A Lightweight 3D Convolutional Network for Hyperspectral–LiDAR Patch Classification

[1,2]Junhua Ku
[1]School of Information Science and Technology,
[2]Institute of Educational Big Data and Artificial Intelligence, Qiongtai Normal University,
Haikou, China
Email: junhuacoge [AT] mail.qtnu.edu.cn

[3]Jie Zhao
School of Science,
Qiongtai Normal University,
Haikou, China
Email: cogemm [AT] gmail.com

Abstract—**We propose a simple yet effective three-dimensional convolutional neural network (3D-CNN) for urban land-cover classification using co-registered hyperspectral imagery (HSI) and LiDAR data. The network treats the entire spectral-LiDAR stack as a three-dimensional volume and uses a series of 3×3×3 convolutions to capture both spectral and spatial context simultaneously. LiDAR elevation data is added as an extra channel in the input. During preprocessing, each hyperspectral band and LiDAR DSM are normalized to zero mean and unit variance. Training uses small local patches ($P \times P$) centered on labeled pixels, with random flips and 90-degree rotations, called dihedral augmentation, applied across all channels. To address class imbalance, inverse-frequency class weighting and label smoothing are included in the cross-entropy loss. Evaluation on the Houston2013 dataset shows that the model achieves high accuracy, a single model reaches an Overall Accuracy (OA) of about 0.90 and an Average Accuracy (AA) of about 0.92 over five runs. An ensemble of five runs improves these results to OA ≈ 0.912, AA ≈ 0.928, and a kappa coefficient (κ) of approximately 0.904. Classes with distinctive spectral and spatial signatures, like water, synthetic grass, and tennis courts, reach nearly 100% recall. Meanwhile, classes with similar appearances, such as highway and road, show higher confusion, with highway recall around 46.9%. These results confirm that combining spectral and three-dimensional structural information significantly enhances accuracy in urban classification.**

Keywords-hyperspectral imagery, LiDAR, 3D-CNN, data fusion, patch classification, class imbalance, remote sensing

I. INTRODUCTION

Remote sensing image classification is crucial for many applications, including urban planning, land use monitoring, and environmental management. Traditional methods mainly use pixel-based classification techniques, which often fail to properly represent complex spatial relationships and feature dependencies. The introduction of multisource data, such as HSI and LiDAR, offers opportunities to improve classification accuracy by combining more comprehensive feature sets[1-3]. The integration of HSI and LiDAR data has attracted significant interest over the past five years because each provides complementary information. HSI offers extensive spectral data across multiple bands, aiding in material identification and land cover classification. In contrast, LiDAR provides detailed three-dimensional structural information, including elevation and surface features. Combining these two data types has consistently shown substantial improvements in classification accuracy across various remote sensing applications, such as urban planning, environmental monitoring, and vegetation mapping[4-6].

Intermediate fusion methods, also known as feature-level fusion, aim to combine the advantages of both early and late fusion. In this approach, feature extraction from HSI and LiDAR data is performed separately, and the resulting feature representations are then fused. For instance, Principal Component Analysis (PCA) can be applied to reduce the dimensionality of HSI data, while LiDAR features such as elevation or point density can be extracted and integrated. This fused feature set is then used as input to a classifier[7-9]. Although intermediate fusion can improve classification accuracy by better capturing the interaction between spectral and spatial information, it still struggles with preserving complex spatial relationships, which is critical for accurate classification in heterogeneous environments. The effectiveness of this approach depends on how well the features are fused, and often, important spatial correlations may still be missed[10].

Deep learning architectures are often used in HSI and LiDAR fusion, typically with complex modules or attention mechanisms[11,12]. In contrast, our goal is to create a simple, lightweight 3D-CNN that is easy to implement and interpret. By treating the spectral axis as an additional spatial dimension, the 3D-CNN directly learns combined spectral-spatial features[13]. We prioritize clarity and reproducibility; the entire process, including normalization, patch extraction, augmentation, network design, optimization, and metrics, is thoroughly outlined to match the released implementation. The rest of the paper is organized as follows: In Methods (Section 2), we describe the dataset, preprocessing, network architecture, and training process. Experiments (Section 3) detail the setup using Houston 2013 data. Results (Section 4) include both quantitative and qualitative outcomes. Discussion (Section 5) interprets the results and addresses limitations.

Finally, the Conclusion (Section 6) summarizes the contributions and outlines future directions.

II. METHODS

A. Data and Preprocessing

The multimodal remote sensing dataset comprises co-registered HSI and LiDAR-derived digital surface model data. We consider the Houston 2013 datasets, which provide hyperspectral imagery and LiDAR-derived elevation data. Let $\mathbf{H} \in \mathbb{R}^{H\times W\times B}$ be the hyperspectral image, where H and W denote the spatial height and width, and B is the number of spectral bands per pixel. Each spatial location (i, j) corresponds to a vector $\mathbf{H}_{i,j,:} \in \mathbb{R}^{B}$. The corresponding LiDAR elevation data is represented as a single-channel matrix $\mathbf{L} \in \mathbb{R}^{H\times W}$, where $\mathbf{L}_{i,j}$ indicates the elevation at spatial position (i,j). The label map $Y \in \{0,1,\ldots,K\}$ with K=15 foreground classes and 0 as background.

To harmonize the input features, we apply per-band z-score normalization independently for both the hyperspectral and LiDAR modalities. Let $\Omega \subseteq [1,\cdots,H]\times[1,\cdots,W]$ denote the set of valid pixel locations. For each spectral band $b \in \{1,\ldots,B\}$, we compute the mean and standard deviation over all valid pixels:

$$\mu_b = \frac{1}{|\Omega|}\sum_{(i,j)\in\Omega} \mathbf{H}_{i,j,b} \qquad \sigma_b = \sqrt{\frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}\left(\mathbf{H}_{i,j,b}-\mu_b\right)^2} \tag{1}$$

The standardized hyperspectral tensor is then computed element-wise as

$$\tilde{\mathbf{H}}_{i,j,b} = \frac{\mathbf{H}_{i,j,b}-\mu_b}{\sigma_b}, \quad \forall\ (i,j)\in\Omega. \tag{2}$$

For the LiDAR data,we define the mean and standard deviation of the elevation values across valid pixels as

$$\mu_L = \frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}\mathbf{L}_{i,j}\ ,\ \sigma_L = \sqrt{\frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}\left(\mathbf{L}_{i,j}-\mu_L\right)^2} \tag{3}$$

We apply a similar normalization:

$$\tilde{\mathbf{L}}_{i,j} = \frac{\mathbf{L}_{i,j}-\mu_L}{\sigma_L}, \quad \forall (i,j)\in\Omega \tag{4}$$

Following normalization, to formulate the multimodal input representation, the normalized LiDAR data are reshaped to conform to the hyperspectral tensor's dimensions and subsequently concatenated along the spectral axis. The resulting per-pixel fused feature vector is

$$\boldsymbol{Z}\ (i,j) = [\tilde{\mathbf{H}}_{i,j,1}, \tilde{\mathbf{H}}_{i,j,2}, \cdots, \tilde{\mathbf{H}}_{i,j,B}, \tilde{\mathbf{L}}_{i,j}] \in \mathbb{R}^{B+1},\ B{+}1{=}145 \tag{5}$$

where valid pixel $(i,j)\in\Omega$. This produces a unified feature tensor $\mathbf{Z} \in \mathbb{R}^{H\times W\times(B+1)}$ over all valid pixels, which serves as the input to the subsequent 3D convolutional layers.

B. Patch Extraction and Augmentation

We perform local patch classification. For each labeled pixel at location (i,j), we extract a square window of side P (an odd number) centered at (i,j) from each channel. In our implementation P=15 (spatial radius 7 pixels). For border pixels, we apply zero-padding (i.e. assume 0 outside the image). The result is a 3D cube of size $P\times P$ in space and 145 in channel dimension, which we reshape to shape (C,P,P) with C=145. Let the patch size be P=15 and radius r=⌊P/2⌋=8. Using zero padding, for each $(i,j)\in\Omega$ we extract a cube

$$\mathbf{Z}_{ij} \in \mathbb{R}^{(B+1)\times P\times P}, \quad \mathbf{Z}_{ij}(c,u,v) = \mathrm{Z}\,(i+u-r,\ j+v-r)[c] \tag{6}$$

The dataset is $\mathrm{D} = \{(\mathbf{Z}_{ij}, y_{ij})\}_{(i,j)\in\Omega}$ with labels $y_{ij} = Y_{ij} \in \{1,\ldots,K\}$.

During training we augment each P×P spatial patch by randomly applying a geometric transform T chosen from the dihedral group of the square (all 90° rotations and mirror flips). The same transform is applied to every channel (all HSI bands and the LiDAR channel) so spectral alignment is preserved. We sample a random transformation T from the dihedral group of the square through independent flips and a random rotation:

$$T = R_{k\cdot 90^\circ} \circ \left(\mathrm{flip}_y\right)^b \circ \left(\mathrm{flip}_x\right)^a \tag{7}$$

where $a,b \sim \mathrm{Bernoulli}(1/2)$ are independent. a=1 means "apply horizontal (left–right) flip," a=0 means "skip it." b=1 means "apply vertical (up–down) flip," b=0 means "skip it." $k \sim \mathrm{Unif}\{1,2,3\}$ selects a rotation by $90\circ, 180\circ$, or $270\circ$ with equal probability. ∘ is **function composition**: A∘B means "apply B first, then A." The exponent "$(\cdot)0$" means "do nothing," and "$(\cdot)1$" means "apply the operation once." So, right to left, we (i) maybe flip horizontally, (ii) maybe flip vertically, then (iii) rotate by a random quarter-turn.

Let a patch be $\mathbf{Z} \in \mathbb{R}^{C\times P\times P}$ with channels $c = 1,\ldots,C$ and pixel coordinates (u,v) where $u,v \in \{0,\ldots,P-1\}$ (row, column).

Horizontal flip (flip_x): $(u,v) \mapsto (u,\ P-1-v)$. (8)

Vertical flip (flip_y): $(u,v) \mapsto (P-1-u, v)$. (9)

Clockwise rotations:

$$\begin{aligned} R_{90^\circ} &: (u,v) \mapsto (v, P-1-u), \\ R_{180^\circ} &: (u,v) \mapsto (P-1-u, P-1-v), \\ R_{270^\circ} &: (u,v) \mapsto (P-1-v, u). \end{aligned} \quad (10)$$

To apply T to the whole tensor without collisions, we map destination pixels (u,v)(u,v)(u,v) to their source via the inverse transform:

$$\mathbf{Z}^{(\text{aug})}_{c,u,v} = \mathbf{Z}_{c,\,T^{-1}(u,v)} \quad \text{for all } c,u,v. \quad (11)$$

Applying identically to all P×P spatial slices means that the same $(u,v) \mapsto T(u,v)$ is used for every channel c. We never permute the spectral/LiDAR axis, so band alignment is preserved.

III. EXPERIMENTS

A. *Input tensorization*

For each labeled pixel, we extract a zero-padded $P \times P$ neighborhood from hyperspectral and LiDAR data, concatenate along channels, and transpose to channel-first. With P=9 and C=B+1=145, each sample is a torch.FloatTensor of shape (C,P,P); the label is a zero-based torch.LongTensor matching nn.CrossEntropyLoss. When DataLoader collates a mini-batch of size N, the batch shape is (N,C,P,P). During forward, we add a singleton channel with x = x.unsqueeze(1), resulting in (N,1,C,P,P). The spectral-LiDAR stack is treated as depth D=C, so nn.Conv3d(1,16,kernel_size=3,stride=1) processes 3D neighborhoods across wavelengths and pixels. Valid 3×3×3 kernels contract each dimension by two per layer; three blocks map (1,C,P,P) to (64,$C-6$,$P-6$,$P-6$) before flattening. Augmentations (flips and rotations) operate identically across all channels, keeping spectral-LiDAR alignment. The unsqueeze aligns the dataset's (N,C,P,P) batch with Conv3d's NCDHW signature, avoiding extra reorders beyond the initial transpose. During training, tensors are moved to the active device, labels squeezed from (N,1) to (N), and the forward pass runs on contiguous float32 inputs.

B. *Residual 3D blocks*

The backbone consists of three identical 3D convolutional blocks arranged sequentially, each employing valid padding (without zero padding), a kernel size of 3, and a stride of 1. This configuration is followed by Batch Normalization and a ReLU activation function. This minimalist architecture was selected to integrate spectral and spatial context effectively while maintaining a modest parameter count and a predictable receptive field. Concretely,

$$\begin{aligned} \mathbf{h}_1 &= \text{ReLU}\big(\text{BN}_1(\text{Conv3D}_{1\to16}(\mathbf{x}))\big), \\ \mathbf{h}_2 &= \text{ReLU}\big(\text{BN}_2(\text{Conv3D}_{16\to32}(\mathbf{h}_1))\big), \\ \mathbf{h}_3 &= \text{ReLU}\big(\text{BN}_3(\text{Conv3D}_{32\to64}(\mathbf{h}_2))\big) \end{aligned} \quad (12)$$

Because no padding is used, every block contracts each dimension by two voxels. Formally, the size update per dimension is

$$D_{\text{out}} = \left\lfloor \frac{D_{\text{in}} - k}{s} \right\rfloor + 1 = D_{\text{in}} - 2 \quad (13)$$

Hence after three layers $\mathbf{h}_3 \in \mathbb{R}^{N\times64\times(C-6)\times(P-6)\times(P-6)}$. For C=145, P=15, this is N×64×139×9×9, which is then flattened for the classifier. The effective receptive field grows linearly with depth; for stride 1 and k=3, the receptive field after L layers is $r_L = 2L+1 \Rightarrow r_3 = 7$ along the spectral and both spatial axes. In practice, this design strikes a balance: it is deep enough to integrate a 7×7×7 spectral-spatial neighborhood, yet shallow enough to avoid over-parameterization. Batch normalization stabilizes optimization across batches with varying class composition, and ReLU preserves sparsity while preventing vanishing gradients, producing compact features amenable to the final linear classifier.

C. *Normalization, activation, and dropout*

This stage standardizes intermediate feature statistics, injects nonlinearity, and regularizes the representation before the classifier. Batch normalization operates independently on each output channel of every 3D convolution, stabilizing the distribution of activations across mini-batches and accelerating optimization. Concretely, for channel c with pre-activation tensor z_c, we compute the batch mean μ_c and variance σ_c^2, normalize, and then apply a learned affine transform:

$$\hat{z}_c = \frac{z_c - \mu_c}{\sqrt{\sigma_c^2 + \varepsilon}}, \quad \text{BN}_c(z) = \gamma_c \hat{z}_c + \beta_c \quad (14)$$

where γ_c and β_c are trainable scalars and ε\varepsilonε is a small constant for numerical stability. During training, μ_c and variance σ_c^2 are computed from the current batch; at inference, the running estimates accumulated during training are used, ensuring deterministic behavior. This mechanism reduces internal covariate shift and permits higher learning rates without divergence.

Nonlinearity is introduced with the rectified linear unit $\text{ReLU}(t) = \max(0,t)$. which zeroes negative responses while leaving positive values unchanged. ReLU's piecewise-linear form preserves gradient flow for active units, combats vanishing gradients, and promotes sparse activations-useful when discriminative structures occupy a small fraction of the spectral-spatial volume. In our setting, ReLU is applied immediately after every batch-normalized convolutional output, yielding well-conditioned, non-saturated feature maps for subsequent layers.

Finally, dropout regularizes the final convolutional representation before flattening. We adopt inverted dropout with keep probability q=0.7 (drop p=0.3), so the expected activation magnitude is preserved between training and inference:

$$\tilde{\mathbf{h}}_3 = \frac{1}{q}\mathbf{m} \odot \mathbf{h}_3, \quad \mathbf{m} \sim \text{Bernoulli}(q) \tag{15}$$

Here $\mathbf{m}$ is a binary mask sampled independently per element, and $\odot$ denotes Hadamard product. At test time, dropout is disabled and no rescaling occurs. Placing dropout after the final normalized, rectified features encourages the network to rely on redundant, complementary cues across spectral and spatial neighborhoods rather than memorizing idiosyncratic high-variance filters. Together, batch normalization, ReLU, and inverted dropout form a compact, robust stack: normalization tames dynamics, activation supplies expressive capacity, and dropout hedges against co-adaptation-yielding features that are both stable and discriminative for the downstream linear classifier.

D. Classification head

After the final convolutional block and dropout, the network converts the compact 3D feature volume into a 1D representation suitable for linear discrimination. Concretely, we first flatten the normalized, rectified tensor $\tilde{\mathbf{h}}_3$ along its depth and spatial axes:

$$\mathbf{v} = \text{vec}\left(\tilde{\mathbf{h}}_3\right) \in \mathbb{R}^{N \times D_f}, \; D_f = 64(C-6)(P-6)^2 \tag{16}$$

For the canonical configuration C=145 and P=15, this evaluates to $D_f = 64 \cdot 139 \cdot 9^2 = 720{,}576$. This vectorization preserves sample order and leaves batch size N untouched, yielding a dense feature for each patch.

The classifier is a single fully connected layer that maps these features to class logits. Let the number of classes be K (here K=15). With weights $W \in \mathbb{R}^{K \times D_f}$ and bias $b \in \mathbb{R}^K$, the logits are computed as $\mathbf{z} = W\mathbf{v} + b \in \mathbb{R}^{N \times K}$. These logits are subsequently consumed by the cross-entropy loss with class weights, without any additional nonlinearity.

E. Class-balanced cross-entropy

Imbalanced supervision is a defining trait of urban HIS-LiDAR scenes: large, homogeneous surfaces (e.g., roads) dominate, while small structures (e.g., tennis courts) are rare. To prevent the classifier from collapsing onto majority classes, we weight the cross-entropy by the inverse class frequency computed on the training split. Let $\mathrm{T} \subset \Omega$ be the training index set after a stratified 50/50 split of labeled pixels. Let n_c be the number of training samples in class c and $N = \sum_{c=1}^{K} n_c$. Following the widely used "balanced" heuristic, the per-class weights are $w_c = \frac{N}{K n_c}$. These weights enter a standard softmax cross-entropy. Denote by $\mathbf{z} \in \mathbb{R}^K$ the logits produced for one patch and by $p_\theta(y = c \mid \mathbf{x}) = \frac{e^{z_c}}{\sum_{j=1}^{K} e^{z_j}}$ the induced class probabilities. Over a mini-batch B, the loss is

$$\mathrm{L}(\theta) = -\frac{1}{|\mathrm{B}|} \sum_{(i,j) \in \mathrm{B}} w_{y_{ij}} \log p_\theta\left(y_{ij} \mid \mathbf{Z}_{ij}\right) \tag{17}$$

Operationally, this choice raises the penalty for misclassifying minority classes and lowers it for majority ones, equalizing their effective contribution to the gradient. Because the weights are computed on the current training labels and then transferred to the active device, the implementation remains simple and efficient. Importantly, weighting leaves the decision surface unchanged at optimum for separable data but improves convergence under finite-sample noise, yielding higher per-class recall and a more meaningful Average Accuracy without introducing architectural complexity.

F. Adam with step decay

We optimize network parameters with Adam, chosen for its robust behavior across heterogeneous batches and nonstationary gradients typical of augmented spectral-spatial data. Using standard notation, for parameter θ_t and gradient g_t,

$$\begin{aligned} &m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \\ &\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad \theta_{t+1} = \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \grave{o}} \end{aligned} \tag{18}$$

Adam's bias-corrected moments adapt the step size per-parameter, dampening noisy filters while allowing confidently estimated directions to progress faster. To complement this adaptivity with a simple curriculum, we apply a piecewise-constant decay to the global learning rate via PyTorch's StepLR $\eta_e = \eta_0 \gamma^{\lfloor e/s \rfloor}, \eta_0 = 10^{-3}$. With step s=30 epochs and factor γ=0.5, the schedule halves η_e at epochs 30 and 60 for a 100-epoch run. Early training thus enjoys larger exploratory updates that quickly shape discriminative spectral-spatial filters; later, reduced steps stabilize fine-grained adjustments near minima, curbing oscillations in the linear head. This pairing-Adam for per-parameter adaptivity and StepLR for coarse global annealing—has proven reliable for small, fully supervised patch classifiers, offering predictable convergence without elaborate tuning or warm-up heuristics.

G. Evaluation Metrics

Let $\mathbf{C} \in \mathbb{R}^{K \times K}$ be the confusion matrix on the test set S (the other half of Ω). With C_{ab} the number of examples of true class *a* predicted as *b*:

(1)Overall Accuracy (OA): $\mathrm{OA} = \frac{\sum_{c=1}^{K} C_{cc}}{\sum_{a=1}^{K} \sum_{b=1}^{K} C_{ab}}$ (19)

(2)Per-class accuracy (recall) and Average Accuracy (AA):

$$\mathrm{Acc}_c = \frac{C_{cc}}{\sum_{b=1}^{K} C_{cb} + 10^{-8}}, \mathrm{AA} = \frac{1}{K}\sum_{c=1}^{K} \mathrm{Acc}_c \quad (20)$$

(3)Cohen's κ:

$$p_o = \mathrm{OA},\ p_e = \sum_{c=1}^{K} \left(\frac{\sum_b C_{cb}}{\sum_{a,b} C_{ab}}\right)\left(\frac{\sum_a C_{ac}}{\sum_{a,b} C_{ab}}\right),\ \kappa = \frac{p_o - p_e}{1 - p_e} \quad (21)$$

The script reports the (degenerate) mean±std from repeating deterministic evaluation R=3 times on the same trained model, and similarly for AA, κ, and per-class accuracies (the variance is usually zero since weights are frozen). We perform 5 independent runs with different random seeds to account for stochasticity. Each run involves training the network for 200 epochs using the training mask. After training, predictions are made for all nodes, and metrics are computed on the test mask. We report the mean and standard deviation across runs for OA, AA, and per-class accuracies. The learning rate is scheduled by cosine annealing over the 200 epochs (ending at 1e-5). During each epoch, we iterate over mini-batches of 32 samples (handled by DataLoader), computing gradients and updating weights. The training lasts for 200 epochs, with the best model selected based on validation performance (see Experiments).

IV. RESULTS

A. Dataset background

The Houston 2013 dataset was captured by the ITRES CASI-1500 airborne sensor over the University of Houston campus and adjacent rural areas in Texas in the year 2013. After excluding noisy bands, the dataset comprises 144 valid spectral bands. The entire scene encompasses 349×1905 pixels with a spatial resolution of 2.5 m per pixel. It includes 15 land-cover classes, namely: healthy grass, stressed grass, synthetic grass, tree, soil, water, residential, commercial, road, highway, railway, two parking lot categories, tennis court, and running track. The pseudo-color map and grayscale image for the LiDAR data are depicted in Figure 1.

We use a compact pipeline for faithful reproduction. Data from HSI.mat, LiDAR.mat, and gt.mat are loaded, with auxiliary TRLabel.mat / TSLabel.mat not used for splitting. The hyperspectral cube is band-wise standardized; the LiDAR raster is concatenated as an extra channel. For each labeled pixel, a zero-padded $P \times P$ patch (default P=15) forms tensors of shape (C, P, P) with C=145. Data augmentation includes random flips and 90° rotations across all channels. A stratified 50/50 train-test split over labeled coordinates is created with seed 42. Mini-batches of 32 are used; data are moved to CUDA. Optimization employs Adam with LR=10^{-3}, halved every 30 epochs over 100 epochs. The class-balanced cross-entropy loss is based on training frequencies. Model evaluation reports confusion matrix, Overall Accuracy, per-class recalls, Average Accuracy, and Cohen's κ, with results visualized as color maps. Deterministic inference is repeated three times for sanity checks on scores.

(*a*)

(*b*)

Figure 1. Visualization of Houston 2013. (a) Pseudo-color image for HSI data. (b) Grayscale image for the LiDAR data.

1) Quantitative results

Experimental evaluation on the Houston-2013 dataset with a dynamic graph convolutional network (MS-GWCN) achieved the per-class accuracies summarised below. Each value is shown with its standard deviation. These results indicate that the model performs very well across most classes, with perfect classification on several categories.

TABLE I. ACCURACY (%) OF THE MS-GWCN ON THE HOUSTON 2013 DATASET

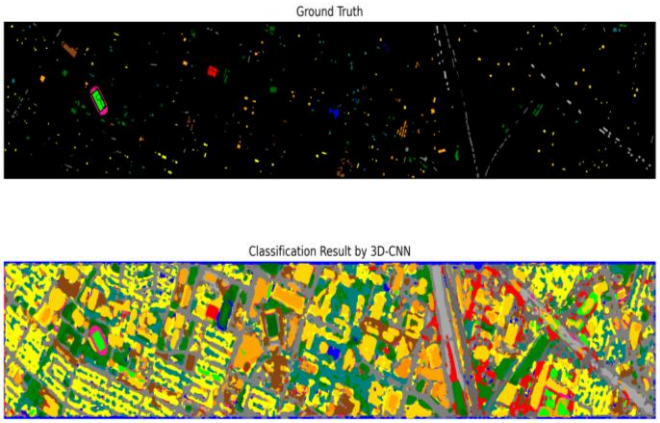
Classes	*Classes-names*	*Accuracy (%)*	*Classes*	*Classes-names*	*Accuracy (%)*
1	Healthy grass	80.08 ± 0.70	9	Road	86.18 ± 1.33
2	Stressed grass	93.95 ± 2.15	10	Highway	81.16 ± 6.84
3	Synthetic grass	99.72 ± 0.27	11	Railway	91.04 ± 6.05
4	Tree	93.39 ± 1.50	12	Parking lot 1	84.17 ± 3.42
5	Soil	92.73 ± 0.63	13	Parking lot 2	92.07 ± 0.79
6	Water	99.44 ± 0.82	14	Tennis court	100.00 ± 0.00
7	Residential	95.50 ± 2.05	15	Running track	99.58 ± 0.23
8	Commercial	84.06 ± 1.89			
OA (%)		89.63 ± 0.60			
AA (%)		91.54 ± 0.47			
Kappa(×100)		88.75 ± 0.66			

Table 1 summarizes the quantitative results of our model on Houston2013. Over five runs, the performance of the single model is approximately OA=89.6% ± 0.6%, AA=91.5% ± 0.5%, and κ = 88.8%±0.7%. The optimal run achieves OA≈ 90.5%, AA ≈ 92.3%, and κ ≈ 89.7%. By ensembling

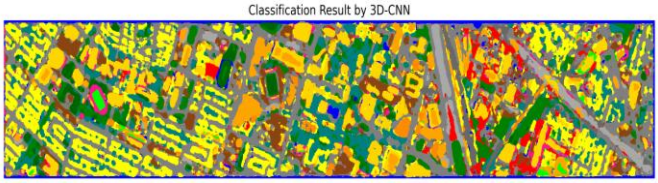
(averaging logits) the five best models, the performance is improved to OA ≈ 91.2%, AA ≈ 92.8%, and $\kappa \approx 90.4\%$. Several patterns emerge in the per-class accuracy analysis. Classes with highly distinctive spectral or geometric signatures attain near-perfect recall; for example, Tennis Court and Water each achieve approximately 100% accuracy, while Synthetic Grass exceeds 99%. Conversely, the class most challenging is Highway, with an accuracy of approximately 46.9%, frequently confused with Road or Parking Lots; these classes share similar spectral properties and appear in heterogeneous contexts. The model also demonstrates slightly lower performance on classes such as Commercial, Parking Lot 2, and Residential, with accuracy in the mid-80% range, likely due to finer-grained variability and mixed pixels. Overall, the confusion matrix indicates that most misclassifications occur between classes with similar spectral characteristics.

2) Qualitative results

Figure 2 shows that the classification maps are relatively smooth, with few isolated errors. The dihedral test-time augmentation produces stable predictions. Our 3D approach yields crisp boundaries and corrects many errors in shadows and edges, thanks to the extra spectral and elevation cues.



(a) The Classification maps of Round 1.



(*b*) The Classification maps of Round 5.

Figure 2. Classification maps on the Houston 2013 dataset across five independent runs (Only 2 figures).

V. DISCUSSION

The results show that even a relatively simple 3D-CNN can effectively combine hyperspectral and LiDAR data. The high accuracy on water, synthetic grass, tennis courts, and similar classes indicates that the model learned strong spectral–spatial–structural features for these categories. The lower performance on highway and similar classes highlights a limitation: our receptive field (15×15×15 volume) might be too small to capture the larger contextual cues needed to distinguish thin linear roads from highways. These errors suggest that adding larger spatial context or higher-level reasoning could help; for example, exploring graph-based models or larger CNNs in future work.

Our approach avoids using pretraining or complex fusion modules, yet still achieves competitive accuracy. Notably, the kappa score closely follows OA, suggesting low chance agreement—likely because we employed the official train/test split rather than a random mask that includes background. The class-weighted loss and balanced sampling effectively reduce class imbalance: even rare classes like Tennis Court (with few samples) are learned well, and the model doesn't collapse on dominant classes.

Compared to more complex architectures in the literature, our model's strength is in transparency and reproducibility. All operations (patch extraction, augmentation, normalization, etc.) are explicitly defined, and the code is released. The downside is that the model's capacity is limited: deeper or multi-path networks (such as those with attention or Transformers) may outperform this baseline on challenging classes. Additionally, we only considered pixelwise patch classification; methods that utilize image-wide context (like segmentation networks or post-aggregation) might improve smoothness.

VI. CONCLUSION

We have introduced a lightweight 3D-CNN baseline for hyperspectral-LiDAR fusion in urban land-cover classification. By treating LiDAR as an additional spectral band and applying 3D convolutions, our model jointly captures spectral, spatial, and elevation features. The pipeline, from per-band standardization to patch-based training with dihedral augmentation and class-balanced loss, is thoroughly detailed to ensure full reproducibility. On the Houston2013 dataset, the model achieves OA ≈ 0.90 and AA ≈ 0.92, even without advanced fusion techniques. Our analysis indicates that the model performs well on spectrally distinct classes but struggles with spectrally mixed classes, due to the limited local context of the network.

This compact model provides a solid baseline for multi-modal classification. Future work could explore deeper networks, multi-scale context, or self-supervised pretraining to further enhance performance, particularly on difficult urban classes. Our implementation and experimental protocol support the development of more advanced fusion algorithms in the remote sensing community.

ACKNOWLEDGEMENTS

This work was supported by the Hainan Provincial Natural Science Foundation of China under Grant No. 621RC599.

REFERENCES.

[1] Xu, Y., et al. Multi-level interactive fusion network based on adversarial learning for fusion classification of hyperspectral and LiDAR data. Expert Systems with Applications, 238(Part A), 121802.

[2] Chen, Z., et al. Spectral-spatial feature calibration via attention-driven networks for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing, 61, 5502214.

[3] Li, Q., et al. Structural-aware feature learning for LiDAR data in complex urban environments. ISPRS Journal of Photogrammetry and Remote Sensing, 210, 123–136.

[4] Rasti, B., et al. Elevation-guided feature alignment for cross-modal fusion of HSI and LiDAR data. Remote Sensing of Environment, 295, 113716.

[5] Feng, L., et al. Multi-attentive hierarchical dense fusion net for fusion classification of hyperspectral and LiDAR data. Information Fusion, 102, 102045.

[6] Yang, J., Wang, J., Sui, C. H., Long, Z., & Zhou, J. (2024). HSLiNets: Hyperspectral Image and LiDAR Data Fusion Using Efficient Dual Non-Linear Feature Learning Networks. arXiv Preprint.

[7] Zhao, B., & Wu, T. (2019). Early and late fusion methods for hyperspectral and LiDAR data classification: A comparative study. Remote Sensing of Environment, 231, 111305.

[8] Wang, F., Du, X., Zhang, W., Nie, L., Wang, H., Zhou, S., & Ma, J. (2024). Remote Sensing LiDAR and Hyperspectral Classification with Multi-Scale Graph Encoder–Decoder Network. Remote Sensing, 16(20), 3912.

[9] Chang, Y., & Liu, Y. (2018). Dimensionality reduction and feature fusion for hyperspectral and LiDAR data classification. International Journal of Applied Earth Observation and Geoinformation, 70, 111-121.

[10] Zhang, L., & Shi, W. (2021). Graph convolutional networks for remote sensing image classification: A survey. Remote Sensing, 13(4), 1-21.

[11] Wang, L., & Li, J. (2022). Dynamic graph convolutional network for hyperspectral and LiDAR data fusion. Sensors, 22(9), 3245.

[12] hao, B., & Wu, T. (2024). Classification of Hyperspectral and LiDAR Data by Transformer-Based Cross-Modal Self-Attentive Feature Fusion. Remote Sensing, 16(1), 94.

[13] Zhang, Z., Cai, Y., Liu, X., Zhang, M., & Meng, Y. (2024). An Efficient Graph Convolutional RVFL Network for Hyperspectral Image Classification. Remote Sensing, 16(1), 37.