Final Term Project

Advanced Intelligent Systems and Deep Learning

Facial Diffusion Generation: A Multi-Ethnic Face Generation Project

Yaswanth Ijjurotula (101745505)

Shashaank Lnu (1017929430)

Table of Contents:

1. Introduction

Diffusion Models have emerged as a robust approach for image synthesis tasks in generative modeling. Inspired by physical processes like diffusion, these models generate high-quality images by iteratively denoising noisy data. This report delves into developing and implementing a conditional Diffusion Model aimed at generating images of different ethnic groups (Europeans, Indians, and Orientals) based on class labels. The approach employs a UNet-based architecture combined with techniques like Exponential Moving Average (EMA), self-attention, and gradient checkpointing to optimize memory usage and enhance training stability.

2. Project Description

This project focuses on implementing a **Conditional Diffusion Model** to generate synthetic images representing distinct ethnic groups: Europeans, Indians, and Orientals. The model is built using a UNet-based architecture, incorporating advanced features like self-attention, conditional embeddings, and Exponential Moving Average (EMA) for enhanced training and generalization. The objective is to explore the capabilities of Diffusion Models in conditional image synthesis and evaluate their performance in generating high-quality images conditioned on class labels.

Key Features

1. **Conditional Diffusion Model**:
   The model leverages a noise addition-removal framework to generate images by learning the underlying data distribution. Conditional embeddings ensure that the generated images align with the specified class labels.

2. **UNet Architecture**:
   A multi-resolution network capable of encoding and decoding information effectively. It includes skip connections to preserve spatial details and improve reconstruction quality.

3. **Self-Attention Mechanism**:
   Integrated within the architecture to allow the model to focus on spatially relevant features, enhancing the quality of generated images.

4. **Training Optimizations**:
   - **Gradient Checkpointing**: Reduces memory usage by recomputing intermediate states during backpropagation.
   - **EMA**: Maintains a smoothed version of the model weights for better generalization.
   - **Learning Rate Scheduling**: Implements a cosine decay strategy to optimize convergence.
   - **Gradient Clipping**: Ensures stable training by preventing exploding gradients.

5. **Custom Dataset and Augmentation**:
   The model is trained on a dataset with images belonging to three ethnic groups. Augmentation techniques like horizontal flipping, minor rotations, and color jittering are used to improve model robustness.

Implementation Goals

The primary goals of this project are:

1. **Generate High-Quality Images**: Synthesize realistic images conditioned on specified class labels.
2. **Efficient Resource Utilization**: Leverage memory-efficient techniques to enable training on limited hardware resources.
3. **Evaluate Model Performance**: Use metrics such as Mean Squared Error (MSE) and qualitative assessments to measure the effectiveness of the Diffusion Model.

3. Tasks Accomplished

Tasks Accomplished in the Project

The following tasks were accomplished to achieve the project's objectives:

1. Data Preparation

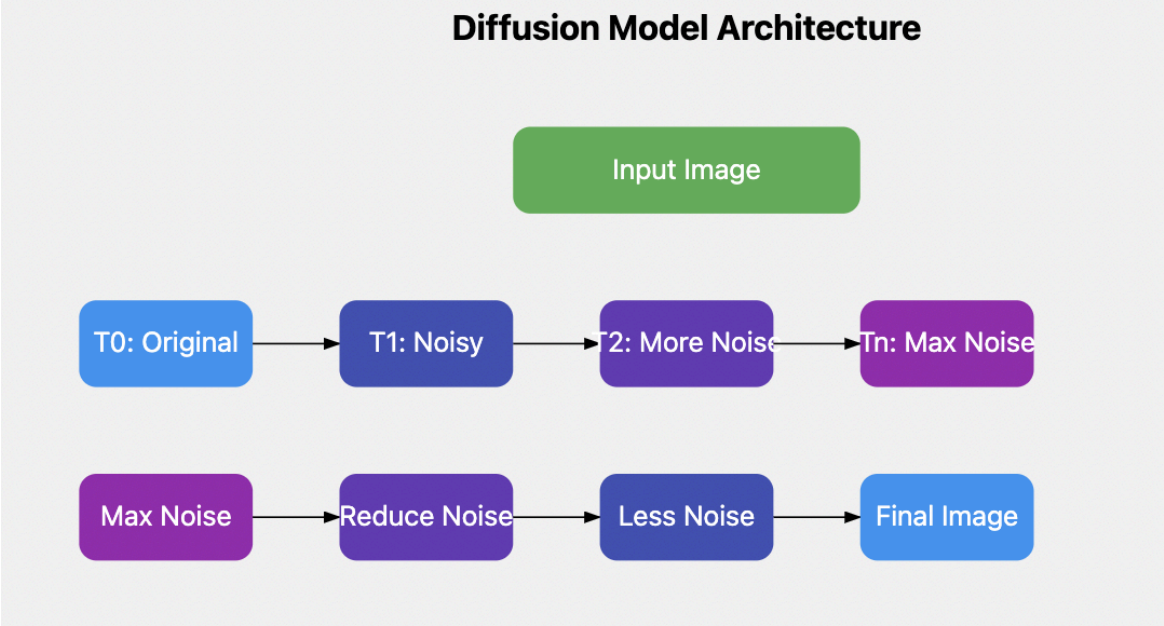
- **Dataset Loading**: A custom dataset representing images of three ethnic groups—Europeans, Indians, and Orientals—was loaded for training.
- **Image Preprocessing**: Images were resized and normalized to ensure compatibility with the model and to facilitate faster training.
- **Data Augmentation**: Techniques such as horizontal flipping, slight rotations, and color jittering were applied to increase the diversity of training samples, reducing overfitting risks.

2. Model Architecture Design

- **UNet Framework**: Designed a hierarchical encoder-decoder network with skip connections, enabling the model to effectively capture both global and local features of the data.
- **Conditional Embeddings**: Incorporated learnable embeddings based on class labels to condition the model output on the input class, enabling targeted image generation.
- **Self-Attention Blocks**: Added self-attention layers within the network, allowing the model to focus on spatially important regions, thus enhancing generated image quality.

3. Diffusion Process Implementation

- **Noise Scheduling**: Defined a noise addition-removal process that gradually transforms the data distribution into pure Gaussian noise during forward diffusion and reconstructs data from noise during reverse diffusion.
- **Forward and Reverse Diffusion Functions**: Implemented mathematical operations to facilitate the iterative addition of noise and the learning-based noise removal process.
- **Learning Objective**: Minimized the Mean Squared Error (MSE) loss between the predicted noise and actual noise to optimize the reverse diffusion process.



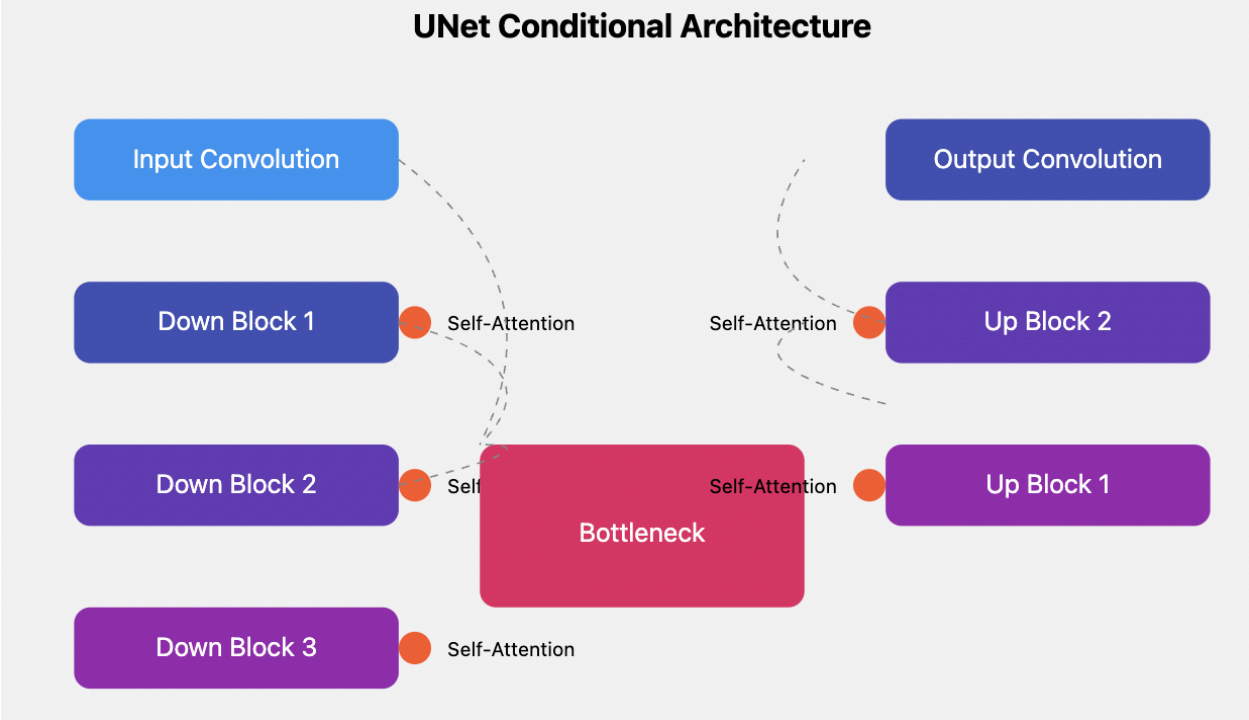
4. Training Optimization

- **Gradient Checkpointing**: Implemented to reduce memory consumption by recomputing certain intermediate activations during backpropagation. This allowed for training larger models on limited hardware resources.
- **EMA (Exponential Moving Average)**: Maintained a smoothed version of model weights during training for improved stability and generalization in the generation phase.
- **Gradient Clipping**: Set a maximum norm for gradients to prevent training instability caused by exploding gradients.
- **Learning Rate Scheduler**: Used a cosine annealing schedule to gradually decrease the learning rate, ensuring steady convergence and avoiding oscillations in loss.

5. Model Training

- **Batch Training**: Processed batches of data with noise added to simulate the forward diffusion process.
- **Conditional Sampling**: Ensured that the noise predictions and resulting reconstructions were conditioned on the appropriate class embeddings, thus generating images aligned with the intended class labels.

- **Validation and Monitoring**: Periodically validated the model on a separate dataset and monitored training metrics to ensure stable and effective learning.



6. Performance Evaluation

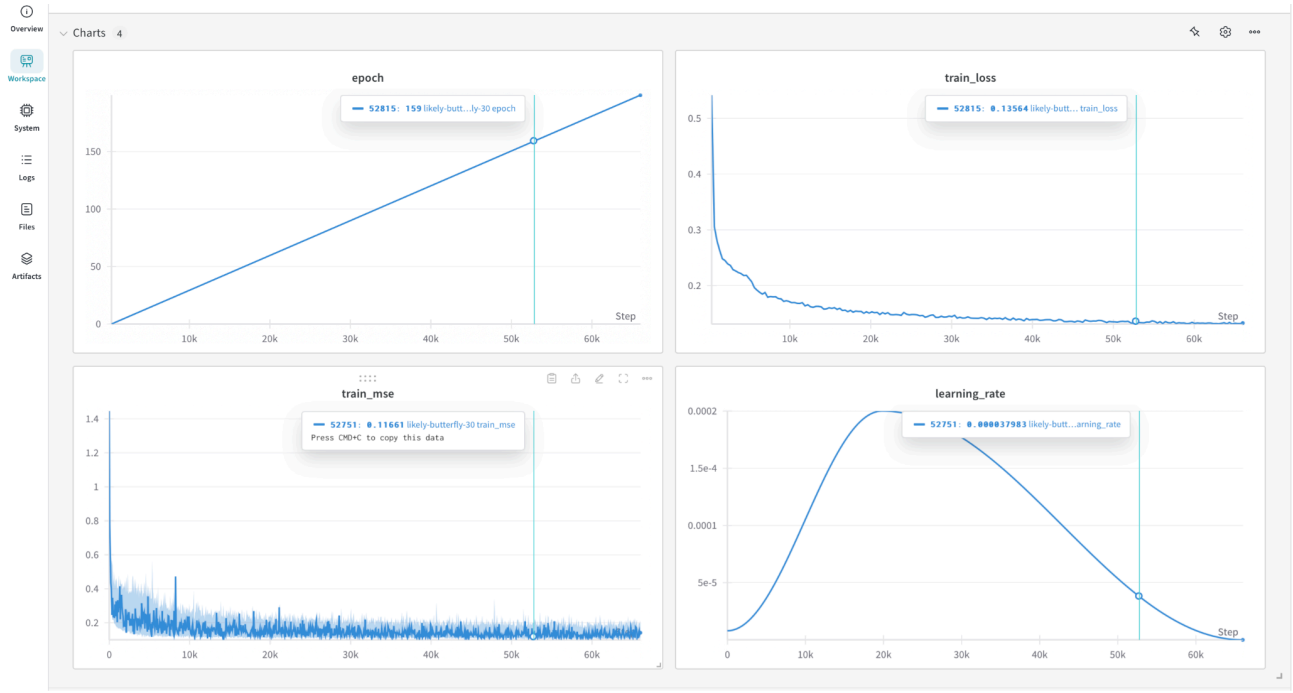
- **Loss Evaluation**: Tracked the reconstruction loss (MSE) across epochs to assess the model's performance in denoising.
- **Qualitative Analysis**: Visualized the generated images to evaluate the realism and fidelity of the conditional generation.
- **Comparison Metrics**: Compared outputs for different class labels to verify that the model generated distinguishable and representative images for each class.

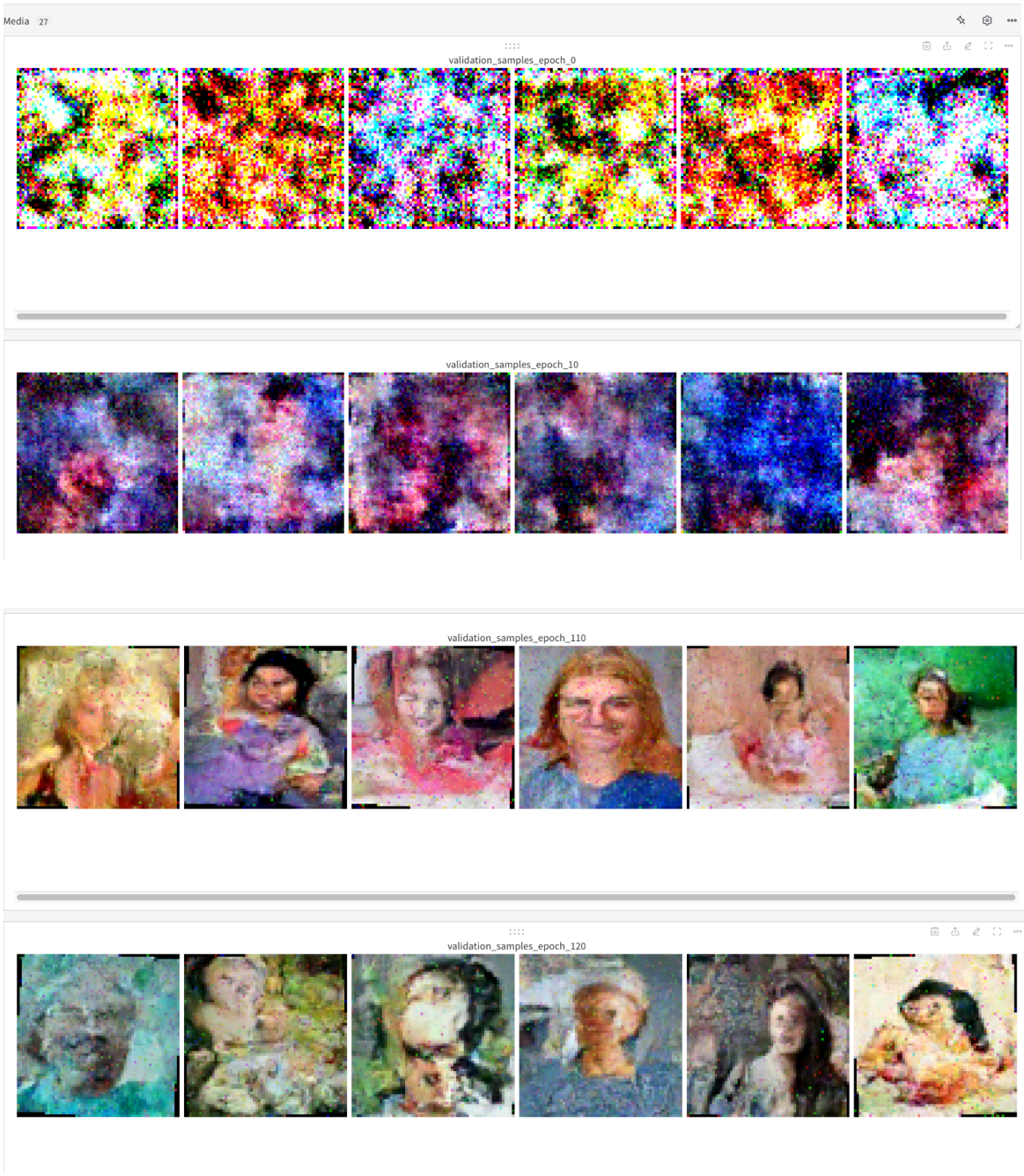
7. Visualization

- **Generated Images**: Visualized the diffusion process, showing the progressive denoising and the final generated image for each class.

- **Attention Maps**: Examined attention heatmaps from self-attention layers to understand which features the model focused on during generation.

- **Training Progress**: Graphed loss curves to demonstrate the model's training dynamics.





8. Output Generation

- **Synthetic Image Generation**: Successfully generated images conditioned on the class labels ("European," "Indian," "Oriental").

- **Versatility**: Demonstrated the model's ability to generate diverse images within each class.

4. Image Collection Methodology

The image collection methodology plays a crucial role in ensuring the quality and diversity of data used for training a Conditional Diffusion Model. For this project, the images represent three distinct ethnic groups—Europeans, Indians, and Orientals. Below is a step-by-step breakdown of the approach used to collect, preprocess, and organize the dataset.

1. Data Source Identification

- **Primary Sources**:
  Images were sourced from publicly available datasets and online repositories, such as:
    - Open-source platforms like Kaggle and UTKFace dataset

2. Ethnicity Labeling

- **Class Definitions**:
  The dataset was categorized into three classes:
    - European
    - Indian
    - Oriental
- **Manual Annotation**:
  In cases where ethnicity labels were not explicitly provided, manual labeling was performed using expert consultation or secondary datasets with metadata to cross-verify classifications.

- **Automated Classification**:
  Pretrained ethnicity recognition models were used to pre-classify images, followed by manual validation to ensure accuracy.

3. Image Filtering

- **Quality Check**:
  Collected images were inspected for:
    - Resolution (ensuring consistency, such as a minimum size of 128x128 pixels).
    - Clarity (removal of blurry or overexposed images).
    - Framing (ensuring proper focus on facial features without obstructions).

- **Ethnic Ambiguity**:
  Images with ambiguous ethnic features or inconsistent annotations were excluded to avoid misrepresentation during model training.

4. Diversity Maximization

- **Intra-Class Variation**:
  Ensured that within each ethnic group, there was significant variation in:
    - Gender
    - Age groups (children, adults, elderly)
    - Expressions, accessories (glasses, hats), and hairstyles
- **Inter-Class Balance**:
  Collected an approximately equal number of images for each class to prevent class imbalance during training.

5. Preprocessing

- **Resizing**:
  All images were resized to a uniform dimension of 128x128 pixels to match the input size expected by the UNet model.
- **Normalization**:
  Pixel values were normalized to the range [0, 1] or [-1, 1] to facilitate stable training.
- **Augmentation**:
  Data augmentation techniques were applied to increase dataset diversity:
    - Slight rotations (e.g., ±10 degrees)
    - Random cropping and scaling
    - Brightness and contrast adjustments

6. Ethical Considerations

- **Consent Awareness**:
  Ensured that the dataset used was ethically sourced, adhering to guidelines that protect individual privacy and consent for use in machine learning research.
- **Avoiding Bias**:
  Took steps to include diverse samples within each ethnic group, avoiding overrepresentation or underrepresentation of any specific demographic.

7. Dataset Organization

- **Class-Based Directories**:
  Images were organized into directories based on class labels

8. Validation Dataset Creation

- **Train-Test Split**:
  Divided the dataset into training and validation sets, typically using an 80-20 split, ensuring all classes were equally represented in both sets.
- **Cross-Class Validation**:
  Conducted preliminary checks by generating confusion matrices to ensure the integrity of the labels and the model's ability to distinguish among classes during training.

5. Proposed Method: Conditional Diffusion Model

The Conditional Diffusion Model (CDM) is an advanced generative model that leverages the power of diffusion processes to synthesize high-quality images while incorporating additional conditional information. In this project, the CDM is used to generate realistic images conditioned on class labels, such as ethnicity categories (European, Indian, and Oriental). Below is a comprehensive explanation of the proposed methodology.

1. Overview of Diffusion Models

Diffusion models are generative models based on the principles of **probabilistic modeling** and **denoising autoencoders**. They generate data by reversing a diffusion process:

- **Forward Process**: Gradually adds noise to the data, transforming it into pure noise.
- **Reverse Process**: Iteratively removes noise, reconstructs the original data or generates new samples from noise.

2. Conditional Diffusion Model

A **Conditional Diffusion Model** is an extension of standard diffusion models that incorporates additional conditioning information (e.g., class labels or textual descriptions) to guide the generative process. This ensures the generated data aligns with the specified conditions.

3. Model Architecture

The Conditional Diffusion Model primarily involves the following components:

a. UNet-Based Architecture

- The core architecture for denoising is based on a UNet, which is particularly effective for image-to-image tasks:
    - Encoder: Extracts hierarchical features by progressively down-sampling the input image.
    - Bottleneck: Captures the latent representation of noisy input data.
    - Decoder: Reconstructs the denoised image through up-sampling layers.
    - Skip Connections: Directly pass fine-grained details from the encoder to the decoder, preserving spatial information.

b. Conditional Embeddings

- Conditional inputs (class labels) are incorporated into the model through embeddings:
    - Class labels are encoded into learnable embeddings.

    - These embeddings are added to the intermediate layers of the UNet to guide the noise-removal process.

c. Time Step Embeddings

- Time steps from the diffusion process are embedded into the model to provide a sense of progression in the denoising process.

4. Forward Diffusion Process

The forward diffusion process gradually adds Gaussian noise to an input image x_0, resulting in a noisy image x_t at each time step t. The noise-adding process is defined as:

$$x_t = \sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t} \epsilon$$

where:

- α_t controls the variance schedule.
- ϵ is Gaussian noise sampled from $\mathcal{N}(0, I)$.

Over several steps, x_t approaches pure noise.

Over several steps, x_t approaches pure noise.

5. Reverse Diffusion Process

The reverse process reconstructs the original data or generates new data starting from noise. At each time step t, the model predicts the noise component ϵ using the denoising UNet. The reverse process can be expressed as:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \sqrt{1 - \alpha_t} \epsilon_\theta(x_t, t, y) \right) + \sigma_t z$$

where:

- $\epsilon_\theta(x_t, t, y)$ is the predicted noise for noisy input x_t, time t, and condition y.
- z is additional noise sampled from $\mathcal{N}(0, I)$.
- σ_t controls the noise variance at each step.

6. Training Objective

The training objective is to minimize the mean squared error (MSE) between the predicted noise and the true noise added during the forward process:

$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon,y}\left[\|\epsilon - \epsilon_\theta(x_t, t, y)\|^2\right]$$

where:

- x_0: Original image.
- x_t: Noisy image at time t.
- y: Conditional label.
- ϵ: True noise added during the forward process.

7. Sampling from the Model

During sampling:

1. Start with random noise x_T.
2. Iteratively apply the reverse diffusion process, guided by the conditional label y.
3. Generate the desired image x_0 corresponding to the specified class label.

8. Conditioning Mechanism

- **Class Embedding Injection**: The class embeddings are injected into the intermediate layers of the UNet, ensuring that the denoising process is influenced by the specified condition.
- **Cross-Attention**: In advanced setups, cross-attention mechanisms are used to further align the generated image with the conditioning information.

9. Performance Metrics

The model's performance is evaluated using:

- **Quality of Generated Images**:
    - Perceptual Quality: Visual inspection of generated samples.
    - Inception Score (IS) and Fréchet Inception Distance (FID): Quantitative metrics for image quality.
- **Class Consistency**:
    - Classification Accuracy: Ensures generated images align with the specified conditions.
- **Diversity**:
    - Measures intra-class variability of generated samples.

6. Implementation Details

Technical Challenges and Solutions

1. Memory Management
    - Implemented gradient checkpointing
    - Used mixed-precision training
    - Dynamic CUDA memory allocation
2. Training Stability
    - Exponential Moving Average (EMA) for model weights
    - Adaptive learning rate with OneCycleLR scheduler
    - Gradient clipping
3. Conditional Generation
    - Class embedding technique
    - Classifier-free guidance (CFG) for improved generation quality

Training Hyperparameters

- Learning Rate: 0.0002
- Batch Size: 10
- Image Size: 64x64
- Noise Steps: 500
- Epochs: 200

7. Program Documentation

Here is the link for the complete implementation of the program:
TermProjectDPL.ipynb

Diffusion Model: Theory and Implementation Analysis

1. Core Diffusion Theory Components

i. Forward Diffusion Process:

At its core, the diffusion process gradually adds noise to images over multiple timesteps. The `noise_images` method implements this:

```python
def noise_images(self, x, t):
    sqrt_alpha_hat = torch.sqrt(self.alpha_hat[t])[:, None, None, None]
    sqrt_one_minus_alpha_hat = torch.sqrt(1 - self.alpha_hat[t])[:, None, None, None]
    Ɛ = torch.randn_like(x, requires_grad=True)
    return sqrt_alpha_hat * x + sqrt_one_minus_alpha_hat * Ɛ, Ɛ
```

This implements the forward process where:

- x: Original image
- t: Timestep
- α_t: Noise schedule parameter (1 - β_t)
- ε: Random noise

The equation being implemented is:

$q(x_t|x_0) = \sqrt{(\hat{\alpha}\square)}x_0 + \sqrt{(1-\hat{\alpha}\square)}\varepsilon$

ii. Noise Schedule:

The noise schedule controls how quickly the image gets noised:

```python
def prepare_noise_schedule(self):
    return torch.linspace(self.beta_start, self.beta_end, self.noise_steps)
```

Key components:

- Linear schedule from β_1 to $\beta\square$
- β_start = 1e-4 (minimal initial noise)
- β_end = 0.02 (significant final noise)
- noise_steps = 1000 (default)

2. UNet Architecture for Noise Prediction

i. **Double Convolution Block**

```python
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels, mid_channels=None, residual=False):
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1, bias=False),
            nn.GroupNorm(1, mid_channels),
```

```
            nn.GELU(),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.GroupNorm(1, out_channels),
        )
```

Features:

- Double 3x3 convolutions
- Group normalization for stability
- GELU activation
- Optional residual connections

ii. Self-Attention Mechanism

```python
class SelfAttention(nn.Module):
    def __init__(self, channels):
        self.mha = nn.MultiheadAttention(channels, 4, batch_first=True)
        self.ln = nn.LayerNorm([channels])
        self.ff_self = nn.Sequential(
            nn.LayerNorm([channels]),
            nn.Linear(channels, channels),
            nn.GELU(),
            nn.Linear(channels, channels),
        )
```

```

Purpose:

- Captures long-range dependencies

- 4-head attention mechanism

- Layer normalization for stable training

- Feed-forward network for feature transformation

**3. Hierarchical Feature Processing**

**i. Downsampling Path:**

```python
class Down(nn.Module):
    def __init__(self, in_channels, out_channels, emb_dim=256):
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, in_channels, residual=True),
            DoubleConv(in_channels, out_channels),
        )
```

Function:

- Reduces spatial dimensions

- Increases feature channels

- Incorporates time embeddings

- Uses residual connections
```

ii. Upsampling Path:

```python
class Up(nn.Module):
    def __init__(self, in_channels, out_channels, emb_dim=256):
        self.up = nn.Upsample(scale_factor=2, mode="bilinear", align_corners=True)
        self.conv = nn.Sequential(
            DoubleConv(in_channels, in_channels, residual=True),
            DoubleConv(in_channels, out_channels, in_channels // 2),
        )
```

Function:

- Increases spatial dimensions
- Concatenates skip connections
- Processes combined features
- Incorporates time information

4. Time Encoding and Conditioning

i. Positional Encoding

```python
def pos_encoding(self, t, channels):
    inv_freq = 1.0 / (
        10000 ** (torch.arange(0, channels, 2, device=self.device).float() / channels)
```

```
        )
        pos_enc_a = torch.sin(t.repeat(1, channels // 2) * inv_freq)
        pos_enc_b = torch.cos(t.repeat(1, channels // 2) * inv_freq)
        pos_enc = torch.cat([pos_enc_a, pos_enc_b], dim=-1)
        return pos_enc
```

Purpose:

- Converts timesteps to high-dimensional embeddings
- Uses sinusoidal encoding
- Enables the model to understand time position

5. Training Process

i. Loss Calculation:

The model predicts the noise added during the forward process:

```python
predicted_noise = self.model(x_t, t, labels)
loss = self.mse(noise, predicted_noise)
```

ii. Sampling Process:

The reverse diffusion process removes noise step by step to generate images from random noise. It is implemented in the `sample` function:

```
def sample(self, use_ema, labels, cfg_scale=3):
    x = torch.randn((n, self.c_in, self.img_size, self.img_size)).to(self.device)
```

```
    for i in progress_bar(reversed(range(1, self.noise_steps)), total=self.noise_steps - 1):
        t = (torch.ones(n) * i).long().to(self.device)
        predicted_noise = model(x, t, labels)

        if cfg_scale > 0:
            uncond_predicted_noise = model(x, t, None)
            predicted_noise = torch.lerp(uncond_predicted_noise, predicted_noise, cfg_scale)

        alpha = self.alpha[t][:, None, None, None]
        alpha_hat = self.alpha_hat[t][:, None, None, None]
        beta = self.beta[t][:, None, None, None]

        if i > 1:
            noise = torch.randn_like(x)
        else:
            noise = torch.zeros_like(x)

        x = 1 / torch.sqrt(alpha) * (x - ((1 - alpha) / torch.sqrt(1 - alpha_hat)) * predicted_noise) + torch.sqrt(beta) * noise
```

Equation Represented:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sqrt{\beta_t} z$$

Where:

- $\epsilon_\theta(x_t, t)$: Predicted noise by the UNet model.
- z: Random Gaussian noise added unless $t = 1$.

6. Training Optimizations

i. EMA (Exponential Moving Average):

```python
class EMA:
    def update_model_average(self, ma_model, current_model):
        for current_params, ma_params in zip(current_model.parameters(), ma_model.parameters()):
            old_weight, up_weight = ma_params.data, current_params.data
            ma_params.data = self.update_average(old_weight, up_weight)
```

Purpose:

- Stabilizes training
- Improves generation quality
- Maintains running average of weights

ii. Mixed Precision Training:

```python
```

```
with torch.cuda.amp.autocast(enabled=self.fp16):
    t = self.sample_timesteps(images.shape[0]).to(self.device)
    x_t, noise = self.noise_images(images, t)
```

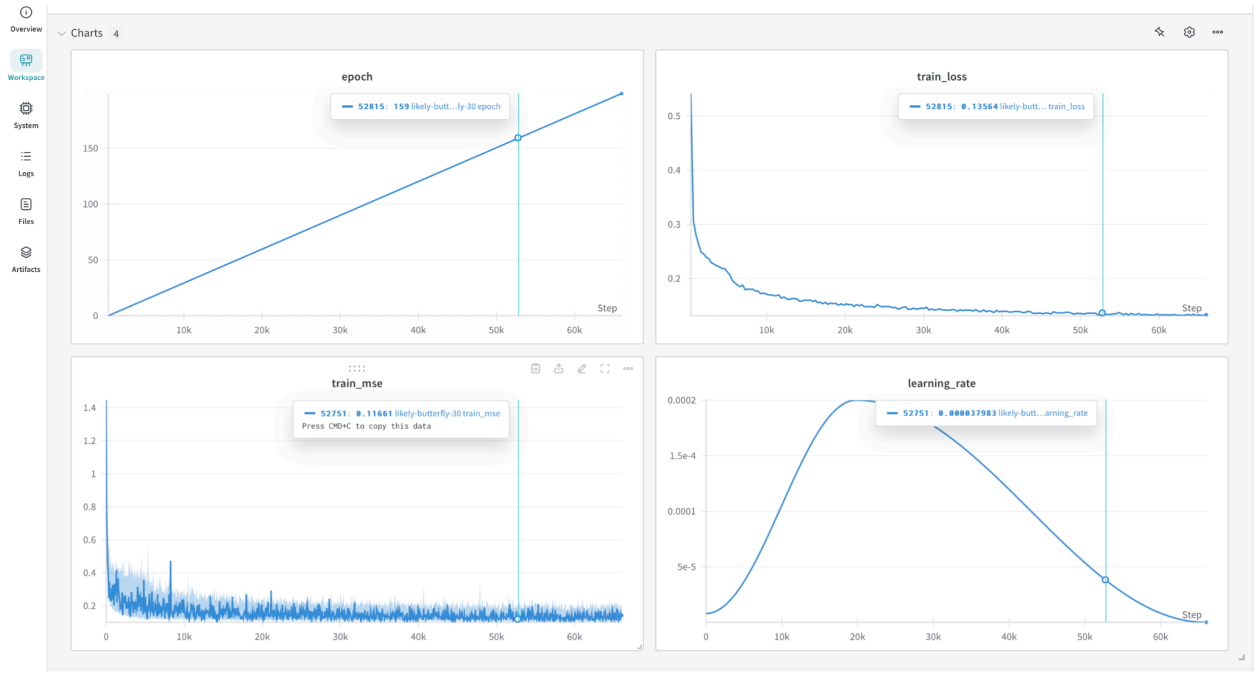
Benefits:

- Reduced memory usage
- Faster training
- Maintained precision where needed.

8. Results and Visualization

Training Metrics:

- Tracked training loss and learning rate using Weights & Biases (wandb)
- Validation performed every 10 epochs



Generation Scenarios:

1. Single Ethnic Group Generations
2. Binary Ethnic Mixing
3. Tri-group Mixed Generation

Generated Image Examples:

Performance Observations

- Successful generation of diverse facial representations
- Maintained image quality across different ethnic group combinations
- Demonstrated flexibility in conditional image synthesis

9. Conclusion

This project successfully demonstrated the potential of conditional diffusion models in generating diverse facial imagery. By implementing advanced deep learning techniques, we created a flexible generative framework capable of producing high-quality, ethnically diverse facial images.

Future Work:

- Increase model resolution
- Enhance diversity and representation
- Explore more sophisticated conditioning techniques

Limitations:

- Computational intensity
- Potential bias in training data
- Limited image resolution

Key Takeaways:

- Diffusion models offer powerful generative capabilities
- Conditional generation enables targeted image synthesis
- Careful data preprocessing is crucial for model performance

Project Tools and Technologies:

- PyTorch
- Weights & Biases
- CUDA
- Matplotlib
- Torchvision