3D Pathfinding Neural Network - Setup Guide

Current Status

- Neural network architecture implemented (3DPathPlanning_nn.py)
- Synthetic dataset generation script (generate_synthetic.py)
- 1000 training samples generated
- Data structure analysis (synthetical_builds_training_samples.py)

Next Steps

1. Verify Your Dataset Structure

First, run the dataset inspector to make sure your 1000 samples are correctly formatted:

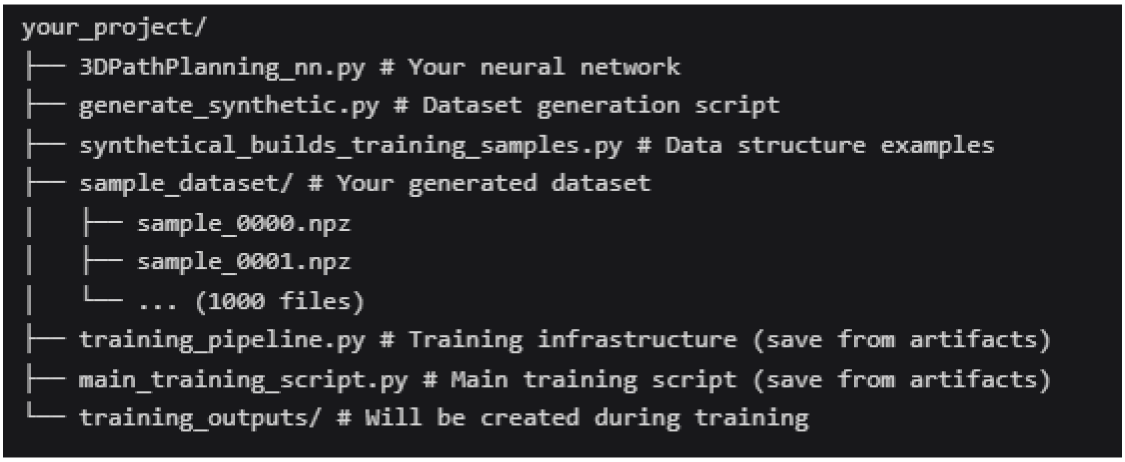
```
# Save this as dataset_inspector.py and run it

python dataset_inspector.py
```

Expected output should show:

- 1000 .npz files in /sample_dataset/
- Consistent shapes: (3, 32, 32, 32) for voxel data
- Valid path lengths and turn counts
- No corrupted files

2. File Organization

Ensure your project structure looks like this:



```
your_project/
├── 3DPathPlanning_nn.py # Your neural network
├── generate_synthetic.py # Dataset generation script
├── synthetical_builds_training_samples.py # Data structure examples
├── sample_dataset/ # Your generated dataset
│   ├── sample_0000.npz
│   ├── sample_0001.npz
│   └── ... (1000 files)
├── training_pipeline.py # Training infrastructure (save from artifacts)
├── main_training_script.py # Main training script (save from artifacts)
└── training_outputs/ # Will be created during training
```

3. Install Required Dependencies

```
pip install torch torchvision torchaudio
pip install numpy matplotlib tqdm
pip install pathlib argparse json datetime
```

4. Test Dataset Loading

Before training, verify everything works:

```
# Test dataset loading only
python main_training_script.py --test-only
# Quick dataset statistics
python main_training_script.py –stats
```

5. Start Training

Once everything is verified, start training:

```
# Basic training with default settings
python main_training_script.py
# Custom training with specific parameters
python main_training_script.py --epochs 50 --batch-size 4 --learning-rate 1e-3
# Resume from checkpoint
python main_training_script.py --resume ./training_outputs/model_epoch_20.pth
```

6. Monitor Training Progress

During training, you'll see:

- Real-time loss values (path loss, turn loss, collision loss)
- Validation metrics
- Automatic model saving every 10 epochs
- Training curves plot generation

Files created:

- ./training_outputs/best_model.pth - Best performing model
- ./training_outputs/final_model.pth - Final trained model
- training_history.json - Complete training metrics
- training_curves.png - Loss visualization plots

7. Expected Training Behavior

Initial Epochs (1-10):

- High path loss as model learns basic navigation
- High turn loss as model learns turn minimization
- Gradual decrease in all loss components

Mid Training (10-50):

- Path loss should stabilize at low values
- Turn loss should decrease as model learns efficient routing
- Validation loss should track training loss closely

Later Epochs (50+):

- Fine-tuning of turn optimization
- Potential overfitting if validation loss starts increasing

8. Troubleshooting Common Issues

CUDA out of memory:

```
python main_training_script.py --batch-size 2 --device cpu
```

No samples found:

- Check /sample_dataset/ directory exists
- Verify .npz files are present
- Re-run generate_synthetic.py if needed

Import error:

- Ensure all Python files are in the same directory
- Check file names exactly match the imports

Training loss not decreasing:

- Try lower learning rate: `--learning-rate 1e-5`
- Check dataset quality with `--stats`
- Verify input data shapes are correct

9. Evaluate Trained Model

After training, test your model:

```
python main_training_script.py --test-inference
```

This will:

- Load a trained model
- Run inference on a sample
- Compare generated path vs ground truth
- Show action sequences (FORWARD, LEFT, UP, etc.)

10. Next Development Steps

Once basic training works:

1. Improve Dataset:

- Generate more diverse environments
- Add different obstacle densities
- Include more complex path scenarios

2. Model Enhancements:

- Experiment with different loss weights
- Add attention mechanisms
- Try curriculum learning (start with simple environments)

3. Evaluation Metrics:

- Path success rate (reaches goal)
- Collision rate (zero collisions)
- Turn efficiency (turns/path_length)
- Path length optimality

4. Integration:

- Connect to Blender for visualization
- Export to ONNX for deployment
- Create real-time path planning interface

Quick Start Commands

```
# 1. Test everything
python main_training_script.py --test-only
# 2. Start training with small batch for testing
python main_training_script.py --epochs 5 --batch-size 2
# 3. If successful, run full training
python main_training_script.py --epochs 100 --batch-size 8
```

Success Indicators

Your training is working well if:

- Path loss decreases to < 0.1
- Turn loss decreases steadily
- No CUDA/memory errors
- Validation loss follows training loss
- Generated paths look reasonable in inference tests

Your model is ready for deployment when:

- Validation loss stabilizes
- Generated paths consistently reach goals
- Turn count is optimized (fewer unnecessary turns)
- Zero collision rate on test data