FuNet-C

---

1. Introduction

This project implements **FuNet-C**, a hybrid spectral–spatial neural network combining Graph Convolutional Networks (GCNs) on pixel-level spectral features and 2D Convolutional Neural Networks (2D CNNs) on spatial patches, for hyperspectral image classification. The main objectives are:

- **Spectral–Spatial Fusion**: Leverage both spectral similarity (via GCN) and local spatial context (via CNN).
- **Modular Pipeline**: Clean separation of data preparation (MATLAB scripts) and model training/inference (Python/TensorFlow).
- **Reproducibility**: Shared directory structure and scripts to regenerate training/testing splits, laplacian matrices, and trained network.

2. Directory Structure

```
FuNet-C/                     % Root Directory
├── 19920612_AVIRIS_IndianPine_Site3.tif   % Raw hyperspectral cube
├── 2DCNN_DataPreparation.m              % script for CNN input
├── GCN_DataPreparation.m                % script for GCN input
├── TR_TE_Generation2d_CNN.m             % helper to extract patches
├── hyperConvert2d.m                     % Convert 3D HSI to 2D bands×pixels
├── hyperConvert3d.m                     % Convert normalized 2D back to 3D cube
├── tf_utils.py                          % Mini-batch generators & helpers
├── FuNet-C.py                           % Model definition & training loop
├── X_train.mat, X_test.mat, ...         % MATLAB-generated inputs
├── Train_X.mat, Train_L.mat, TrLabel.mat  % GCN features, laplacians, labels
└── features.mat                         % Saved latent features after training
```

3. Data Preparation (MATLAB)

3.1 Hyperspectral Cube Loading

- **Script**: 2DCNN_DataPreparation.m and GCN_DataPreparation.m load IndianPine_Site3.tif.
- **Band Selection**: Retain bands [1:103,109:149,164:219] to exclude noisy channels.

- **Dimensionality**: Cube size becomes (m × n × z), typically ~145×145×188.

3.2 Normalization & Conversion

- **Spectral Normalization**: 2D flattening (hyperConvert2d) yields a (bands × pixels) matrix; each row scaled to [0,1].
- **Spatial Reconstruction**: For CNN inputs, hyperConvert3d reshapes normalized bands back to (m × n × z).

3.3 Train/Test Mask Generation

- **Ground Truth Masks**: IndianTR123_temp123.tif & IndianTE123_temp123.tif define pixel-level train/test labels.
- **Patch Extraction**: TR_TE_Generation2d_CNN.m pads the cube and slides a $(2r+1)\times(2r+1)$ window, building:
  - HSI_TR, HSI_TE: flattened spectral–spatial patches (size: #samples × $\text{patchSize}^2 \cdot z$)
  - HSI_TR_P, HSI_TE_P: central pixel's spectral vector (#samples × z)
  - TR2d, TE2d: one-hot encoded class labels

3.4 GCN Graph Construction

- **Feature Matrix**: Combine all train + test spectral vectors in GCN_DataPreparation.m to form X (bands×N).
- **Affinity & Laplacian**: Compute K-NN graph (K=10), affinity W, degree D, symmetrically normalized Laplacian L = D^(-1/2)·W·D^(-1/2) + I.
- **Outputs**: Save ALL_X, ALL_L, ALL_Y in MAT-files for GCN input.

4. Model Architecture (Python / TensorFlow)

4.1 Placeholders & Inputs

- **x_in**: GCN spectral features (batch × 200)

- **x_in1**: CNN spatial patches (batch × 9800)
- **lap_train**: Laplacian submatrix (batch × batch)
- **y_in**: One-hot labels (batch × C)

4.2 GCN Branch

1. **Linear Transform**: x_mid = x_in · W1 + b1
2. **Graph Convolution**: x_a1 = ReLU(L · x_mid)

4.3 CNN Branch

1. **Reshape** → (7×7×200)
2. **Conv1**: 3×3, 200→32 filters + max-pool (2×2), ReLU
3. **Conv2**: 3×3, 32→64 filters + max-pool, ReLU
4. **Conv3**: 1×1, 64→128 filters + max-pool, ReLU
5. **Flatten** → 1D features (batch × 6272)

4.4 Feature Fusion & Classification

- **Concatenate** GCN output (batch×128) and CNN features → (batch×6400)
- **Two Dense Layers**: 6400→128→C
- **Softmax Cross-Entropy** + **L2 Regularization**

5. Training Pipeline

1. **Load Inputs**: via scio.loadmat for both branches.
2. **Mini-batches**: random_mini_batches_GCN1 shuffles and yields synchronized batches of (x, x1, y, L).
3. **Optimizer**: Adam with exponential-decay LR (base=0.001, decay 0.5 every 50 epochs).
4. **Metrics**: Track train & validation cost and accuracy every 5 epochs; print every 50.

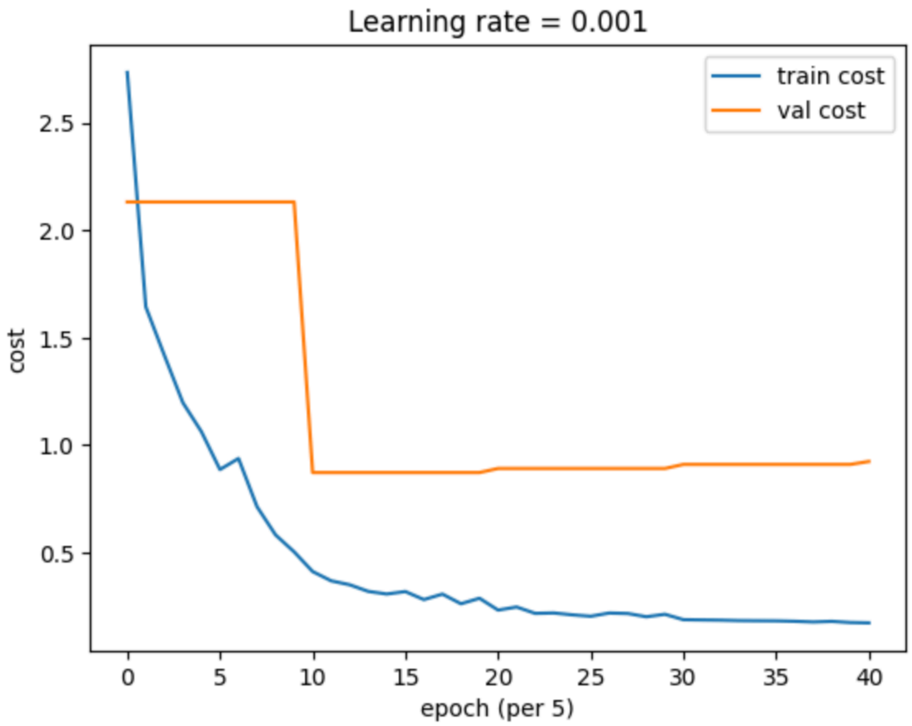
5. **Visualization**: Plot cost & accuracy curves post‑training.

6. **Feature Extraction**: Save final layer activations (features.mat).

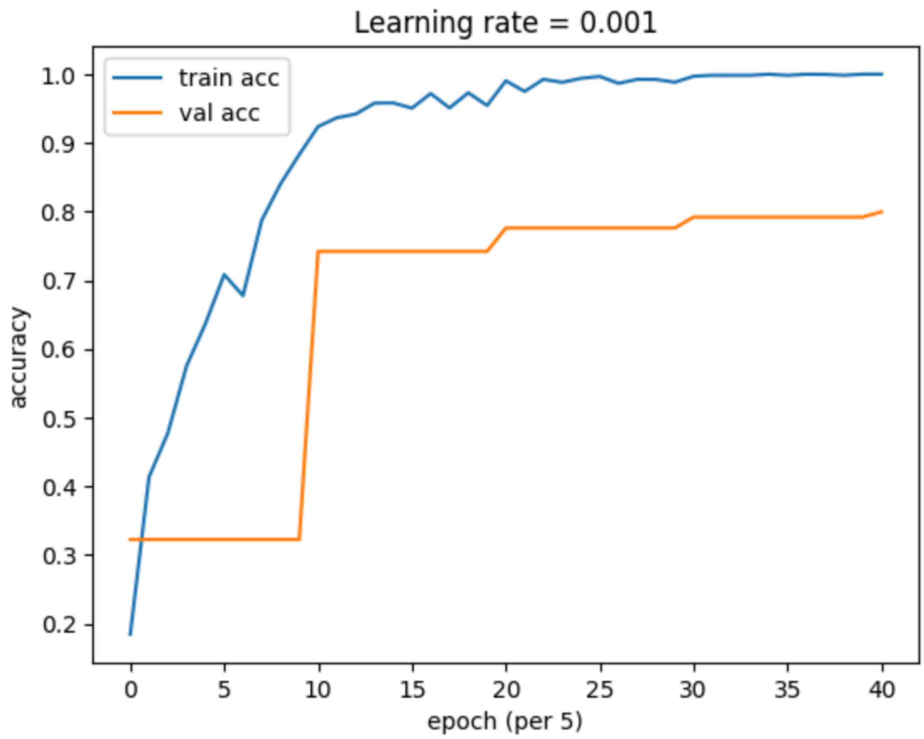
6. Experimental Results

epoch 0: Train_loss=2.735093, Val_loss=2.132374, Train_acc=0.1845, Val_acc=0.3224
epoch 50: Train_loss=0.412249, Val_loss=0.872993, Train_acc=0.9238, Val_acc=0.7419
epoch 100: Train_loss=0.232829, Val_loss=0.891881, Train_acc=0.9904, Val_acc=0.7760
epoch 150: Train_loss=0.188072, Val_loss=0.910777, Train_acc=0.9970, Val_acc=0.7920
epoch 200: Train_loss=0.173238, Val_loss=0.924551, Train_acc=1.0000, Val_acc=0.7996

Cost Curve :



Accuracy Curve :



Best achieved validation accuracy: ~80% at epoch 200.

7. Conclusion & Future Work

- **Achievements**: Demonstrated effective fusion of spectral and spatial features in hyperspectral classification.
- **Limitations**: Moderate overfitting beyond epoch 200; deeper graph layers may help.
- **Next Steps**:
  - Experiment with varying patch sizes (r > 3)
  - Add batch-normalization in conv-branch
  - Replace static Laplacian with learnable adjacency

Github : https://github.com/ashutoshrabia/funetC