Individual Final Report

Kismat Khatri

1. Introduction

Our project focuses on creating realistic dog images using a technology called Generative Adversarial Networks (GANs). We are intrigued by the wide range of dog breeds and their unique features, which makes this task both challenging and exciting. This work is not just about making new images; it's also about pushing the boundaries in AI-based art and improving data for other machine learning projects. We use a specific type of GAN known as Deep Convolutional GANs (DCGANs) and UNET2d Diffusers to make square-shaped images of different dog breeds. Our main aim is to make these images look as real as possible. To achieve this, we experiment with different loss functions, adjust hyper-parameters, and add more convolutional layers to our models.

2. Description of my individual work

(a) Models Built

Initially, I focused on building and training a Traditional GAN model to generate photorealistic dog images, experimenting with various configurations for optimal performance. This involved adjusting batch sizes, experimenting with different loss functions, and fine-tuning hyperparameters like learning rates, optimizers, and epochs. But the result from this model is quite bad and unclear. So, we decided to go with DCgan model and Unet2dDiffuser model for our final report and presentations at class.

(b) Streamlit App

To effectively demonstrate our GAN models' capabilities, I developed an interactive Streamlit application. This platform allowed users to view generated dog images in real-time, offering a dynamic and engaging way to assess the effectiveness of different model architectures. The application, designed to compare outputs from various models, became an essential tool for analysis and presentation, particularly in our class presentations, showcasing the advancements in generating high-quality dog images.

(c) Report and slide writing

Finished the report and slides with team members.

3. Describe the portion of the work

Model Architecture:

```python
# ================================== Generator ==================================
class Generator(nn.Module):
    def __init__(self, NZ, NGF, NC):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(NZ, NGF),
            nn.BatchNorm1d(NGF),
            nn.ReLU(True),
            nn.Linear(NGF, NGF * 2),
            nn.BatchNorm1d(NGF * 2),
            nn.ReLU(True),
            nn.Linear(NGF * 2, NGF * 4),
            nn.BatchNorm1d(NGF * 4),
            nn.ReLU(True),
            nn.Linear(NGF * 4, NC * IMAGE_SIZE * IMAGE_SIZE),
            nn.Tanh()
        )

    def forward(self, input):
        output = self.main(input)
        return output.view(-1, NC, IMAGE_SIZE, IMAGE_SIZE)

# ================================== Discriminator ==================================
class Discriminator(nn.Module):
    def __init__(self, NDF, NC):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(NC * IMAGE_SIZE * IMAGE_SIZE, NDF * 4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(NDF * 4, NDF * 2),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(NDF * 2, NDF),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(NDF, 1),
            nn.Sigmoid()
        )

    def forward(self, input):
        output = input.view(-1, NC * IMAGE_SIZE * IMAGE_SIZE)
        return self.main(output)
```

Training Loop:

```
for epoch in range(NUM_EPOCHS):
    for i, data in enumerate(dataloader, 0):
        # Update Discriminator: maximize log(D(x)) + log(1 - D(G(z)))
        netD.zero_grad()
        real_data = data[0].to(device)
        b_size = real_data.size(0)
        label = torch.full((b_size,), real_label, dtype=torch.float, device=device)
        output = netD(real_data).view(-1)
        errD_real = criterion(output, label)
        errD_real.backward()

        # Generate batch of latent vectors
        noise = torch.randn(b_size, NZ, device=device)
        fake = netG(noise)

        label.fill_(fake_label)
        output = netD(fake.detach()).view(-1)
        errD_fake = criterion(output, label)
        errD_fake.backward()

        errD = errD_real + errD_fake
        optimizerD.step()

        # Update Generator: maximize log(D(G(z)))
        netG.zero_grad()
        label.fill_(real_label)
        output = netD(fake).view(-1)
        errG = criterion(output, label)
        errG.backward()
        optimizerG.step()

        if i % 100 == 0:
            print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f\tD(x): %.4f\tD(G(z)): %.4f'
                  % (epoch, NUM_EPOCHS, i, len(dataloader), errD.item(), errG.item(), output.mean().item(), errG.item()))
```

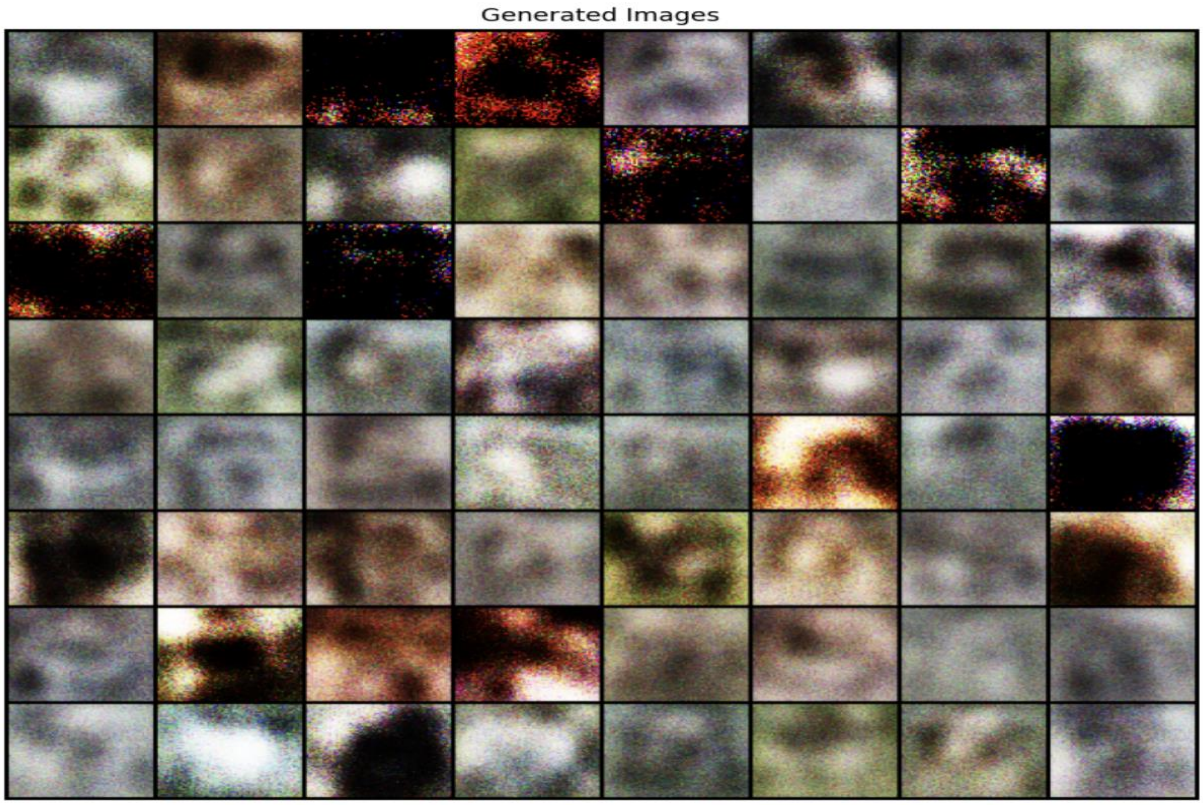
Streamlit App:

```
def load_model(model_path):
    try:
        filename = os.path.basename(model_path)

        if 'dcgan' in filename.lower():
            model = Generator(ngpu).to(device)  # DCGAN Generator
        elif 'gan' in filename.lower():
            model = TraditionalGANGenerator(NZ, NGF, NC).to(device)  # Traditional GAN Generator
        elif 'unet2' in filename.lower():
            model = load_unet2_diffuser_model(model_path)  # UNet2 Model
        else:
            raise ValueError(f"Unsupported model type in filename: {filename}")

        model.load_state_dict(torch.load(model_path, map_location=device))
        model.eval()
        return model
    except Exception as e:
        st.error(f"Error loading model: {e}")
        return None


def load_unet2_diffuser_model(model_path):
    try:
        model = UNet2DModel.from_pretrained(model_path, use_safetensors=True, local_files_only=True)
        print("Model loaded successfully.")
        return model
    except Exception as e:
        print(f"Error loading UNet2D model: {e}")
        return None

def generate_unet2_image(image_path, model):
    image = Image.open(image_path)
    resized_image = image.resize((128, 128))
    transform = transforms.ToTensor()
    resized_image = transform(resized_image)
    resized_image = resized_image.unsqueeze(0)

    noise_scheduler = DDPMScheduler(num_train_timesteps=10000)
    noise = torch.randn(resized_image.shape)
    timesteps = torch.LongTensor([0])
    noisy_image = noise_scheduler.add_noise(resized_image, noise, timesteps)

    output = model(noisy_image, timesteps, return_dict=False)[0]
```

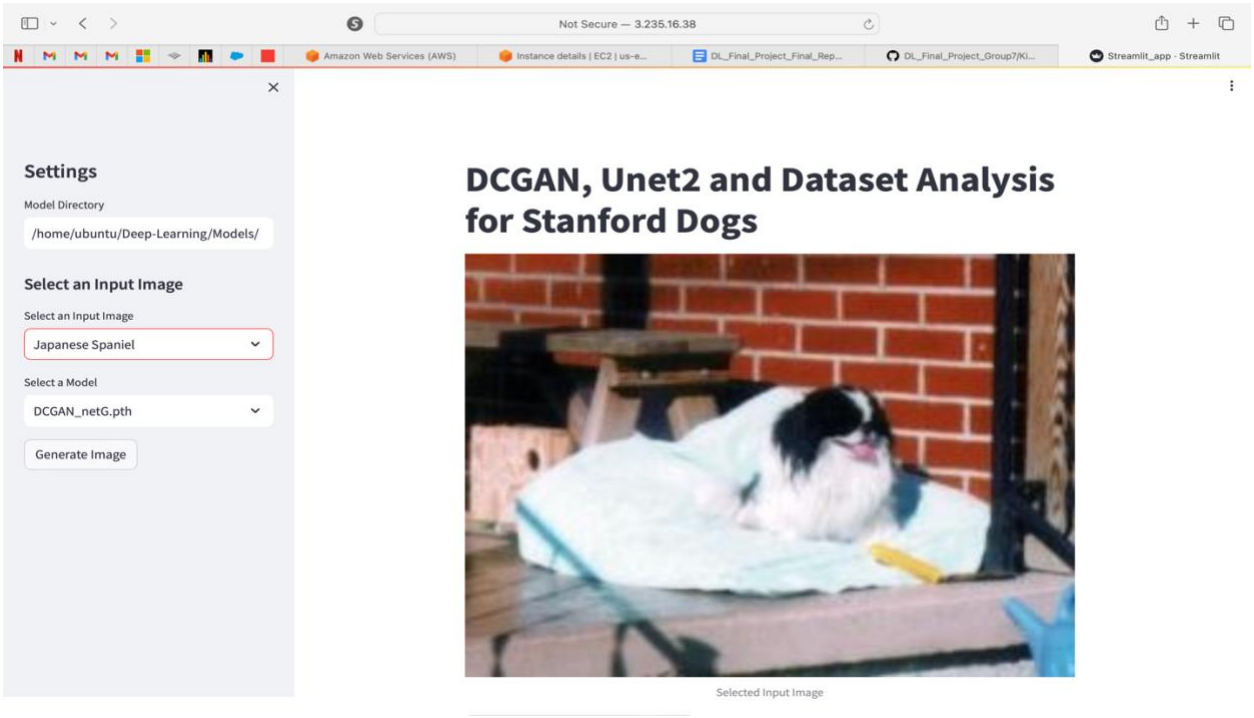
```python
# Dropdown menu for pre-selected images
st.sidebar.header('Select an Input Image')
image_options = {
    "Chihuahua": "/home/ubuntu/Deep-Learning/Data/StanfordDogs/Images/n02085620-Chihuahua/n02085620_10074.jpg",
    "Japanese Spaniel": "/home/ubuntu/Deep-Learning/Data/StanfordDogs/Images/n02085782-Japanese_spaniel/n02085782_1058.jpg",
    "Maltese Dog": "/home/ubuntu/Deep-Learning/Data/StanfordDogs/Images/n02085936-Maltese_dog/n02085936_10297.jpg",
    "African Hunting Dog": "/home/ubuntu/Deep-Learning/Data/StanfordDogs/Images/n02116738-African_hunting_dog/n02116738_10476.jpg"
}
selected_image_name = st.sidebar.selectbox('Select an Input Image', list(image_options.keys()))
selected_image_path = image_options[selected_image_name]

# Display the selected image
input_image = Image.open(selected_image_path)
st.image(input_image, caption='Selected Input Image', use_column_width=True)

# Load model for DCGAN
model_files = os.listdir(MODEL_DIR)
selected_model = st.sidebar.selectbox('Select a Model', model_files)
model_path = os.path.join(MODEL_DIR, selected_model)
netG = load_model(model_path)

# Button to generate images using DCGAN
if st.sidebar.button('Generate Image'):
    with st.spinner('Generating Image...'):
        model_type = 'dcgan' if 'dcgan' in selected_model.lower() else 'gan'
        if 'unet2' in selected_model.lower():
            # Try loading directly from the link instead of local file
            unet2_model = load_unet2_diffuser_model("JeffreyHuLLaMA2/DogDiffusion")
            if unet2_model is not None:
                output_image = generate_unet2_image(selected_image_path, unet2_model)  # Adjust arguments as needed
                st.image(output_image, caption='Generated Image', use_column_width=True)
            else:
                st.error("Failed to load UNet2 model.")
        else:
            output_tensor = generate_images(netG, model_type, NZ, device).squeeze(0).detach().cpu()
            output_image = transforms.ToPILImage()(output_tensor)
            st.image(output_image, caption='Generated Image', use_column_width=True)
```

4. Results

Image Generated By Traditional GAN Model



Result of Streamlit App



Input Image in Streamlit App

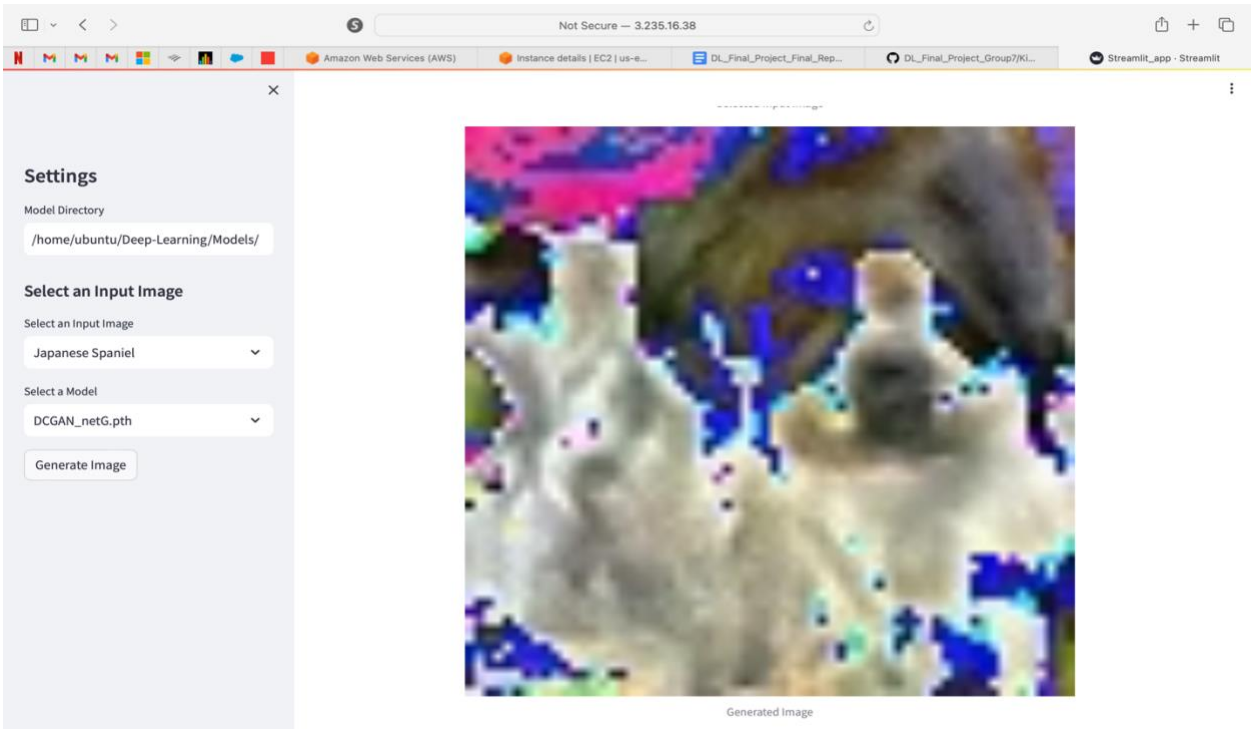
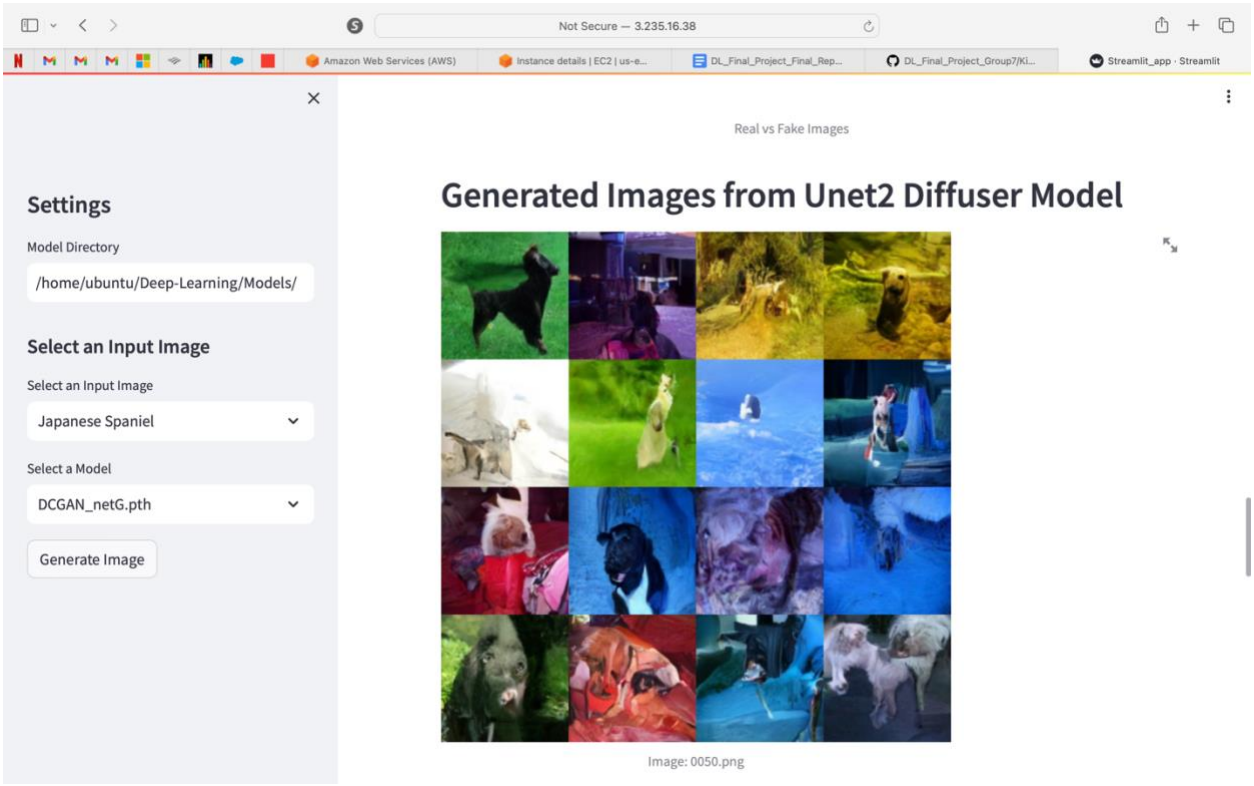


Image Generated by DCGAN Model in streamlit App



Generated image of Unet2D Diffuser Model In Streamlit App

5. Conclusion and Future Work

Contributions and Implications

This project stands as a testament to the potential of GANs in generating high-fidelity images. The success of this project opens avenues in digital art creation, training data enhancement, and possibly in educational tools for veterinary studies.

Future Directions

Future work could explore the integration of more advanced GAN architectures and the expansion of the dataset to include more varied and challenging images. Additionally, the application of these models in other domains of image generation presents an exciting opportunity for further research.

6. References

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Ozair, S., Bengio, Y. (2014). Generative Adversarial Networks. arXiv preprint arXiv:1406.2661.

Rocca, J. Understanding Generative Adversarial Networks (GANs).

Brownlee, J. A Gentle Introduction to Generative Adversarial Networks (GANs).

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv preprint arXiv:1511.06434.

Hui, J. GAN — DCGAN (Deep convolutional generative adversarial networks).

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. (2016). Photo-realistic single image super-resolution using a generative adversarial network. arXiv preprint arXiv:1609.04802.

Streamlit. (n.d.). Retrieved from https://streamlit.io