DLSS Assignment 2

Kuon Ito

June 2025

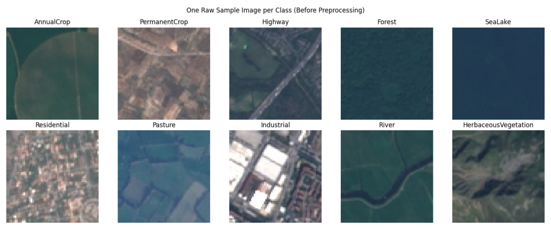
1 Introduction

Understanding land use patterns from satellite imagery is essential for applications like urban planning, agriculture, and environmental monitoring. This project tackles the task using the EuroSAT RGB dataset, which includes labeled images from 10 land use categories across Europe. I approach it as a supervised image classification problem and apply a Convolutional Neural Network (CNN) to learn spatial features. The pipeline involves data preprocessing, model training with techniques like augmentation and regularization, and performance evaluation through metrics such as accuracy, F1 score, and confusion matrix, with attention to common misclassifications.

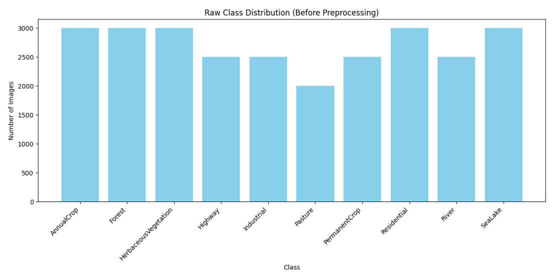
2 Results

2.1 Data

The dataset used in this project is the EuroSAT RGB dataset, which contains 27,000 color satellite images collected from various locations across Europe. Each image is 64×64 pixels in size and is labeled with one of ten land use categories, such as Residential, Industrial, Pasture, Forest, and River. This allows the task to be formulated as a supervised image classification problem, where each input image is paired with a known label. According to 3(b), the data set is balanced, with 2,000 to 3,000 items in each category.



(a) EuroSat RGB



(b) Distribution of classes

Figure 1: EuroSAT RGB dataset and class distribution

To enable effective model training and evaluation, the EuroSAT dataset was first divided into three subsets: 70% for training, 15% for validation, and 15% for testing. This partitioning was accomplished by using TensorFlow's image_dataset_from_directory() function with a fixed random seed (seed=123) to ensure reproducibility. In practice, 30% of the data was initially set aside for validation and testing, and that portion was then split evenly into validation and test sets. Once the three subsets were defined, each image was uniformly resized to 64 × 64 pixels to match the CNN's expected input dimensions, and labels were automatically inferred from the folder structure.

For the training subset, every image was passed through a rescaling layer that divided pixel intensities by 127.5 and subtracted 1.0, effectively mapping raw values from the range [0, 255] to [−1, 1]. Immediately after this normalization step, a series of random augmentations was applied: images could be flipped horizontally, rotated by up to ±10 degrees, or randomly zoomed between 80% and 120% of their original size. These transformations introduced controlled variability into the training data, helping to reduce overfitting and encouraging the model to learn features that are invariant to shifts, rotations, and scale changes. In contrast, images in the validation and test subsets underwent only the normalization step and no augmentations, ensuring that evaluation metrics reflected performance on unaltered data.

Finally, each of the three TensorFlow datasets was configured with the AUTOTUNE option for both mapping and prefetching, which parallelizes data-loading operations and minimizes I/O bottlenecks. After completing the TensorFlow-based pipeline, PyTorch's ImageFolder and DataLoader classes were used to recreate the same splits under an equivalent PyTorch configuration. The training DataLoader was instantiated with a batch size of 32, shuffle=True, and num_workers=4 to load data in parallel and randomize the example order each epoch, while the validation and test DataLoaders also used a batch size of 32 but with shuffle=False and num_workers=4, guaranteeing consistent ordering when computing evaluation metrics.

2.2 Model

The core model is defined in a class called EuroSAT_CNN, which inherits from nn.Module. This CNN begins with three convolutional blocks operating on 3-channel, 64×64 input images and ultimately produces a feature map of shape (128, 8, 8). Each convolutional block follows the sequence: a 3×3 convolution, ReLU activation, another 3×3 convolution, ReLU activation, 2×2 max pooling to downsample the spatial dimensions, and a 25% dropout layer. In the first block, the number of channels increases from 3 to 32 while the spatial size shrinks from 64×64 to 32×32. In the second block it goes from 32 to 64 channels and 32×32 to 16×16, and in the third block from 64 to 128 channels and 16×16 to 8×8. The 0.25 dropout rate in each block is intended to reduce overfitting during training.

After the three convolutional blocks, the resulting (batch_size, 128, 8, 8) tensor is passed through nn.Flatten() to convert it into a one-dimensional vector. The first fully connected layer then compresses the $128 \times 8 \times 8 = 8192$ dimensions down to 256, applies ReLU activation and a 30% dropout, and finally a second fully connected layer outputs logits for the ten land-use categories.

For optimization, I use nn.CrossEntropyLoss as the loss function and the Adam optimizer with a learning rate of 0.001 and an L2 weight decay of 0.0001. Within each epoch of training, I first call model.train() to switch to training mode, iterate over mini-batches from the training DataLoader to perform forward propagation, backpropagation, and parameter updates, and accumulate the batch losses to compute the average training loss. Next, I switch the model to model.eval()—inference mode—and run the same training data through the network again, without dropout or augmentation, to compute what I call the "training loss in evaluation mode." This step corrects for the fact that noise introduced by dropout and augmentation can make the training loss appear artificially high, ensuring that it is directly comparable to the validation loss. I then iterate through the validation DataLoader—also in model.eval() mode—to compute the average validation loss and pass that value to the early stopping criterion.

The EarlyStopping class checks whether the validation loss has improved by at least min_delta = 0.0001 compared to the previous best. If the validation loss fails to improve for patience = 5 consecutive epochs, training is halted early. Whenever the validation loss does improve, the model's weights are saved to a file named eurosat_best.pt. After training ends, the best-performing weights (from the lowest validation loss) are automatically reloaded so that we can evaluate the model before it has overfit.

Once training is complete, I visualize the epoch-by-epoch progression of three loss curves—training loss in train mode, training loss in eval mode, and validation loss. Typically, the training-mode loss is highest (due to dropout and augmentation), while the eval-mode training loss and the validation loss track closely together. Finally, I load the saved best model, call model.eval(), and run through the test DataLoader one final time to calculate test loss and accuracy. Because dropout is disabled and batch normalization uses running statistics in eval mode, this test evaluation yields an accurate measure of the model's generalization performance.

2.3 Training

The training process was monitored through loss curves for both the training and validation sets, as shown in the 2(a). Three curves are plotted: training loss during training mode, training loss in evaluation mode (without dropout), and validation loss.

From the outset, all three losses show a consistent downward trend, indicating that the model is successfully learning relevant patterns from the data. In the early epochs, both training and validation losses decrease sharply, demonstrating effective initial convergence.

As training progresses, the gap between the training loss (eval mode) and validation loss remains small, suggesting good generalization and minimal overfitting. The difference between training loss (with dropout) and evaluation mode loss also becomes more visible in later epochs, which is expected due to the regularization effect of dropout during training.

Importantly, the validation loss continues to decline or stabilize even in the later epochs, which indicates that the model does not suffer from significant overfitting. This implies that the use of data augmentation, dropout, and other regularization techniques was effective in promoting generalization.

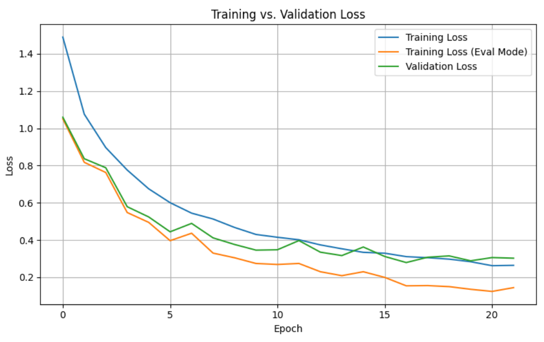
The learning curves show that early stopping was triggered at the 22nd epoch, at which point the validation loss had not improved for the specified patience, and the best model weights were saved for final evaluation.

Overall, the training dynamics indicate that the model learned progressively and remained stable throughout the training phase. The learning curves support the conclusion that the training procedure was well-balanced and that the model is unlikely to be either underfit or overfit.

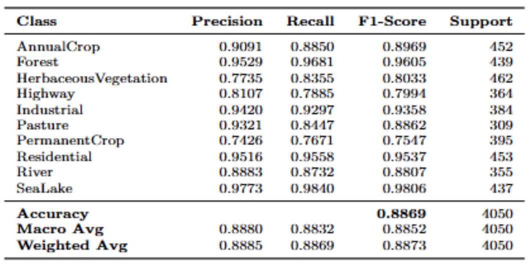
2.4 Evaluation

The trained CNN model was evaluated on the test set to assess its generalization performance on unseen data. Several performance metrics were computed, including overall accuracy, macro-averaged F1 score, a full classification report, and a confusion matrix.

The model achieved a macro-averaged F1 score of 0.90, indicating strong performance across all classes The classification report provided detailed precision, recall, and F1 scores for each of the ten land use categories. Most categories showed consistently high scores, especially those with distinct visual features such as Forest and Residential.



(a) Learning curves



Class	Precision	Recall	F1-Score	Support
AnnualCrop	0.9091	0.8850	0.8969	452
Forest	0.9529	0.9681	0.9605	439
HerbaceousVegetation	0.7735	0.8355	0.8033	462
Highway	0.8107	0.7885	0.7994	364
Industrial	0.9420	0.9297	0.9358	384
Pasture	0.9321	0.8447	0.8862	309
PermanentCrop	0.7426	0.7671	0.7547	395
Residential	0.9516	0.9558	0.9537	453
River	0.8883	0.8732	0.8807	355
SeaLake	0.9773	0.9840	0.9806	437
Accuracy			**0.8869**	4050
Macro Avg	0.8880	0.8832	0.8852	4050
Weighted Avg	0.8885	0.8869	0.8873	4050

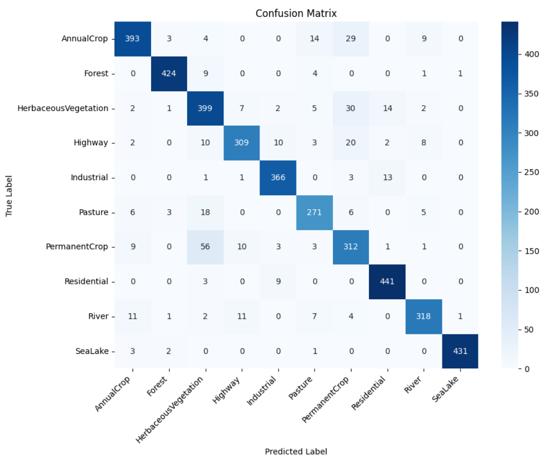
(b) Classification Report for EuroSAT Test Set

Figure 2

However, certain categories like ”Permanent Crop” and ”Herbaceous Vegetation” showed slightly lower performance, which is expected due to their visual similarity and overlapping features in satellite imagery. This pattern is further highlighted in the confusion matrix, where some misclassifications occurred between similar terrain types.

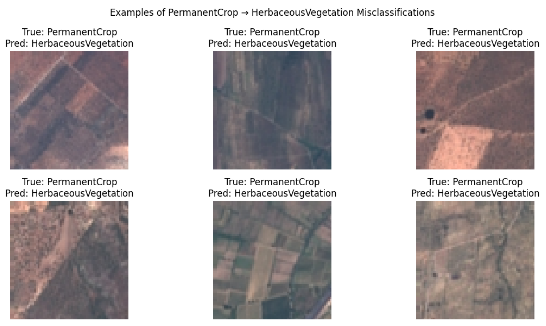
The confusion matrix, plotted as a heatmap, reveals that the majority of predictions fall along the diagonal, showing a high rate of correct classifications. A few off-diagonal elements indicate where the model struggled most—these misclassifications can often be attributed to inter-class visual ambiguity or lower representation in the training set.

One of the most difficult to classify images observed in the confusion matrix was between the Permanent Crop and Herbaceous Vegetation categories. In particular, there were 56 instances where images labeled as Permanent Crop were incorrectly predicted as Herbaceous Vegetation. Some of hypothesize reasons including the following: Permanent Crop and Herbaceous Vegetation are difficult to distinguish because they often share similar textures and color patterns in low-resolution satellite images. Moreover, the RGB-only EuroSAT dataset lacks the spectral information needed to capture subtle differences in vegetation type and density.



True Label \ Predicted Label	AnnualCrop	Forest	HerbaceousVegetation	Highway	Industrial	Pasture	PermanentCrop	Residential	River	SeaLake
AnnualCrop	393	3	4	0	0	14	29	0	9	0
Forest	0	424	9	0	0	4	0	0	1	1
HerbaceousVegetation	2	1	399	7	2	5	30	14	2	0
Highway	2	0	10	309	10	3	20	2	8	0
Industrial	0	0	1	1	366	0	3	13	0	0
Pasture	6	3	18	0	0	271	6	0	5	0
PermanentCrop	9	0	56	10	3	3	312	1	1	0
Residential	0	0	3	0	9	0	0	441	0	0
River	11	1	2	11	0	7	4	0	318	1
SeaLake	3	2	0	0	0	1	0	0	0	431

(a) Confusion Matrix


Examples of PermanentCrop → HerbaceousVegetation Misclassifications
True: PermanentCrop
Pred: HerbaceousVegetation
True: PermanentCrop
Pred: HerbaceousVegetation
True: PermanentCrop
Pred: HerbaceousVegetation
True: PermanentCrop
Pred: HerbaceousVegetation
True: PermanentCrop
Pred: HerbaceousVegetation
True: PermanentCrop
Pred: HerbaceousVegetation

(b) Examples of misclassified images

Figure 3: Confusion Matrix and Examples of misclassified images

Overall, the evaluation results demonstrate that the CNN model performs robustly on the test set, meeting the target benchmark of 85% accuracy, and offering a reliable tool for land use classification from satellite imagery.

3 Conclusion and Discussion

In this project, I developed a CNN to classify satellite images into ten land use categories using the EuroSAT RGB dataset. Through data augmentation and regularization, the model achieved strong generalization, with a macro-averaged F1 score of 0.89.

The model performed well on distinct classes like Forest and Residential but struggled with visually similar ones such as Permanent Crop and Herbaceous Vegetation, partly due to the limitations of RGB-only data.

These results show that CNNs are effective for land use classification, though future improvements could include using additional spectral bands, more advanced architectures, and interpretability tools like Grad-CAM.

4 Bonus Task 1: MLP vs CNN comparison

4.1 Model

In my experiments, the multi-layer perceptron (MLP) model was designed to accept 64 × 64 RGB images as flattened vectors and to output ten land-use category logits. Concretely, each input image—originally a three-channel tensor of shape (3, 64, 64)—is reshaped into a one-dimensional vector of length $3 \times 64 \times 64 = 12288$ before being passed through a sequence of fully connected layers. The first hidden layer consists of 1024 neurons, the second hidden layer has 512 neurons, and the third hidden layer is composed of 256 neurons. Every linear transformation is followed by a ReLU activation and then by a dropout layer with a dropout probability of 0.5. Finally, the output layer is a fully connected layer mapping the 256-dimensional representation to ten output logits—one per class—without an activation function, since I apply CrossEntropyLoss directly to the raw logits.

During training, I used the Adam optimizer with an initial learning rate of 0.001 and an L2 weight decay of 1×10^{-4}. The batch size was set to 32, and all tensors were moved to GPU when available for efficient computation. To prevent overfitting, I implemented early stopping: after each epoch, I evaluated the model's performance on a held-out validation set and recorded the validation loss. If the validation loss did not improve by at least 1×10^{-4} over five consecutive epochs (our patience value), training was halted and the model weights corresponding to the best observed validation loss were restored. Without these dropout and early stopping mechanisms, the fully connected network—which has over two million parameters in total—quickly overfits the relatively small EuroSAT dataset.

In the forward pass, every batch of images is flattened on the fly inside the model's forward method, so no manual tensor reshaping is required in the training loop. At the end of each epoch, we record both the average training loss and the average validation loss to monitor convergence. After training completes, we switch the model into evaluation mode—disabling dropout—and compute the final test loss and accuracy over the unseen test set. This MLP configuration, with its three hidden layers, 50% dropout at each hidden layer, Adam optimization ($\text{lr} = 0.001$, $\text{weight\_decay} = 1 \times 10^{-4}$), and early stopping ($\text{patience} = 5$, $\text{min\_delta} = 1 \times 10^{-4}$), can be implemented directly in PyTorch using the standard nn.Sequential, nn.Linear, nn.ReLU, nn.Dropout, and torch.optim.Adam classes, along with a simple training loop that checks validation loss at each epoch.

4.2 Training

The MLP's loss curves (Figure 4) indicate a quick decrease in both training and validation losses during the first few epochs, followed by a gradual plateau. Throughout, validation loss remains below training loss—evidence that 50% dropout, a 0.001 learning rate, and L2 weight decay effectively regularize the network. Because loss improvements become minimal after early epochs, early stopping was triggered at epoch 8. Despite this stabilization, the MLP's absolute loss values remain relatively high compared to a well-tuned CNN, reflecting its limited capacity to capture spatial patterns from flattened inputs.

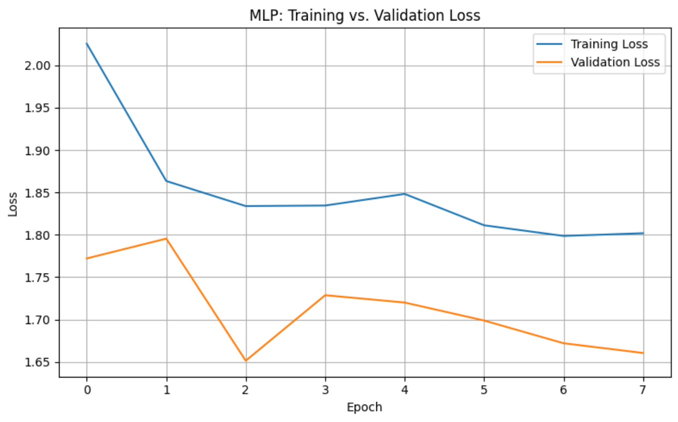
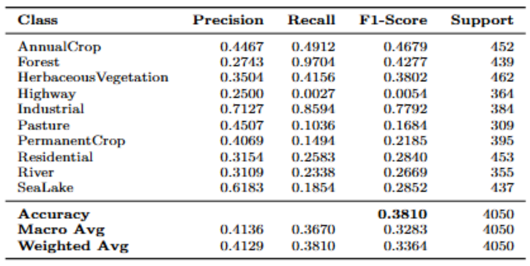


Figure 4: Loss curves for MLP model

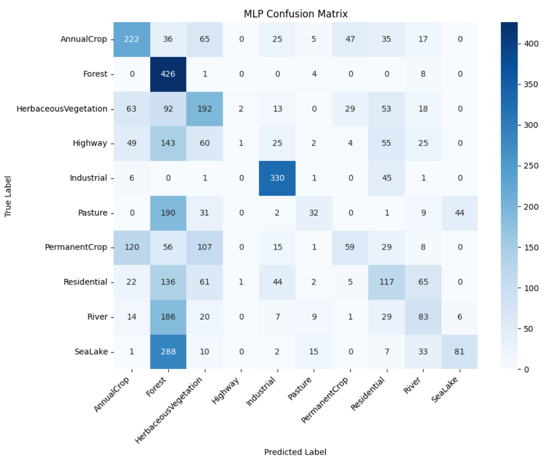
4.3 Evaluation and Comaprison of CNN and MLP

The MLP's test performance was far below that of the CNN: overall accuracy was only 38.10% with a macro-averaged F1 score of 0.3283. Although the MLP could detect some classes—"Industrial," for example, achieved recall = 0.8594 and F1 = 0.7792—most categories suffered. "River" and "SeaLake" showed especially poor recall (0.2338 and 0.1854, respectively), and "Pasture" had the lowest F1 (0.1664). The confusion matrix makes this clear: many "SeaLake" images were misclassified as "Forest" (288 cases), "Pasture" as "Forest" (190 cases), "PermanentCrop" as "HerbaceousVegetation" (107 cases), and "Highway" as "Forest" (143 cases). Because the MLP flattens all 64 × 64 pixels into a single vector, it cannot capture local textures or spatial arrangements, causing visually similar vegetation or land-cover classes to collapse into the same "confident but incorrect" predictions. In short, while the MLP learned a few discriminative signals, its inability to exploit two-dimensional structure makes it fundamentally unsuited for complex land-use classification, a task for which a convolutional architecture is far better equipped.



Class	Precision	Recall	F1-Score	Support
AnnualCrop	0.4467	0.4912	0.4679	452
Forest	0.2743	0.9704	0.4277	439
HerbaceousVegetation	0.3504	0.4156	0.3802	462
Highway	0.2500	0.0027	0.0054	364
Industrial	0.7127	0.8594	0.7792	384
Pasture	0.4507	0.1036	0.1684	309
PermanentCrop	0.4069	0.1494	0.2185	395
Residential	0.3154	0.2583	0.2840	453
River	0.3109	0.2338	0.2669	355
SeaLake	0.6183	0.1854	0.2852	437
Accuracy			**0.3810**	4050
Macro Avg	0.4136	0.3670	0.3283	4050
Weighted Avg	0.4129	0.3810	0.3364	4050

(a) Classification Report by MLP



	AnnualCrop	Forest	HerbaceousVegetation	Highway	Industrial	Pasture	PermanentCrop	Residential	River	SeaLake
AnnualCrop	222	36	65	0	25	5	47	35	17	0
Forest	0	426	1	0	0	4	0	0	8	0
HerbaceousVegetation	63	92	192	2	13	0	29	53	18	0
Highway	49	143	60	1	25	2	4	55	25	0
Industrial	6	0	1	0	330	1	0	45	1	0
Pasture	0	190	31	0	2	32	0	1	9	44
PermanentCrop	120	56	107	0	15	1	59	29	8	0
Residential	22	136	61	1	44	2	5	117	65	0
River	14	186	20	0	7	9	1	29	83	6
SeaLake	1	288	10	0	2	15	0	7	33	81

(b) Confusion Matrix by MLP

Figure 5

5 Bonus Task 2: Feature visualization and interpretation

I performed feature visualization on the final convolutional block using activation maps. This approach allows us to inspect the internal representations the model builds to distinguish between different land use categories.

5.1 Activation Extraction and Visualization

I registered a forward hook on the third MaxPool2d layer (features[16]) of our trained EuroSAT_CNN, capturing its 128 × 8 × 8 activation tensor whenever an image is passed through the network. By plotting the first eight channels of these 8 × 8 feature maps, I observe that each filter appears to respond to different visual cues. For example, in the "AnnualCrop" row, Channel 2 and Channel 4 light up on linear field boundaries, whereas Channels 0, 1, and 3 remain nearly dark—indicating those filters are not sensitive to large uniform crop areas. In the "SeaLake" row, Channel 2 produces a roughly uniform activation across the water region, showing that this filter is detecting large homogeneous dark patches consistent with water. Meanwhile, "Industrial" examples often activate Channel 0 and Channel 4 on bright rectangular roof patterns, suggesting those filters encode sharp edges and grid-like man-made structures. Collectively, these maps reveal that the final convolutional layer has learned mid-to-high-level features such as vegetation texture, linear boundaries, and repeated geometric patterns rather than simply reacting to raw color values.

5.2 Class-Specific Activation Examples and Interpretation

Figure 6 presents one test image per class alongside its corresponding eight activation channels. In the "HerbaceousVegetation" example, Channel 4 highlights a small patch of homogeneous soil or bare earth, while Channel 6 activates on denser green clusters—likely distinguishing bushy vegetation from open fields. For "Highway," Channel 2 and Channel 6 both exhibit nonzero activations precisely along the winding road, confirming that those filters detect elongated linear structures (asphalt lanes). In the "Forest" example, Channel 2 shows a localized bright region atop an otherwise dark map, indicating that that filter is sensitive to dense, fractal-like tree canopies. Similarly, in "Residential," Channel 6 exhibits broad, multi-colored activations over the grid of rooftops, implying that filter encodes small rectangular features of urban housing. Finally, "Pasture" images light up Channels 2 and 6 over irregular grassy regions, suggesting those filters pick up on moderately textured pastureland versus uniform cropland or water. By inspecting these exemplars, I can see how the network's learned filters correlate with semantic patterns: water bodies, roads, tree canopies, and built environments each produce distinct channel activations.

5.3 Insights into Model Decisions from Activation Maps

These activation-map visualizations serve as a qualitative sanity check: they confirm that the CNN is not memorizing arbitrary pixel positions but instead responding to semantically meaningful visual structures. For instance, when the network classifies an image as "SeaLake," we consistently observe coherent activations across one or two specific channels wherever a large, uniform blue region appears. Likewise, correct "Industrial" predictions coincide with filters that sharply activate on rectangular building rooftops. Conversely, when "PermanentCrop" is misclassified as

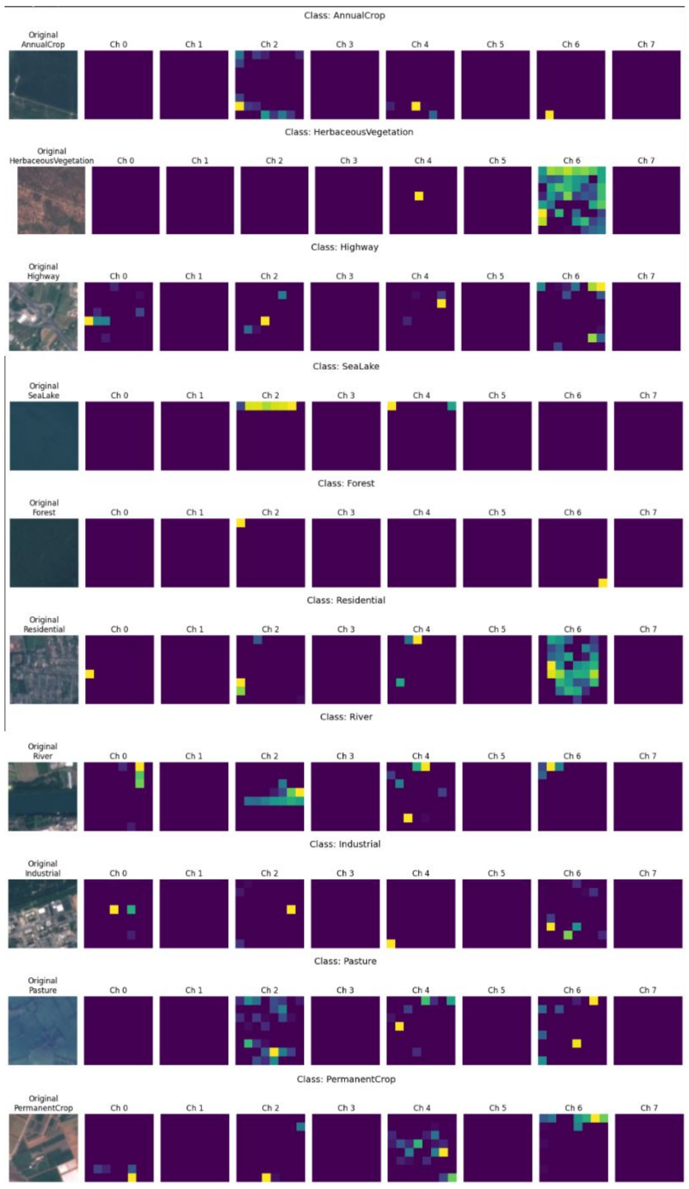


Figure 6: Feature visualization

"HerbaceousVegetation," the activation maps for both classes frequently overlap—both show similar mid-tone patch activations (Channels 4 or 6) that blur the boundary between cultivated fields and natural vegetation. By comparing these heatmaps, we can pinpoint exactly which filters misfire and lead to confusion. In effect, seeing channels "light up" over particular pixel clusters lets us trace a classification decision back to the visual features that most strongly influenced it—for example, a filter detecting repeated linear furrows in cropland or one detecting fractal canopy patterns in forest. In summary, these visualizations demystify the black-box by showing that the final conv layer's filters indeed capture the kinds of land-use-relevant patterns that drive the CNN's high classification accuracy.