Dog Class

Appendix 4. Synthetic Data for Horse Class

Appendix 5. Code Snippet of DCGAN Generator Architecture

```
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 64 x 64
        )

    def forward(self, input):
        return self.main(input)
```

Appendix 6. Code Snippet of DCGAN Discriminator Architecture

```
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

Appendix 7. Code Snippet for Generating Synthetic Data

```
from torchvision.utils import save_image

def  save_generated_images(generator,  num_images,  save_path,  device,
latent_dim):
    generator.eval()
    os.makedirs(save_path, exist_ok=True)

    with torch.no_grad():
        for i in range(num_images):
            noise = torch.randn(1, latent_dim, 1, 1).to(device)
            fake_image = generator(noise)
            normalized_fake_image = (fake_image + 1) / 2
       save_image(normalized_fake_image[0],os.path.join(save_path,f'gen
erated_image_{i+1  }.png'))

if __name__ == '__main__':
    num_images = 5200
    save_path = "generated\\cat"
    latent_dim = 100
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
    gen_path = 'models/generator_parameters.pt'

    generator = Generator(latent_dim).to(device)
    generator.load_state_dict(torch.load(gen_path))

      save_generated_images(generator,  num_images,  save_path,  device,
latent_dim)
```

Appendix 8. Code Snippet of EfficientNet-B0 Classification Model

```
class CustomEfficientNetB0(nn.Module):
    def __init__(self, num_classes):
        super(CustomEfficientNetB0, self).__init__()
        self.efficientnet = models.efficientnet_b0(pretrained=False)
        self.in_features = self.efficientnet.classifier[1].in_

        self.efficientnet.classifier[1] = nn.Identity()
        self.fc = nn.Linear(self.in_features, num_classes)

    def forward(self, x):
        features = self.efficientnet(x)
        return self.fc(features)

efficientnet = CustomEfficientNetB0(num_classes=num_classes).to(device)
```

Appendix 9. Code Snippet of MobileNetV2 Classification Model

```
class CustomMobileNetV2(nn.Module):
    def __init__(self, num_classes):
        super(CustomMobileNetV2, self).__init__()
        self.mobilenet = models.mobilenet_v2(pretrained=False)
        self.in_features = self.mobilenet.classifier[1].in_features
        self.mobilenet.classifier[1] = nn.Identity()
        self.fc = nn.Linear(self.in_features, num_classes)

    def forward(self, x):
        features = self.mobilenet(x)
        return self.fc(features)
```

Appendix 10. Source Code

The full code can be accessed via the GitHub repository provided below. https://github.com/Fieter955/Thesis