Contents

1 Problem Statement **2**

2 Baseline CNN **2**
2.1 Split dataset 2
2.2 Build CNN model 3
2.3 Dataloader 4
2.4 Training 5
2.5 Prediction on test set 7

3 Apply improvement **8**
3.1 Data augmentation 8
3.2 Dataloader 8
3.3 Batch normalization, more CNN layer 8
3.4 Dropout, Adam optimiser 9
3.5 Training improved 9
3.6 Prediction on test set 11
3.7 Ensemble model 12

4 AlexNet **13**
4.1 Data Loading and Preprocessing 13
4.2 Fine-Tuning the Final Layer of AlexNet 13
4.3 Directed Acyclic Graph SVM (DAGSVM) 13
4.4 Results of Fine-Tuning the Final Layer 14
4.5 Results of SVM Classification on AlexNet Features 14

Computer Vision Report

Leonardo Angellotti

December 2025

1 Problem Statement

The goal of this project is to design and implement an image classification system using a Convolutional Neural Network (CNN). The dataset provided, originating from Lazebnik et al. (2006), consists of grayscale images of size 64×64×1 categorized into 15 distinct scene classes (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb).
The dataset is already separated into training and test sets.

The task is to build a CNN based on a specified architecture that includes multiple convolutional layers, ReLU activations, max-pooling operations, a fully connected layer, and a softmax output layer.
The network must learn to correctly classify images into one of the 15 categories using cross-entropy loss.

The main challenge is to implement the architecture accurately, train it effectively on the given dataset, and evaluate its performance on the test set.
Success will be measured by the classifier's accuracy and its ability to generalize across different scene types.

2 Baseline CNN

2.1 Split dataset

To ensure proper model tuning and to prevent overfitting, the provided training set was further divided into two subsets.
Specifically, 85% of the original training data was used as the actual training set, while the remaining 15% was allocated as a validation set.

The validation set was used during training to monitor the network's performance on unseen data, guide hyperparameter adjustments, and determine early stopping criteria.

This split allowed for a more reliable assessment of the model's generalization ability before evaluating it on the separate test set.

2.2 Build CNN model

The following architecture is implemented using pytorch.

#	type	size
1	Image Input	64×64×1 images
2	Convolution	8 3×3 convolutions with stride 1
3	ReLU	
4	Max Pooling	2×2 max pooling with stride 2
5	Convolution	16 3×3 convolutions with stride 1
6	ReLU	
7	Max Pooling	2×2 max pooling with stride 2
8	Convolution	32 3×3 convolutions with stride 1
9	ReLU	
10	Fully Connected	15
11	Softmax	softmax
12	Classification Output	crossentropyex

To train this architecture, two helper functions were defined.

The `train_one_epoch` routine performs a complete pass over the training data loader: for each mini-batch it moves data to the appropriate device (CPU or GPU), computes the forward pass, evaluates the cross-entropy loss, performs backpropagation, and updates the model parameters using the chosen optimizer (SGD). It also accumulates the average training loss and accuracy over the epoch.

The `evaluate` function mirrors this pipeline but runs with gradients disabled and without weight updates, providing the average loss and accuracy on a given validation or test loader.

2.3 Dataloader

The dataset was loaded from the directory structure containing the training, validation, and test sets. All images were resized to 64×64 pixels, converted to grayscale, and transformed into PyTorch tensors.
The data were then wrapped into `DataLoader` objects with a batch size of 32.

Here reported a printing results of preprocessing:

```
Number of training images:  1275
Batch image tensor shape:  torch.Size([32, 1, 64, 64])
Batch label tensor shape:  torch.Size([32])
Single image shape:  torch.Size([1, 64, 64])
```

The training set contained a total of 1275 images after preprocessing.
The data loaders generated mini-batches of 32 samples, where each batch consisted of image tensors with shape `[32, 1, 64, 64]`, corresponding to 32 grayscale images of size 64×64.
The associated label batch had shape `[32]`, containing one class index per image. A single preprocessed image therefore had tensor dimensions `[1, 64, 64]`, indicating a single-channel (grayscale) input.

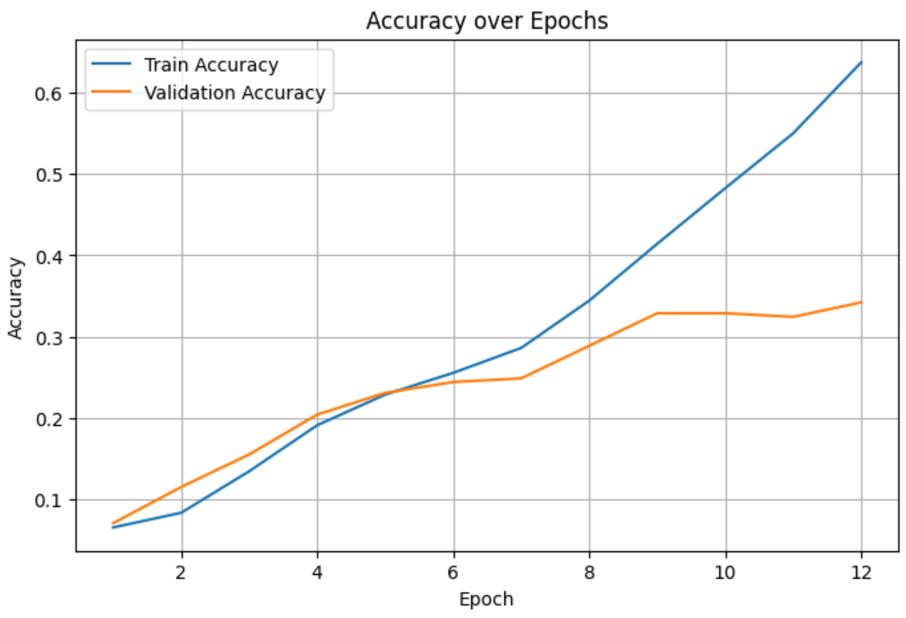
Here the images for train and validation set plotted:

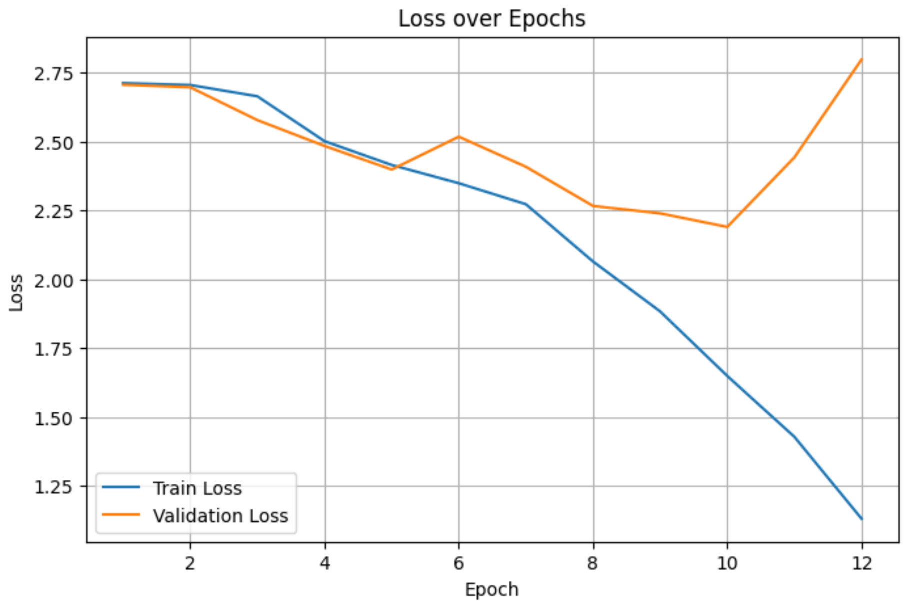
2.4 Training

The model was trained using stochastic gradient descent with cross-entropy loss. At each epoch, the network was optimized on the training set and subsequently evaluated on the validation set, recording both **loss** and **accuracy**.
Early stopping was employed with a patience of 10 epochs: whenever the validation loss improved, the model parameters were saved; otherwise, a counter was incremented until the patience threshold was reached, at which point training halted.

After completion, the best-performing model (lowest validation loss) was reloaded. Finally, the training and validation accuracy and loss curves were plotted to visualize the learning dynamics and detect potential overfitting.

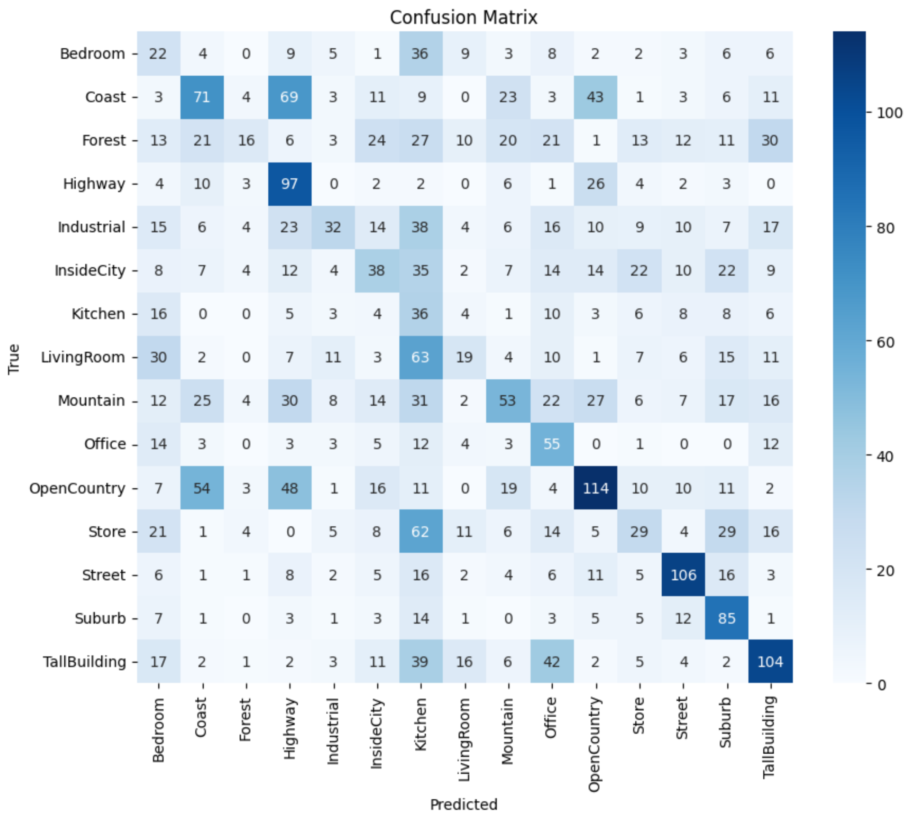
Here reported the plots of **accuracy** and **Loss** of the model:





2.5 Prediction on test set

After training, the best model was evaluated on the test set. For each test batch, the network's predicted class was recorded and compared to the ground-truth labels to compute the overall test accuracy.
A confusion matrix was then generated to visualize per-class performance, highlighting common misclassifications.



True \ Predicted	Bedroom	Coast	Forest	Highway	Industrial	InsideCity	Kitchen	LivingRoom	Mountain	Office	OpenCountry	Store	Street	Suburb	TallBuilding
Bedroom	22	4	0	9	5	1	36	9	3	8	2	2	3	6	6
Coast	3	71	4	69	3	11	9	0	23	3	43	1	3	6	11
Forest	13	21	16	6	3	24	27	10	20	21	1	13	12	11	30
Highway	4	10	3	97	0	2	2	0	6	1	26	4	2	3	0
Industrial	15	6	4	23	32	14	38	4	6	16	10	9	10	7	17
InsideCity	8	7	4	12	4	38	35	2	7	14	14	22	10	22	9
Kitchen	16	0	0	5	3	4	36	4	1	10	3	6	8	8	6
LivingRoom	30	2	0	7	11	3	63	19	4	10	1	7	6	15	11
Mountain	12	25	4	30	8	14	31	2	53	22	27	6	7	17	16
Office	14	3	0	3	3	5	12	4	3	55	0	1	0	0	12
OpenCountry	7	54	3	48	1	16	11	0	19	4	114	10	10	11	2
Store	21	1	4	0	5	8	62	11	6	14	5	29	4	29	16
Street	6	1	1	8	2	5	16	2	4	6	11	5	106	16	3
Suburb	7	1	0	3	1	3	14	1	0	3	5	5	12	85	1
TallBuilding	17	2	1	2	3	11	39	16	6	42	2	5	4	2	104

```
Test Accuracy:  29.38%
```

3 Apply improvement

3.1 Data augmentation

To increase the variability of the training data and improve the model's generalization, data augmentation was applied exclusively to the training set. After separating 15% of the original images from each class into a validation set, the remaining original training images were augmented using three transformations:

(i) horizontal flipping
(ii) a random rotation between -15° and 15°
(iii) a center crop retaining 90% of the original area

Each transformation generated a new image file, resulting in three augmented samples per training image. The validation set remained untouched and contained only the original images to ensure an unbiased evaluation.

3.2 Dataloader

Here the new augmented train dataset loaded:

```
Number of training images:  5100
Batch image tensor shape:  torch.Size([32, 1, 64, 64])
Batch label tensor shape:  torch.Size([32])
Single image shape:  torch.Size([1, 64, 64])
```

3.3 Batch normalization, more CNN layer

The CNN architecture was extended by adding **batch normalization layers** after each convolution. Batch normalization stabilizes the learning process by normalizing intermediate feature maps, reducing internal covariate shift and allowing for higher learning rates.

In addition to the original structure, **several deeper convolutional blocks were introduced**, using larger kernels (5×5 and 7×7) to increase the receptive field and enable the network to capture broader spatial patterns within the input scenes.

3.4 Dropout, Adam optimiser

Dropout was incorporated after the convolutional blocks and before the fully connected layer to reduce overfitting.

A light spatial dropout (5%) was applied after each convolutional block, while a stronger dropout (10%) was used before the classification layer, where overfitting is typically more pronounced.

The model was optimized using the **Adam optimizer**.

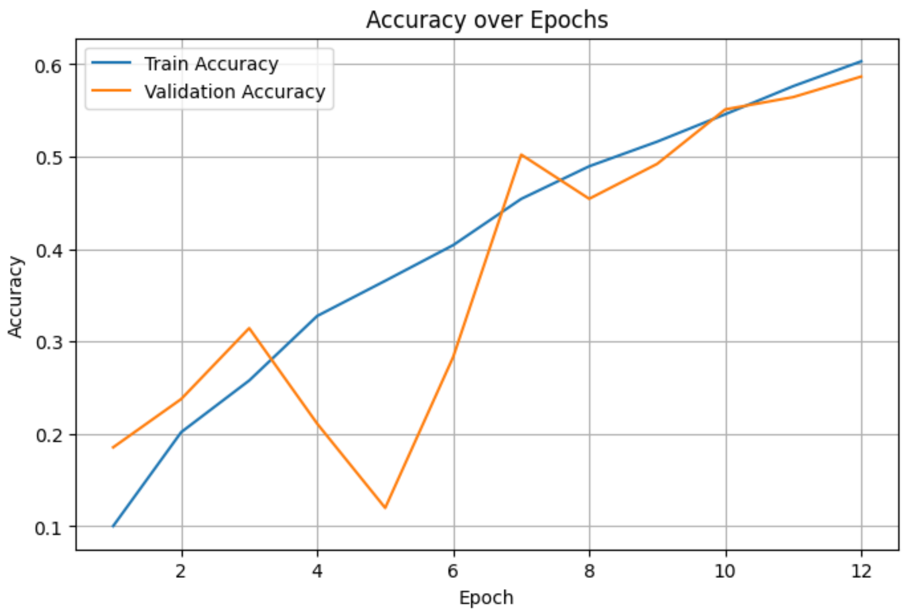
Compared to standard SGD, Adam requires less hyperparameter tuning and accelerates training, particularly when the loss landscape is noisy or gradients vary across layers.

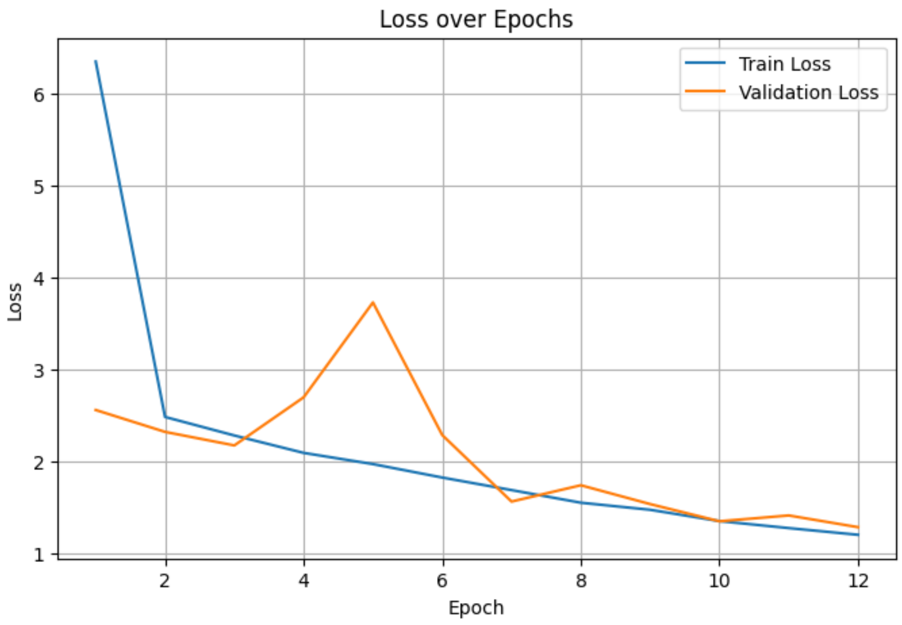
3.5 Training improved

After improvement, the training and validation curves show a clear improvement in stability and overall performance.

The training accuracy increases steadily across epochs, as the validation, reaching approximately 60% by the end of training.

The training as validation loss decreases monotonically better than before.
This suggests that the model generalizes better compared to the baseline architecture, benefiting from the combination of normalization, regularization, and deeper feature extraction.


Accuracy over Epochs
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.1
0.2
0.3
0.4
0.5
0.6
2
4
6
8
10
12


Loss over Epochs
Train Loss
Validation Loss
Loss
Epoch
1
2
3
4
5
6
2
4
6
8
10
12

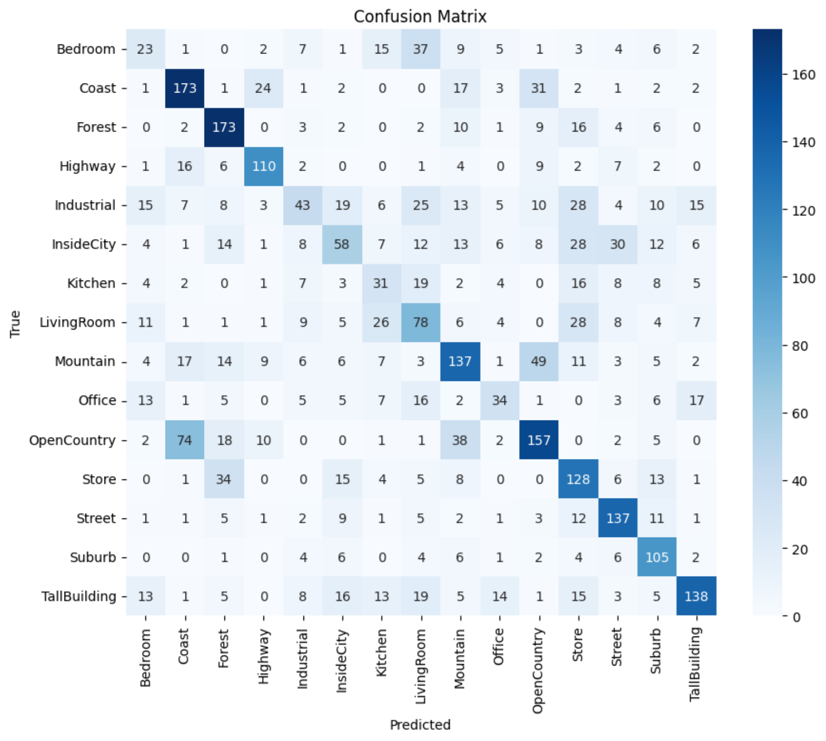
3.6 Prediction on test set

The improved model achieved a test accuracy of 51.09%, representing a substantial increase over the baseline architecture.
The confusion matrix shows that several scene categories are now classified with significantly higher reliability.

In particular, classes such as *Coast*, *Forest*, *Mountain*, *Open Country*, *Store*, and *Tall Building* exhibit strong diagonal dominance, indicating that the model has learned distinctive and discriminative features for these environments.

Some confusion persists among visually similar indoor categories—such as *Kitchen*, *Living Room*, and *Office*—as well as between *Bedroom* and *Suburb*, reflecting the inherent difficulty of separating classes with overlapping textures or spatial layouts.

Nevertheless, the overall structure of the matrix demonstrates clearer class boundaries and markedly fewer severe misclassifications compared to the baseline model.

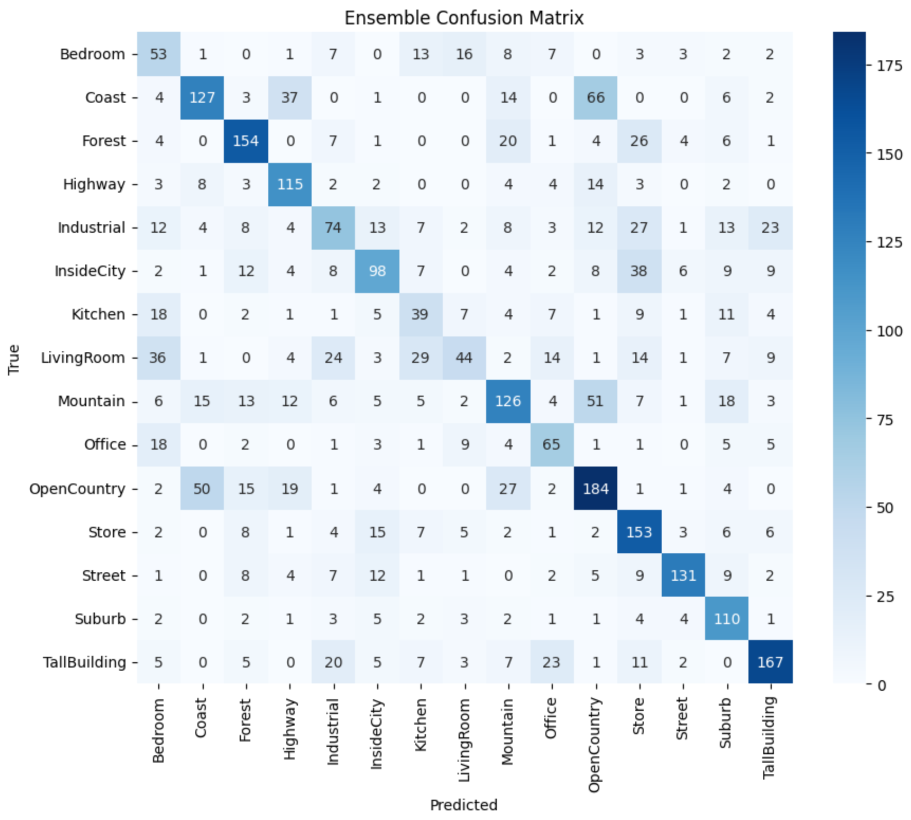


```
Test Accuracy: 51.09%
```

3.7 Ensemble model

To improve generalization performance, we trained an ensemble of **four CNN models**.
Each model was trained independently with early stopping based on validation loss, and the best checkpoint for each was saved.
At test time, the ensemble prediction was obtained by averaging the class logits of the four networks and selecting the class with the highest averaged logit.

This 4-model ensemble achieved a test accuracy of 54.94%, compared to 51.09% for the best single model, confirming that ensembling helps reduce variance and improves robustness of the classifier



True \ Predicted	Bedroom	Coast	Forest	Highway	Industrial	InsideCity	Kitchen	LivingRoom	Mountain	Office	OpenCountry	Store	Street	Suburb	TallBuilding
Bedroom	53	1	0	1	7	0	13	16	8	7	0	3	3	2	2
Coast	4	127	3	37	0	1	0	0	14	0	66	0	0	6	2
Forest	4	0	154	0	7	1	0	0	20	1	4	26	4	6	1
Highway	3	8	3	115	2	2	0	0	4	4	14	3	0	2	0
Industrial	12	4	8	4	74	13	7	2	8	3	12	27	1	13	23
InsideCity	2	1	12	4	8	98	7	0	4	2	8	38	6	9	9
Kitchen	18	0	2	1	1	5	39	7	4	7	1	9	1	11	4
LivingRoom	36	1	0	4	24	3	29	44	2	14	1	14	1	7	9
Mountain	6	15	13	12	6	5	5	2	126	4	51	7	1	18	3
Office	18	0	2	0	1	3	1	9	4	65	1	1	0	5	5
OpenCountry	2	50	15	19	1	4	0	0	27	2	184	1	1	4	0
Store	2	0	8	1	4	15	7	5	2	1	2	153	3	6	6
Street	1	0	8	4	7	12	1	1	0	2	5	9	131	9	2
Suburb	2	0	2	1	3	5	2	3	2	1	1	4	4	110	1
TallBuilding	5	0	5	0	20	5	7	3	7	23	1	11	2	0	167

```
Test Accuracy:  54.94%
```

4 AlexNet

In this assignment, we investigate the use of transfer learning for image classification by leveraging a **pre-trained convolutional neural network, specifically AlexNet**.
The system is expected to achieve a test accuracy above 85%.

4.1 Data Loading and Preprocessing

To prepare the dataset for transfer learning with AlexNet, we implemented a dedicated data-loading pipeline. Since AlexNet expects input images of size 224×224 with three colour channels, all grayscale images were resized accordingly and converted to RGB format.

All images were normalised using the standard ImageNet mean and standard deviation values to ensure compatibility with the pre-trained network.

4.2 Fine-Tuning the Final Layer of AlexNet

We fine-tuned only the final fully connected layer of a pre-trained AlexNet model.

All convolutional and intermediate classifier layers were frozen, ensuring that only the last classification layer was updated during training.
The original final layer was replaced with a new linear layer with 15 output units, corresponding to the number of target classes.

The model was trained using stochastic gradient descent with momentum, while cross-entropy loss was used as the optimisation objective.
The best-performing model on the validation set was saved and subsequently evaluated on the test set to obtain the final accuracy.

4.3 Directed Acyclic Graph SVM (DAGSVM)

To extend binary Support Vector Machines to the multiclass setting, we implemented a Directed Acyclic Graph SVM (DAGSVM).

The method relies on training all one-vs-one classifiers for every pair of classes. For K classes, this results in $K(K-1)/2$ binary SVMs. Each classifier is trained only on samples from the two classes it distinguishes.

During prediction, the model performs a sequence of pairwise comparisons arranged in a directed acyclic graph.
Starting with the full set of classes, the algorithm evaluates the classifier corresponding to the first and last remaining classes.
The predicted class survives, while the other is eliminated.

This elimination process continues until only one class remains, which is returned as the final prediction.

4.4 Results of Fine-Tuning the Final Layer

The experiment in which only the final fully connected layer of AlexNet was fine-tuned achieved strong performance.
Training accuracy increased steadily throughout the ten epochs, reaching a final value of $\approx 95.33\%$, indicating that the newly added classifier adapted effectively to the task.
Validation accuracy remained consistently high, with a peak of 92.89%, demonstrating good generalisation despite updating only a single layer of the network.

The final test accuracy of 87.04% confirms that the model successfully transferred discriminative features learned from ImageNet to the new dataset.
Although slightly lower than the best validation accuracy, the result remains well above the expected performance threshold of 85%.
Overall, these findings show that even minimal fine-tuning of AlexNet is sufficient to achieve strong classification performance on this task.

```
Best validation accuracy:  92.89%
Test accuracy (fine-tuning last layer):  87.04%
```

4.5 Results of SVM Classification on AlexNet Features

Using AlexNet as a fixed feature extractor followed by a multiclass linear SVM achieved a test accuracy of 85.56%.
This result meets the required performance threshold.

Overall, the SVM approach provides a little worse performance while requiring significantly less training time compared to fine-tuning neural network layers.

```
Test accuracy (SVM on features):  85.56%
```