Google Summer of Code 2025

Proposal

Learning Transformations in Latent Space Using Variational Autoencoders (VAE)

(2j)

Organization: Machine Learning For Science(ML4sci)

Name: Suvit Kumar

Email: suvitkumar03@gmail.com

GitHub Profile: SuvitKumar003

LinkedIn Profile: linkedIn/suvitkumar03

University: Thapar Institute of Engineering and Technology

Country: India

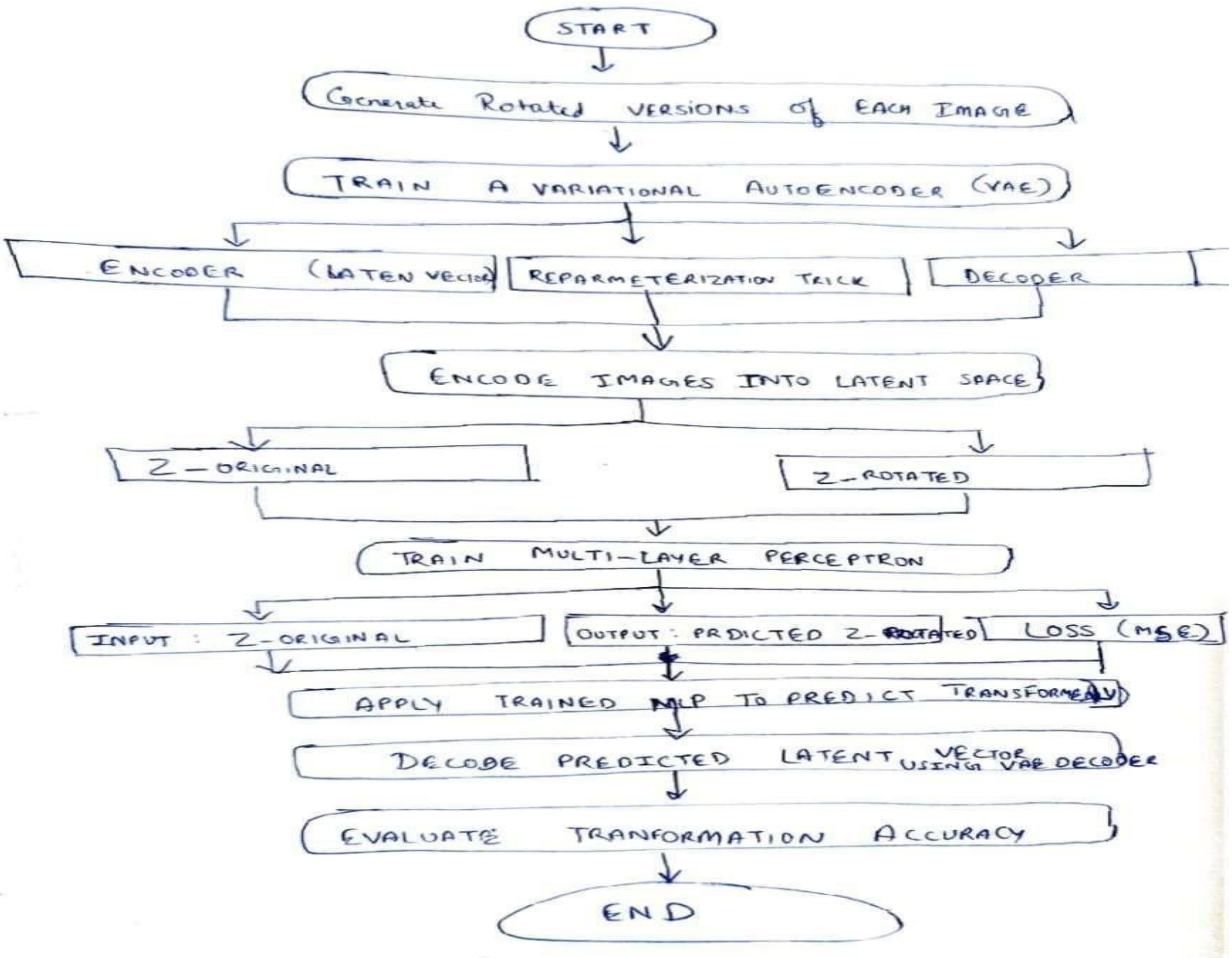
GitHub Repository

You can find the complete implementation, codebase, and relevant resources for this project in my GitHub repository: [GitHub Repository: ML4sci_Projects_Repo_Suvit_Kumar]

This repository contains:

- The implementation of Variational Autoencoders (VAEs) for latent space learning.
- The trained models and transformation learning techniques.
- Detailed documentation of the experiments, challenges, and future scope.

Project



About Me

I'm Suvit Kumar, a third-year Computer Engineering student at Thapar Institute of Engineering & Technology (TIET), with a CGPA of 8.44. I am passionate about open-source development, machine learning, and deep learning, and I enjoy working on projects that solve real-world challenges.

Technical Skills

- Programming Languages: Python, C++
- Frameworks: TensorFlow, PyTorch, OpenCV
- Specialized Knowledge: Advanced machine learning, financial technology, data-driven decisionmaking

Projects

- Credit Card Fraud Detection: Built an anomaly detection model using autoencoders to identify fraudulent transactions.
- Finance Capstone Project: Conducted predictive modeling and AI-driven risk assessments for financial trends.
- Recommender System: Developed a personalized product recommendation system.
- Book Script Generation & Sentiment Analysis: Combined NLP and sentiment analysis to generate summaries and analyze reader sentiment.

Experience

- Worked with large datasets, performed exploratory data analysis (EDA), and implemented feature engineering techniques.
- Participated in Kaggle competitions to refine skills in handling real-world datasets.
- Active open-source contributor with a focus on collaborative development.

Hobbies & Interests

- Reading technical blogs and staying updated on AI advancements.
- Participating in coding challenges and hackathons to enhance problem-solving skills.
- Running and experimenting with AI models in my free time.
- Networking on LinkedIn to build industry connections.

Availability for GSoC 2025

I am fully available throughout the GSoC timeline:

1. Community Bonding Period (May - Early June): 20–25 hours/week while engaging with mentors and setting up the project environment.
2. Coding Period (Mid-June - Mid-August): 30–40 hours/week during summer break for coding, testing, and refining contributions.
3. Final Evaluation (Late August - September): Focus on documentation, refinements, and final testing.

I am flexible with working hours across time zones and committed to regular communication and steady progress.

Common Task: Electron/Photon Classification

Implementation Highlights

- Dataset: 32×32 matrices (hit energy & time) provided in HDF5 format.
- Preprocessing: Used h5py, NumPy, and PyTorch Data Loader for efficient handling of large datasets.
- Model Architecture: Modified ResNet-15 with Batch Normalization and Dropout layers to prevent overfitting.
- Training & Evaluation:
    - Split dataset into 80% training and 20% validation sets.
    - Trained using CrossEntropyLoss and Adam optimizer.
    - Applied hyperparameter tuning for optimal performance.
    - Validated generalization ability on an independent test set.

This task showcased my ability to preprocess complex datasets, design deep learning models in PyTorch, and optimize architectures for high accuracy—skills essential for GSoC contributions.

Code Images:

```python
import torch
import h5py
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader

class ParticleDataset(Dataset):
    def __init__(self, data_files, transform=None):
        self.data_files = data_files
        self.transform = transform
        self.data = []
        self.targets = []
        self.load_data()

    def load_data(self):
        for file_path in self.data_files:
            with h5py.File(file_path, "r") as f:
                dataset = f["X"][:]
                labels = f["y"][:]

                self.data.extend(dataset)
                self.targets.extend(labels)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sample = self.data[idx]
        target = int(self.targets[idx])

        sample = sample.unsqueeze(0) if len(sample.shape) == 2 else sample

        if self.transform:
            sample = self.transform(sample)

        return sample, torch.tensor(target, dtype=torch.long)
```

```python
dataset = ParticleDataset(data_files, transform=transform)

train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

print(f"Dataset Size: {len(dataset)}")
print(f"Training Samples: {len(train_dataset)}, Validation Samples: {len(val_dataset)}")
```

```
Dataset Size: 498000
Training Samples: 398400, Validation Samples: 99600
```

```python
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

class ResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = torch.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        return torch.relu(out)

# Define the ResNet15 model
class ResNet15(nn.Module):
    def __init__(self, num_classes=2):
        super(ResNet15, self).__init__()
        self.initial = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
        self.layer1 = ResNetBlock(64, 64)
        self.layer2 = ResNetBlock(64, 128, stride=2)
        self.layer3 = ResNetBlock(128, 256, stride=2)
        self.layer4 = ResNetBlock(256, 512, stride=2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.initial(x)
```

```python
# Training Loop
epochs = 10
for epoch in range(epochs):
    model.train()
    running_loss, correct, total = 0.0, 0, 0
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(dev
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

    train_acc = 100 * correct / total
    print(f"Epoch [{epoch+1}/{epochs}] | Loss: {running_los

# Save the Model
torch.save(model.state_dict(), "resnet15_cifar10.pth")
print("Model saved successfully!")
```

```
00%|██████████| 170M/170M [00:37<00:00, 4.55MB/s]
poch [1/10] | Loss: 1.2387 | Accuracy: 55.08%
poch [2/10] | Loss: 0.7746 | Accuracy: 72.66%
poch [3/10] | Loss: 0.5827 | Accuracy: 79.56%
poch [4/10] | Loss: 0.4379 | Accuracy: 84.73%
poch [5/10] | Loss: 0.3209 | Accuracy: 88.78%
poch [6/10] | Loss: 0.2209 | Accuracy: 92.21%
poch [7/10] | Loss: 0.1447 | Accuracy: 94.91%
poch [8/10] | Loss: 0.1070 | Accuracy: 96.32%
poch [9/10] | Loss: 0.0842 | Accuracy: 97.07%
poch [10/10] | Loss: 0.0656 | Accuracy: 97.72%
odel saved successfully!
```

Project Overview: Learning Transformations in Latent Space using Variational Autoencoders (VAE)

Introduction

Understanding transformations in image data is a critical aspect of deep learning, particularly in applications such as image recognition, medical imaging, and robotics. Traditional transformation models rely on explicit rules or pre-defined geometric functions, limiting their adaptability to unseen data.

This project aims to train a model to learn transformations directly in latent space using Variational Autoencoders (VAE) and Multi-Layer Perceptron (MLP). The core idea is to encode images into a compact latent space and then learn a function that models transformations, such as rotations, scaling, or warping, within this space.

To validate this approach, the model is trained on the MNIST dataset, where digits are rotated by fixed angles (0°, 30°, 60°, …, 330°). The transformation learning process involves the following steps:

- Encoding images into a latent space using a Variational Autoencoder (VAE).
- Training an MLP model to predict how latent representations change when the input image undergoes transformation (e.g., rotation).
- Decoding the predicted latent representation back to an image to verify if the learned transformation is accurate.

This approach enables a deep learning model to understand and generalize transformations in an unsupervised manner, paving the way for advancements in generative modeling and representation learning.

Key Components and Implementation Details

1. Variational Autoencoder (VAE) for Image Encoding

A Variational Autoencoder (VAE) is employed to encode images into a lower-dimensional latent representation, ensuring that important structural information is preserved while reducing noise.

Architecture Details

The VAE consists of three primary components:

- Encoder
    - A convolutional neural network (CNN) that compresses input images into a latent representation.
    - The encoder outputs mean (μ) and variance (σ^2) parameters, which define a probability distribution for the latent vector.
- Reparameterization Trick
    - Since sampling directly from the latent distribution is nondifferentiable, the reparameterization trick is applied:

$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0,1)$$

    - This ensures the model remains trainable using standard backpropagation.
- Decoder o A deconvolutional network that reconstructs the original image from the sampled latent representation.
    - The decoder ensures that the generated image closely resembles the input image.

Loss Function

The VAE loss function consists of two terms:

- Reconstruction Loss (Mean Squared Error / Binary Cross-Entropy):
    - Measures how well the reconstructed image matches the input image.
- KL Divergence Loss:
    - Ensures the latent space follows a standard normal distribution, preventing overfitting and improving generalization.

Results and Observations

Achievements:

- The VAE successfully encodes and reconstructs images from the MNIST dataset.
- The latent representations capture essential digit characteristics while removing noise.

Challenges Faced:

- Choosing an optimal latent dimension was crucial. A too-small latent space led to poor reconstructions, while too-large latent dimensions resulted in excessive noise.
- Balancing KL loss was challenging. A high KL weight led to overly smooth representations, while a low KL weight caused poor generalization.

2. Learning Transformations in Latent Space using MLP

Once images are encoded into latent vectors, the next step is to train a Multi-Layer Perceptron (MLP) to model transformations directly in latent space.

Transformation Learning Objective

The MLP learns a mapping function that predicts the latent representation of a transformed image given the original image's latent vector:

$$MLP(z_{\text{original}}) = z_{\text{rotated}}$$

where:

- Z_{original} is the latent representation of the original image.
- Z_{rotated} is the latent representation of the same image after a specific transformation (rotation).

MLP Architecture

- Input Layer: Accepts the original latent vector.
- Hidden Layers:
  - Three fully connected layers with ReLU activation.
  - Provides sufficient complexity to model transformations effectively.
- Output Layer: Outputs the transformed latent vector.
- Loss Function:
  - Mean Squared Error (MSE) Loss is used to minimize the difference between predicted and actual latent vectors.

Results and Observations

Achievements:

- The MLP successfully learns small rotation transformations (e.g., 0° to 30°).
- Latent vector predictions align well with actual latent representations for moderate rotations.

Challenges Faced:

- Poor performance for large rotations (>90°). The MLP struggles to capture complex nonlinear transformations.
- Linear transformations may not be sufficient. Rotations are inherently nonlinear, and a more advanced architecture (e.g., Transformers) might be needed.

3. Custom Dataset Creation: Rotated MNIST

To train the transformation model effectively, a custom dataset was created by rotating MNIST images in fixed intervals of 30° (0°, 30°, 60°, …, 330°).

Achievements:

- Ensured that each MNIST digit had multiple rotated versions, creating a diverse dataset.
- Efficient data loading was implemented using PyTorch's DataLoader.

Challenges Faced:

- Edge cropping issue: Rotating MNIST images within a fixed 28×28 frame caused information loss.
- Digit confusion: Some digits (e.g., "1" and "8") were harder to recognize at extreme angles.

4. Testing and Evaluation

After training the VAE and MLP, the evaluation phase involved:

1. Encoding an original image into latent space.
2. Using the MLP to predict the rotated latent representation.
3. Decoding the predicted latent vector back into an image using the VAE decoder.

Achievements:

- Successfully reconstructed rotated images for small transformations.
- Latent interpolations showed smooth transitions between transformations.

Challenges Faced:

- Large rotations resulted in blurry, incorrect reconstructions due to the limited expressiveness of MLP.
- The decoder sometimes lost fine details, affecting digit clarity.

Limitations & Future Directions

While the project demonstrates the feasibility of learning transformations in latent space, several limitations remain:

1. Latent Space Representation Needs Improvement
   - The VAE struggles with rotation invariance.
   - Possible Solution: Implement Rotation-Invariant VAEs or Group Equivariant CNNs (GCNNs).
2. MLP May Not Be Expressive Enough
   - Complex transformations like large-angle rotations require a more powerful model.
   - Possible Solution: Use Transformer Networks or RNN-based latent modeling.
3. Dataset Limitations
   - MNIST is too simplistic; real-world applications need diverse datasets.
   - Possible Solution: Extend the model to CIFAR-10 or medical imaging datasets.

Conclusion

This project successfully explores how image transformations can be learned in latent space using a combination of Variational Autoencoders (VAE) and Multi-Layer Perceptrons (MLP). While the approach works for small rotations, further improvements are needed for complex transformations. Future work could involve more advanced architectures, diverse datasets, and additional transformations beyond rotations.

This research has implications for robotics, autonomous systems, and generative modeling, where transformation learning plays a critical role in real-world applications.

BELOW ARE MY IMPLEMENTATION IMAGES:

```python
from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor, functional as F
import torch
from torch.utils.data import Dataset

class RotatedMNIST(Dataset):
    def __init__(self, root="./data", train=True, digits=(1, 2), angles=None):
        self.mnist = MNIST(root=root, train=train, download=True, transform=ToTensor())
        self.indices = [i for i, (_, label) in enumerate(self.mnist) if label in digits]
        self.angles = angles if angles else list(range(0, 360, 30))  # Rotations in steps of 30 degre

    def __len__(self):
        return len(self.indices) * len(self.angles)

    def __getitem__(self, idx):
        img_idx = self.indices[idx // len(self.angles)]
        angle = self.angles[idx % len(self.angles)]
        img, label = self.mnist[img_idx]

        img_rotated = F.rotate(img, angle)
        return img, img_rotated, torch.tensor(angle / 360.0)  # Normalize angle

# Define dataset and DataLoader
train_dataset = RotatedMNIST(root="./data", train=True)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
```

4.7s Python

```python
        )

    def encode(self, x):
        h = self.encoder(x)
        return self.fc_mu(h), self.fc_var(h)

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

    def decode(self, z):
        return self.decoder(z)

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

def vae_loss(recon_x, x, mu, logvar):
    BCE = F.binary_cross_entropy(recon_x, x, reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = VAE(latent_dim=32).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)


print("Starting VAE Training...")
for epoch in range(20):
    model.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()

        recon_batch, mu, logvar = model(data)
        loss = vae_loss(recon_batch, data, mu, logvar)

        loss.backward()
        train_loss += loss.item()
        optimizer.step()
```

```python
model.load_state_dict(torch.load("vae_mnist.pth"))
model.train()
```

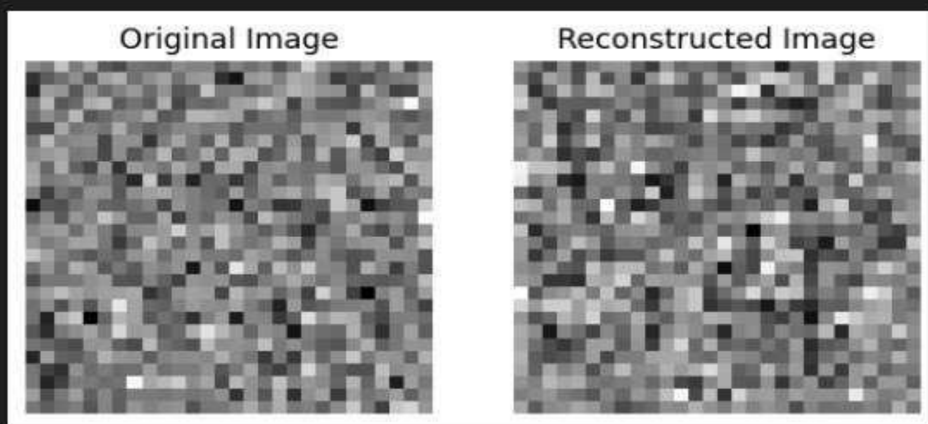
✓ 0.0s Python

```
VAE(
  (encoder): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU()
    (4): Flatten(start_dim=1, end_dim=-1)
    (5): Linear(in_features=3136, out_features=256, bias=True)
    (6): ReLU()
  )
  (fc_mu): Linear(in_features=256, out_features=32, bias=True)
  (fc_var): Linear(in_features=256, out_features=32, bias=True)
  (decoder): Sequential(
    (0): Linear(in_features=32, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=3136, bias=True)
    (3): ReLU()
    (4): Unflatten(dim=1, unflattened_size=(64, 7, 7))
    (5): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1)
    (6): ReLU()
    (7): ConvTranspose2d(32, 1, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1)
    (8): Sigmoid()
  )
)
```

```python
for epoch in range(21, 31):
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()

        recon_batch, mu, logvar = model(data)
        loss = vae_loss(recon_batch, data, mu, logvar)

        loss.backward()
        train_loss += loss.item()
        optimizer.step()

    print(f'Epoch {epoch}, Loss: {train_loss/len(train_loader.dataset):.4f}')
```

```
Epoch 21, Loss: 97.4159
Epoch 22, Loss: 97.3002
Epoch 23, Loss: 97.1484
Epoch 24, Loss: 97.0593
Epoch 25, Loss: 96.8989
Epoch 26, Loss: 96.7997
Epoch 27, Loss: 96.7098
Epoch 28, Loss: 96.6413
Epoch 29, Loss: 96.5300
Epoch 30, Loss: 96.4831
```

```python
import matplotlib.pyplot as plt

with torch.no_grad():
    recon_x, _, _ = vae(sample_input)

input_img = sample_input.cpu().squeeze().numpy()
recon_img = recon_x.cpu().squeeze().numpy()

fig, ax = plt.subplots(1, 2)
ax[0].imshow(input_img, cmap='gray')
ax[0].set_title("Original Image")
ax[0].axis("off")

ax[1].imshow(recon_img, cmap='gray')
ax[1].set_title("Reconstructed Image")
ax[1].axis("off")

plt.show()
```



Timeline

Community Bonding (May 20 – June 17, 2025)

Subtask	Start Date	End Date
Engage with mentors to refine objectives and finalize the implementation plan.	May 20, 2025	May 25, 2025
Explore existing research on latent space transformations, VAEs, and equivariant representations.	May 26, 2025	May 31, 2025
Set up the development environment (PyTorch/TensorFlow compatibility).	June 1, 2025	June 5, 2025
Prepare dataset pipelines for MNIST, Rotated MNIST, and explore generalization datasets.	June 6, 2025	June 11, 2025
Participate in community discussions to align expectations.	June 12, 2025	June 17, 2025

Phase 1: VAE Implementation & Latent Space Analysis (June 17 – July 15, 2025)

Subtask	Start Date	End Date
Implement VAE encoder-decoder architecture.	June 17, 2025	June 20, 2025
Train VAE on MNIST (optimize KL divergence + reconstruction loss).	June 21, 2025	June 24, 2025
Analyze latent space with t-SNE/PCA and visualize encodings.	June 25, 2025	June 28, 2025
Experiment with latent dimensionalities and activation functions.	June 29, 2025	July 1, 2025
Validate performance (Reconstruction Error, SSIM, PSNR).	July 2, 2025	July 5, 2025
Submit Mid-Term Evaluation Report.	July 6, 2025	July 15, 2025

Phase 2: Learning Transformations using MLP (July 16 – August 12, 2025)

Subtask	Start Date	End Date
Develop MLP model for latent space rotation prediction.	July 16, 2025	July 19, 2025
Train MLP on paired latent embeddings (original vs. rotated).	July 20, 2025	July 23, 2025
Experiment with loss functions (MSE, Cosine Similarity).	July 24, 2025	July 27, 2025
Subtask	Start Date	End Date
Evaluate MLP predictions vs. actual latent vectors.	July 28, 2025	July 31, 2025

Explore Transformer-based architectures for improvement.	August 1, 2025	August 4, 2025
Submit Progress Report (training curves, validation results).	August 5, 2025	August 12, 2025

Phase 3: Model Refinement & Testing (August 13 – September 2, 2025)

Subtask	Start Date	End Date
Train on additional transformations (scaling, translations).	August 13, 2025	August 16, 2025
Optimize computational efficiency (parameter reduction).	August 17, 2025	August 20, 2025
Conduct ablation studies on network components.	August 21, 2025	August 24, 2025
Test robustness on unseen rotations and synthetic data.	August 25, 2025	August 28, 2025
Finalize codebase, documentation, and scripts.	August 29, 2025	September 2, 2025

Final Submission & Evaluation (September 3 – September 9, 2025)

Subtask	Start Date	End Date
Submit final report, code, and experimental findings.	September 3, 2025	September 4, 2025
Conduct live demonstration of transformation learning.	September 5, 2025	September 6, 2025
Write blog post summarizing key takeaways and future work.	September 7, 2025	September 9, 2025

Why Me?

I believe I am an excellent fit for this project due to my strong technical background, hands-on experience with deep learning, and passion for solving complex AI problems. Below are key reasons why I am well-suited for this project:

1. Strong Foundation in Machine Learning & Deep Learning

- I have worked extensively with Variational Autoencoders (VAEs), CNNs, and deep generative models, which are crucial for this project.
- My experience with representation learning and transformation modeling gives me an edge in handling latent space manipulation.
- I have built projects like Credit Card Fraud Detection (Autoencoders), Recommender Systems, and NLP-based sentiment analysis, demonstrating my ability to work with complex datasets and neural architectures.

2. Proficiency in Relevant Technologies

- I am proficient in PyTorch and TensorFlow, with hands-on experience in training VAEs, MLPs, and deep neural networks.
- My experience in computer vision, data preprocessing, and handling large-scale datasets makes me confident in implementing and optimizing deep learning models.
- I have a strong grasp of linear algebra, probability, and optimization techniques, which are critical for generative modeling.

3. Research-Oriented Mindset & Analytical Thinking

- My deep curiosity about latent space transformations and equivariant learning aligns well with the project's research scope.
- I have experience in experimenting with different architectures, fine-tuning hyperparameters, and analyzing model performance to improve results.
- I am committed to documenting my work, sharing insights, and ensuring code clarity and maintainability for future research and development.

4. Problem-Solving & Commitment to GSoC

- I am a dedicated and self-motivated learner who enjoys tackling complex challenges in AI and deep learning.

- My ability to break down problems, experiment with different solutions, and analyze results ensures that I can successfully complete the project.
- I am eager to collaborate with mentors, receive constructive feedback, and contribute meaningful code to the open-source community.

With my passion, skills, and dedication, I am confident that I can successfully implement this project, contribute to the community, and make a lasting impact through GSoC 2025!