Land Use and Land Cover ClassificationUsing Satellite Imagery and Deep Learning

A Thesis Submitted

in Partial Fulfillment of the Requirements for the Award of

the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

Submitted by

RAJAN (24/MSCMAT/15)

Under the supervision of

MS. TRASHA GUPTA

DEPARTMENT OF APPLIED MATHEMATICS

DELHI TECHNOLOGICAL UNIVERSITY

(FORMERLY DELHI COLLEGE OF ENGINEERING)

Bawan Road, Delhi 110042

CERTIFICATE

I hereby attest that the project dissertation ***"Land Use and Land Cover Classification using Satellite Imagery and Deep Learning"*** submitted by Rajan (24/MSCMAT/15) of Department of Applied Mathematics, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Masters of Science in Mathematics, is a record of the dissertation work carried out by the student under the supervision of Ms. Trasha Gupta.

To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Ms. Trasha Gupta
Supervisor, Department of Applied Mathematics
Delhi Technological University, Delhi

Acknowledgement

We extend our sincere gratitude to our supervisor, Ms. Trasha Gupta of the Department of Applied Mathematics at Delhi Technological University, for providing invaluable guidance, encouragement, and clarity throughout the duration of this dissertation. Their expertise, constructive feedback, and constant motivation played a pivotal role in shaping the direction and successful completion of this research.

I would like to acknowledge the broader open-source community, specifically the developers of PyTorch, torchvision, and the EuroSAT dataset creators — Helber, Bischke, Dengel, and Borth — whose freely available tools and datasets made this research possible.

We would also like to thank our fellow classmates, colleagues, and faculty members who offered helpful discussions, shared resources, and supported us during this journey.

Finally, we express our heartfelt thanks to our families for their unwavering support and encouragement throughout our academic pursuit.

Rajan

24/MSCMAT/15

Abstract

The precise and effective monitoring of land use and land cover (LULC) is important for urban planning, environmental protection, agriculture, and disaster management. Current approaches for classifying satellite images are dependent on the manual inspection process, which is time-consuming and can cause errors. Hence, a deep learning and computer vision-based solution for efficient LULC classification has been presented in this dissertation.

For the purpose of developing our model, we used the EuroSat data set. It is a publicly available data set generated from the images collected by the Sentinel-2 satellite, which is owned and operated by the European Space Agency. Our deep convolutional neural network (CNN), inspired by the LeNet-5 architecture, is intended to classify satellite images of size 64x64 into ten classes such as Annual crop, Forest, Highway, Residential area, and Water bodies. For implementation, we used PyTorch, the popular deep learning framework, in conjunction with a GPU and CUDA.

As a part of this dissertation, we have also considered the use of transfer learning. For that, we froze all but the last layer of a ResNet-18 CNN trained on the Imagenet database.

It was shown through experiments that the proposed lightweight custom CNN model was able to achieve a validation accuracy of 66.9% in just five training iterations while taking only 2.44 minutes to train on the GPU altogether. The findings show that although deep learning models present themselves as a more reliable option compared to manual classification techniques, the similarity of spectra among vegetation classes cannot be ignored as the source of any possible mistakes.

The findings suggest that small CNN architectures are an excellent choice for achieving real-time environmental monitoring in a more effective manner, and transfer learning with ResNet-18 represents a way forward to obtaining better quality. In total, these two concepts present a great platform for making further improvements using the multispectral analysis method and pixel-wise semantic segmentation techniques. (Keywords: Land Use and Land Cover (LULC), Convolutional Neural Network (CNN), EuroSAT, Transfer Learning, ResNet-18, PyTorch, Remote Sensing, Sentinel-2, Deep Learning, Image Classification.)

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Overview of Remote Sensing and Computer Vision

Remote sensing refers to the process of collecting information about Earth's surface without making direct physical contact between the sensor and the target. This technique entails the use of sensors mounted on aircraft and satellites, among other platforms, to collect data about earth's surface. Remote sensing has become a necessity within contemporary geosciences as it allows for monitoring of various dynamic processes on Earth's surface, ranging from increasing urban areas, decreasing forests, coastal erosion, and drought-ridden agricultural lands.

The basis for all remote sensing processes is Electromagnetic Radiation (EMR). Every feature on Earth surface absorbs, reflects, or emits EMR at certain wavelengths. Remote-sensing satellites use their sensors to collect EMR from the earth surface, and then process this collected data into images used to determine characteristics of surface objects using the kind of EMR received.

The analysis and interpretation of such images traditionally has been done manually through visual inspection by analysts who classify surface objects based on color, texture, pattern, etc. Although this method can be highly accurate, it is quite inefficient since current earth-observing satellites can capture more than tens of terabytes of data daily.

There have been significant changes in this area owing to the quick development of Computer Vision, which is a branch of Artificial Intelligence aimed at providing the machines with the capability to perceive and interpret the world visually. Computer Vision algorithms can analyze huge amounts of data within milliseconds. Thanks to Deep Learning, Computer Vision algorithms can identify hierarchical features differentiating various types of land cover without having any need for explicit feature design in the case of CNNs. Unlike earlier Computer Vision algorithms analyzing each pixel separately, the modern CNNs possess an ability to recognize higher-order patterns in the image, like the winding structure of a river system, regular grid structure in residential urban environments, or texture of a forest.

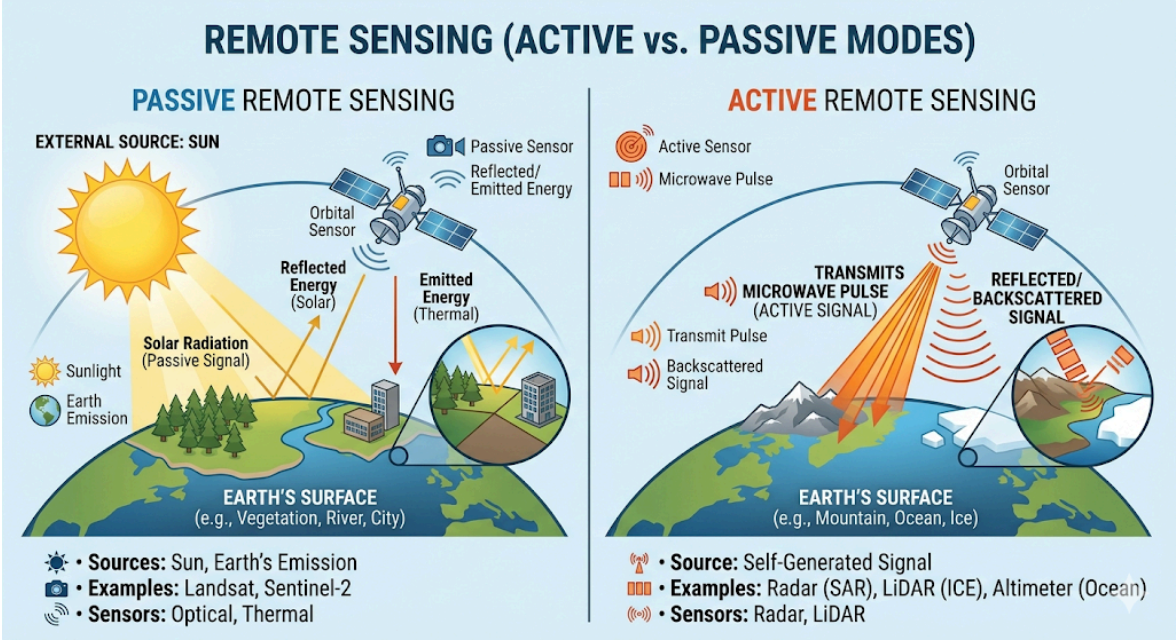


Figure 1.1

1.2 The Role of Earth Observation Satellites (Sentinel-2)

Earth Observation Satellites (EOS) are recognized as the main information resource used for modern geographical studies. Among EOS that can be distinguished as particularly essential for the present study is Sentinel-2, which is a satellite created by the European Space Agency (ESA) as a component of the Copernicus Programme. In this regard, the Copernicus Program involves the creation of two satellites named Sentinel-2A and Sentinel-2B, both equipped with MSI (Multi-Spectral Instrument). It allows capturing images with the maximum resolution of up to 10 meters of a single pixel. In other words, the resolution would be 10 × 10 meters for each individual pixel, which means that there is enough clarity to recognize roads, buildings, and even particular farms within the captured images. Besides, the satellite orbits the Earth once every five days when viewed from the equatorial line. This means that regular monitoring is possible for each and every point of the globe. Moreover, what makes Sentinel-2 ideal for the research is that no license is required to gain access to the satellite's data.

The Sentinel-2 platform is specifically relevant to this project due to its critical role in constructing the EuroSAT dataset. The EuroSAT dataset uses the RGB spectral bands (Bands 4, 3, and 2 representing the Red, Green, and Blue bands respectively) in Sentinel-2 satellite imagery to

provide a consistent ground-truthed reference point for land cover classification. The use of RGB images enables us to employ computer vision models that were originally designed for natural imagery in a geographical domain.

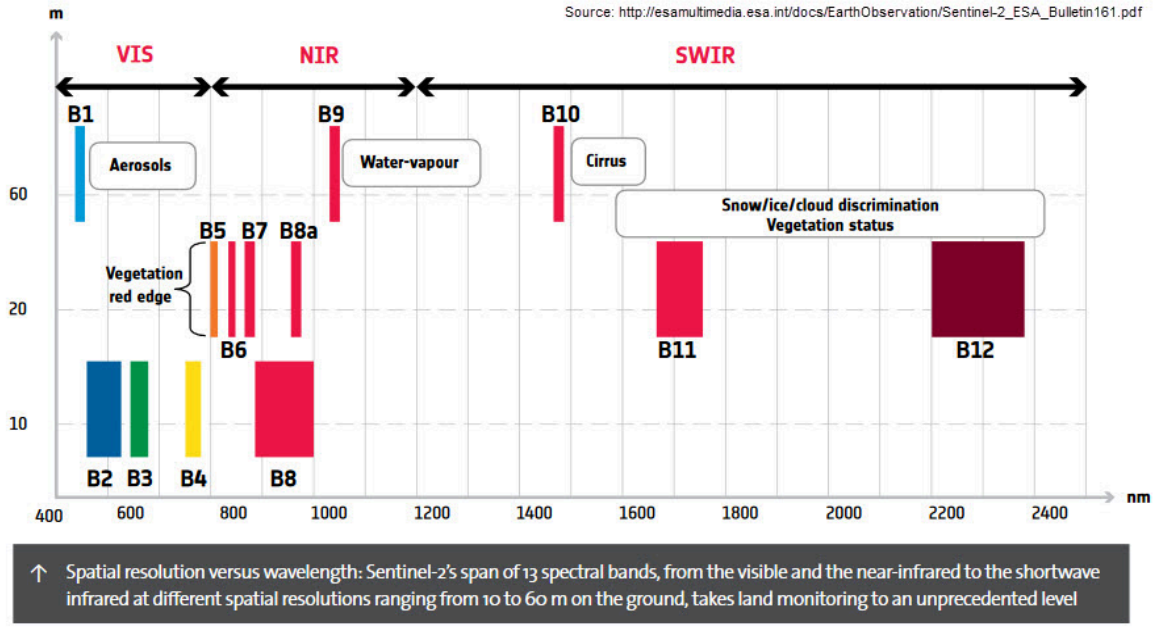


Figure 1.2

1.3 Motivation

Motivation for this work originates from several factors which make automated land cover monitoring a crucial need on a global level.

First, the issue of scale. Modern observation satellites produce several petabytes of data each year. The amount of data collected by even one of the Sentinel-2 satellite system far surpasses what is possible to review manually. The development of deep learning pipeline is the only solution that can turn all of this data into valuable information.

Second, the acceleration of land use changes is taking place. Agricultural land is being converted into urban areas, forests are disappearing and bodies of water are deteriorating at a fast rate. It is necessary to provide policymakers with timely, accurate and geographically extensive data for their work.

First of all, the current state of the art of the deep learning infrastructure, especially the presence of such tools like open-source PyTorch library and the use of the pre-trained models from torchvision, allows the construction of highly performing classification algorithms without significant financial investments. This PhD work shows that good classification performance can be achieved using small custom CNN design trained only for five epochs.

Free access to satellite imagery (Sentinel-2), the EuroSAT dataset, and deep learning infrastructure (PyTorch) opens the door to the democratization of geospatial intelligence, and this project is designed to address precisely this niche.

1.4 Problem Statement

Even though there are currently high-resolution satellite images available, the problem of automated land cover classification using satellite imagery is still considered a challenging task because of the complicated visual appearance and spectral properties of the Earth surface. The following reasons are behind the choice of method described in this thesis.

Firstly, inter-class spectral similarities pose a great challenge for land cover classification. Many classes like "Permanent Crop" vs "Herbaceous Vegetation", or "River" vs "Sea/Lake" possess very similar colors and textures in their RGB versions, which makes it hard for simple classifiers to distinguish between them correctly. Deep learning techniques that allow learning context rather than only taking into account a pixel value are especially suitable to deal with this problem.

Secondly, there is the problem of scale and variability within one class. For example, a residential area can have drastically different appearance depending on geographical location, population density and vegetation.

This project aims to overcome these problems using a well-organized Deep Learning workflow involving all stages of the process, including data collection and preprocessing, modeling, and validation, for the classification of satellite images into ten different LULC classes: Annual Cropland, Forest, Herbaceous Vegetation, Highway, Industrial Area, Pasture, Permanent Cropland, Residential, River, and Sea/Lake.

Chapter 2: Theoretical Foundations

2.1 Deep Learning in Remote Sensing

Deep Learning, being part of Machine Learning, has become the main approach used for image analysis tasks. While classical approaches are based on hand-designed features, including texture descriptors (such as GLCM), spectral indices (such as NDVI), or shape descriptors, deep learning allows automatically constructing a hierarchy of feature spaces using only raw pixels. Such a feat is due to the composition of several layers applying non-linear transformations, thereby learning higher-order abstractions of input data.

As far as satellite imagery is concerned, this ability is especially advantageous. In the case of satellite imagery, there is a variety of spatial relationships, such as the texture of tree canopies, linear features like highways, or the regularity of an industrial area layout. All those characteristics are challenging to code using simple statistical measures. The best type of deep learning model for image processing, namely, Convolutional Neural Network (CNN), extracts those features using a series of filters.

The primary difference compared to conventional machine learning techniques is that in deep learning, there is no need for an expert in the field to decide on the feature that should be extracted. The neural network learns the best features automatically using just the information conveyed through the label data. That is the essence of the strength of deep learning systems.

2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNN) refer to a type of deep neural networks explicitly tailored to processing grid-structured inputs, including images. CNNs have an ability to extract local spatial information through learning filters while keeping track of the relationships among adjacent pixels; this is not possible in a simple fully connected network.

The general structure of a typical CNN model involves feature extraction part, which includes convolutional and pooling layers, and classification part, consisting of fully connected layers. In this work, a customized CNN model will be developed using torch.nn package of the PyTorch framework, whose architecture will be explained in detail.

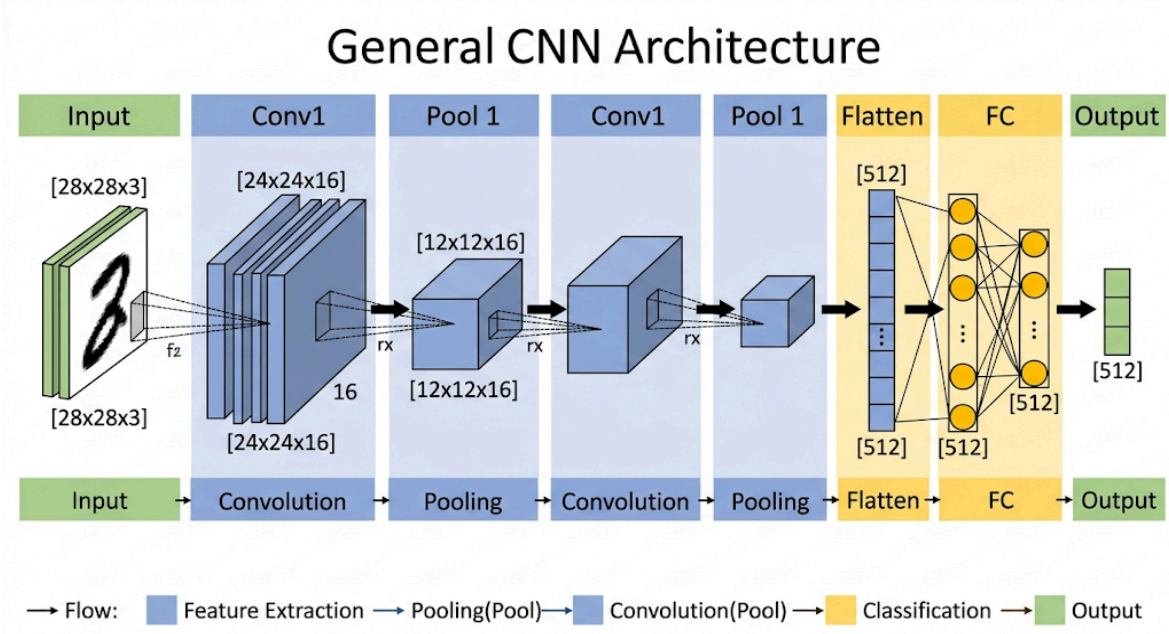


Figure 2.1

2.2.1 Convolutional Layers (nn.Conv2d)

The convolutional layer forms the backbone of CNN architecture since it utilizes a number of trainable filters (kernels) on the input image. These filters are matrices (in our case 3×3 pixels), which are slid over the entire image and compute a dot product at each step. In essence, the resulting map is an indication of the presence of the trained pattern of this kernel on the image (a vertical line, certain color gradient, etc.).

Our network uses the following convolutional operation: nn.Conv2d(3, 6, kernel_size=3). Essentially, the three-channel RGB image gets filtered by 6 distinct filters, thus producing 6 feature maps corresponding to low-level vision operations such as edges, color contrast detection, and corner detection. The resulting size of the output is smaller than input (64×64 → 62×62), as we have not used padding.

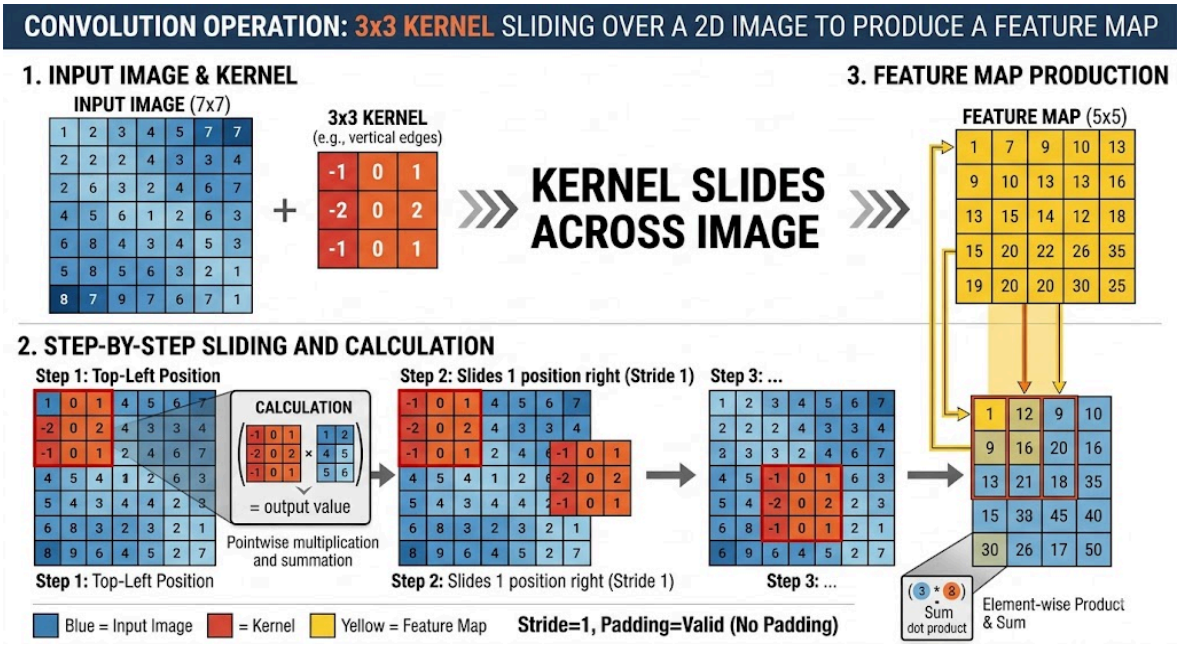


Figure 2.2

2.2.2 Pooling Layers (nn.MaxPool2d)

Pooling layers reduce the spatial dimensions (width and height) of the feature maps, thereby reducing the number of parameters and the computational cost of subsequent layers. They also introduce a degree of translation invariance: if a feature shifts slightly within the image, the pooled output remains the same.

We employ Max Pooling (**nn.MaxPool2d(2, 2)**), which selects the maximum activation value within each non-overlapping 2×2 window. After the first convolutional layer, this reduces the feature map from 62×62 to 31×31. After the second convolutional layer, the 29×29 maps are reduced to 14×14, yielding a final feature volume of 16×14×14 = 3,136 values.

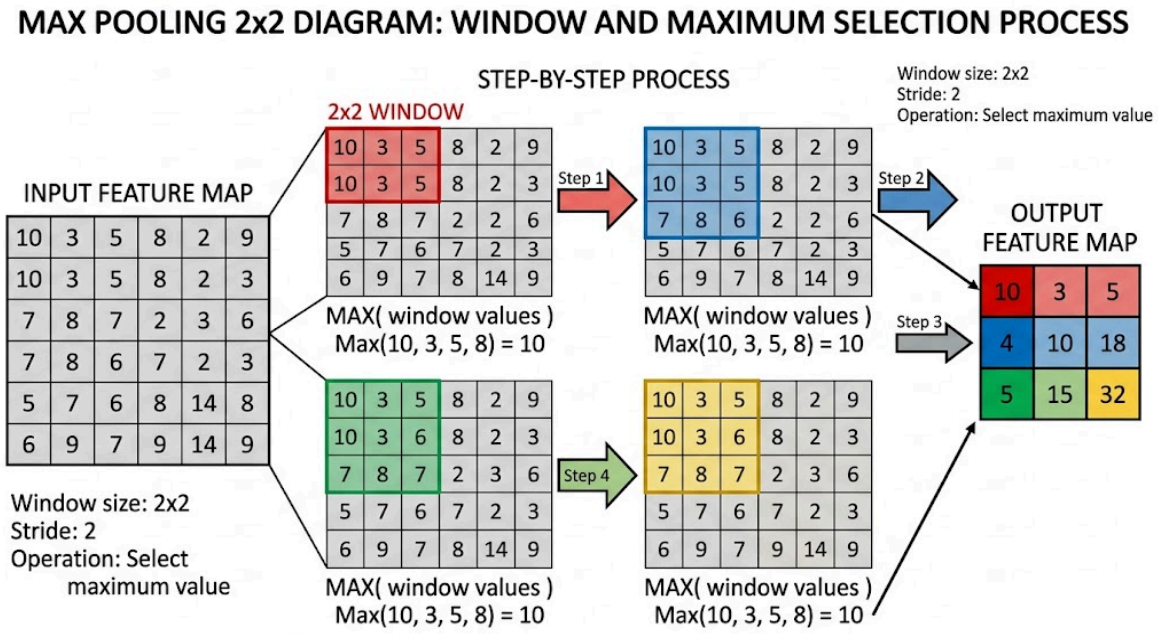


Figure 2.3

2.2.3 Activation Functions (ReLU)

These functions add non-linearity to the network. Absent them, no matter how many layers a network may have, its computations will amount to no more than a linear transformation of the input data, that is, a linear network, which will make it unable to learn any non-linear patterns.

The chosen activation function is called the Rectified Linear Unit (ReLU): f(x) = max(0, x). This function passes on positive activations and clamps negatives to zero. ReLU does not slow down computations, has simple derivatives, and helps prevent the vanishing gradient problem typical of sigmoid and tanh. Function F.relu() is used after every convolutional layer and each fully connected layer except for the last one.

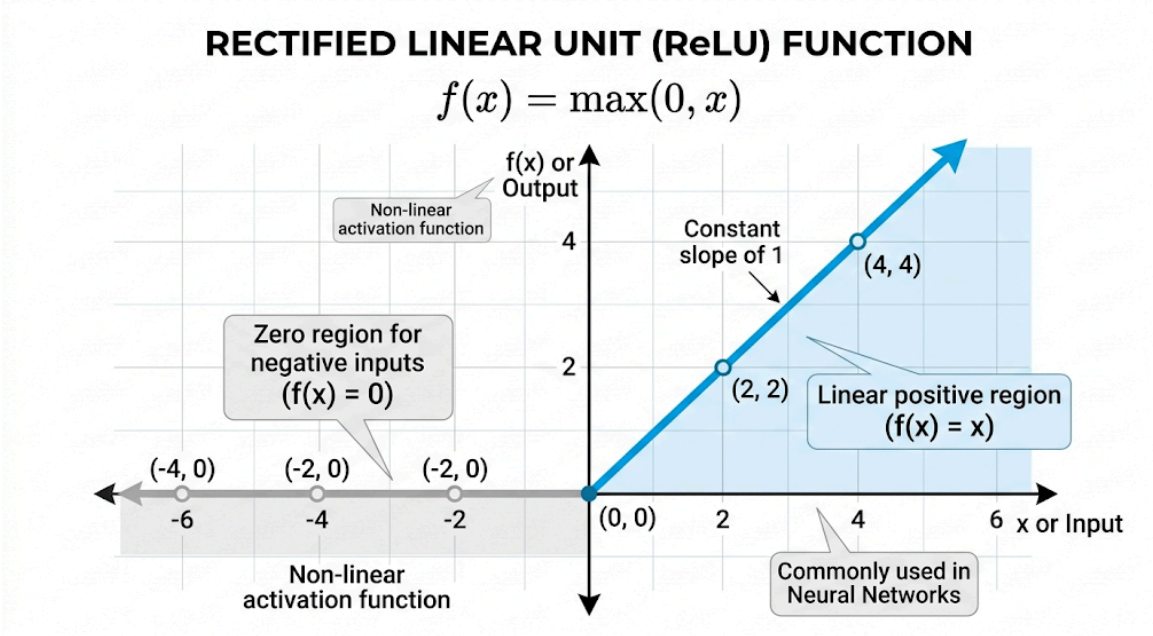


Figure 2.4

2.2.4 Fully Connected Layers (nn.Linear)

Once the feature extraction process is completed, the 3D feature tensor will be flattened into a one-dimensional feature vector. After that, the 3D feature tensor undergoes through the series of fully connected neural network layers called dense layers. All neurons of one layer are connected with every other neuron in the subsequent layers, allowing the combination of all spatial features in the global context of classification.

The FC layers of our model classifier start with 3,136 input features (the number of features after flattening the feature volume), move to 120 hidden units (fc1), then to 84 hidden units (fc2) and finally to 10 output logits (fc3), where the total number equals the number of the EuroSat classes. These outputs are the raw scores representing each class, which pass through CrossEntropyLoss().

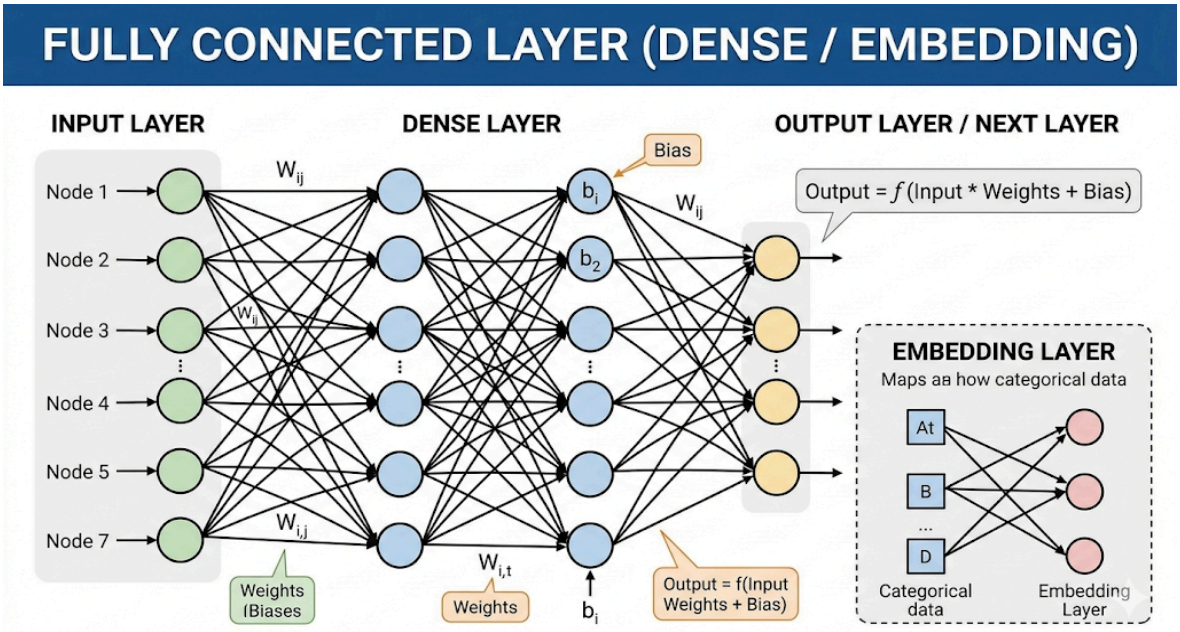


Figure 2.5

2.3 Optimization and Loss Functions

The training of the neural network is basically an optimization process where one tries to minimize the cost function, which measures the difference between the output of the network and the actual labels. This can be done through iterative process by using optimization techniques such as gradients.

2.3.1 Stochastic Gradient Descent (SGD) with Momentum

In Gradient Descent, the model parameters are adjusted in the direction of negative gradient descent. The gradient calculation in gradient descent includes the whole training set at once which makes it very computationally expensive especially when dealing with a huge number of training examples.

For SGD, the approximation of the gradient is computed using some small subset of training examples called minibatches. In our experiment, the size of the minibatch will be equal to 32 images per mini-batch. The noisy approximation of the gradient in SGD serves well in escaping shallow minima and saddle points.

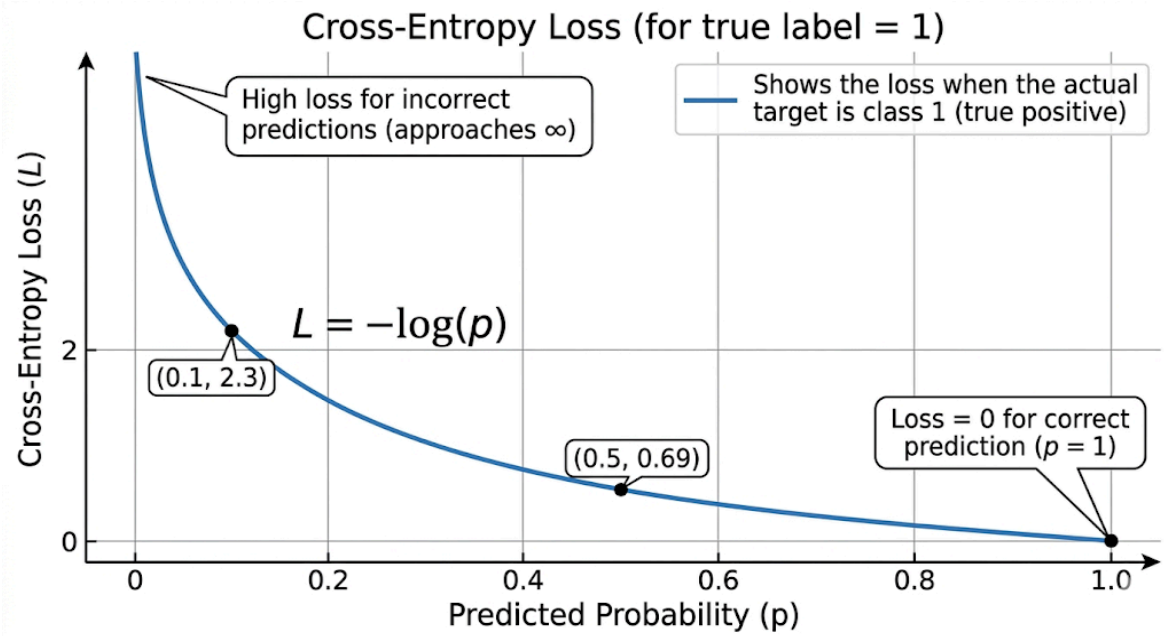
Additionally, the momentum term is set to be 0.9. Momentum adds an inertia effect in the directions of consistent gradient values. Update formula of SGD with momentum is as follows: $v = 0.9*v - lr*\nabla L$, $\theta = \theta + v$

2.3.2 Cross-Entropy Loss

Cross-Entropy Loss is used as the objective function for the case of multi-class classification tasks. It compares the predicted probability distribution, which is calculated using a softmax of logits and the true distribution, which is encoded by an array of zeros except for the index of the correct class that is set as 1.

Formally, it is written for a single sample as **L = -log(p_y)**, where p_y denotes the predicted probability corresponding to the correct class y. The loss can be easily computed using **nn.CrossEntropyLoss()**, which takes care of both the log-softmax and negative log likelihood in PyTorch.

As the name suggests, the cross-entropy loss is zero if the network predicts a value of 1 for the correct class, but grows larger as the model diverges from the truth.



Chapter 3: Related Work

3.1 Evolution of Land Use Classification

The classification of Earth's surface has evolved dramatically over the past several decades. The earliest methods relied entirely on manual field surveys and the visual interpretation of aerial photography by trained cartographers. This work was precise but extraordinarily slow — a single large-area map could take months to produce, and updates required repeat surveys.

The launch of the first operational Earth observation satellites — including the Landsat program, initiated by NASA in 1972 — marked the transition to digital image processing. Early digital methods analyzed satellite imagery on a pixel-by-pixel basis, classifying each pixel solely based on its spectral signature (i.e., its color values across different spectral bands). Algorithms such as maximum likelihood classification, k-nearest neighbors, and minimum-distance classifiers were standard.

These methods, based on pixel values, had a critical flaw: no consideration was taken about the spatial information. It didn’t matter whether a green pixel was representing grass in a park within a city area or was the edge of a forest because, not knowing what other pixels looked like around, you couldn’t tell.

The solution halfway between those two approaches was introduced at the beginning of 2000s, which was the Object-Based Image Analysis (OBIA). The difference from previous ones was that, prior to classifying any object, an image was segmented and its objects were classified not only by their spectra but also by their shapes.

3.2 Traditional Machine Learning vs. Deep Learning

In addition to that, during the early years of the 2000s, machine learning algorithms, especially Support Vector Machines (SVM) and Random Forests (RF), became the prevalent methods used in remote sensing classification, outperforming their statistical alternatives by means of superior accuracy and generalization capability.

Nonetheless, there is one crucial common aspect of both SVM and RF methods; these techniques rely on the prior availability of a feature extraction phase where attributes are derived from the imagery through expert-defined formulas in order to describe it. Examples for these attributes

include GLCM textures, normalized difference vegetation index (NDVI) and water index (NDWI), shape parameters, edge histogram, among others.

Deep Learning represents a fundamental paradigm shift. A Convolutional Neural Network trained end-to-end learns its own feature representations directly from the raw pixel data during the training process, without any human-defined feature engineering. The groundbreaking work of Helber et al. (2019) — the creators of the EuroSAT dataset used in this dissertation — demonstrated that even relatively simple CNN architectures trained on EuroSAT data can achieve classification accuracies exceeding 98%, substantially outperforming traditional ML methods that typically plateau around 80–90% on the same task.

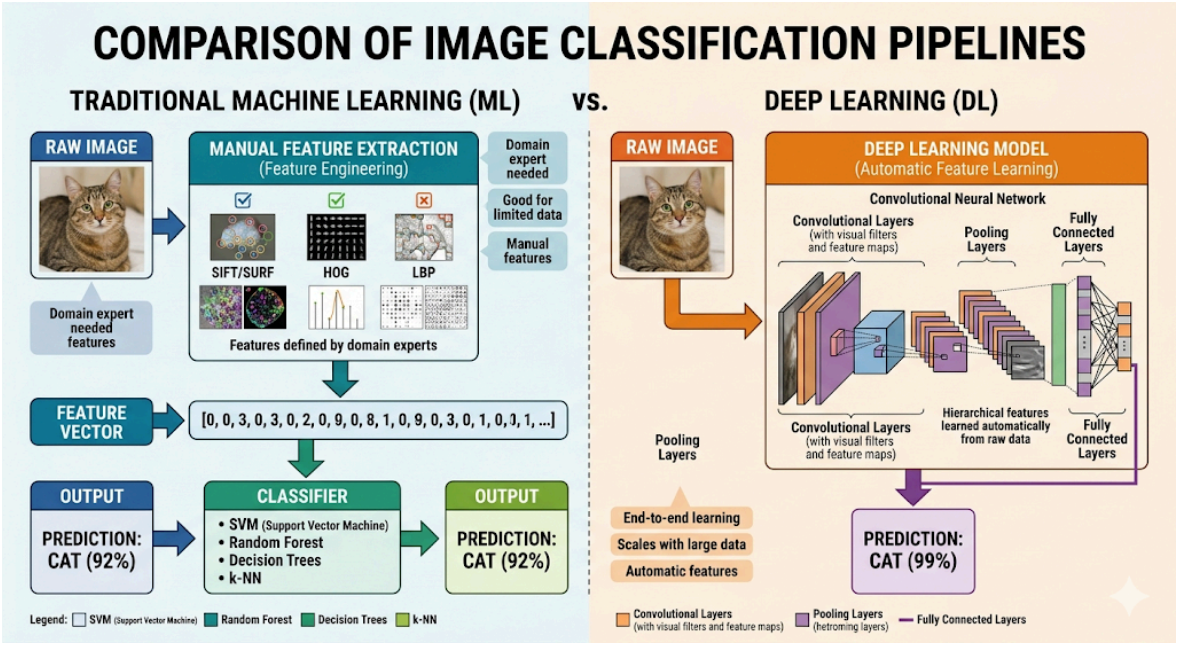


Figure 3.1

3.3 Review of Existing Architectures

Various CNN architectures have been proposed and adapted for remote sensing tasks. This project draws inspiration from two key points on the accuracy-efficiency spectrum.

3.3.1 LeNet-5

The custom MyNN architecture that we are proposing is heavily based on the LeNet-5 architecture created by LeCun et al. (1998). LeNet-5 introduced the basic structure of neural networks, which became a template for most current deep learning architectures; namely, convolution and pooling layers for feature extraction and fully connected layers for classification. This was first used for 32x32 grayscale digits, but the network could be easily scaled up for 64x64 color patches that were used in the EuroSAT dataset.

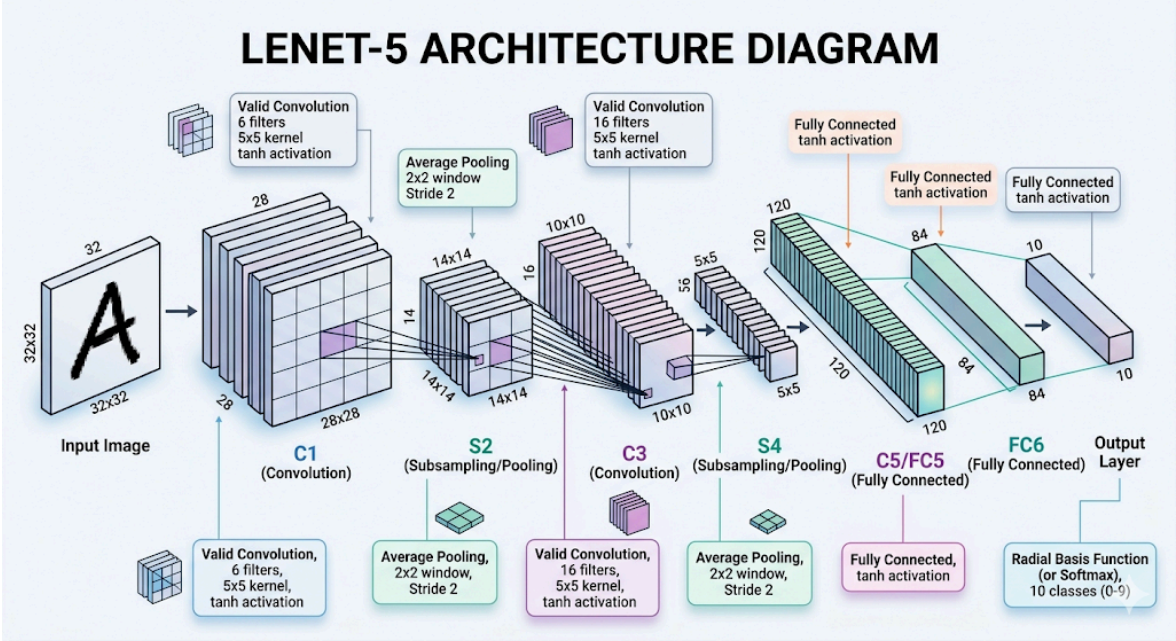


Figure 3.2

3.3.2 VGGNet

The idea of using very deep neural networks that use only small 3×3 convolutional kernels was introduced by VGGNet (Simonyan and Zisserman, 2014). Both VGG-16 and VGG-19 feature 16 to 19 weight layers and offer state-of-the-art performance on ImageNet. In remote sensing studies, the VGG architecture was applied for high-resolution aerial imagery classification. However, its extremely high number of parameters (138 million for VGG-16) makes such networks costly in terms of computational resources and subject to overfitting.

3.3.3 ResNet and Transfer Learning

The issue of vanishing gradient while training very deep neural networks was solved using skip connections by ResNet (He et al., 2016). ResNet-18 is the smallest architecture among the family of ResNets which is made up of 18 layers and 11 million parameters. The pre-trained model of ResNet-18 trained on ImageNet is used as the backbone of the Transfer Learning module of this dissertation. By only freezing the last layer, we utilize ResNet-18's ability to learn better feature representations leading to much improved performance on our LULC task.

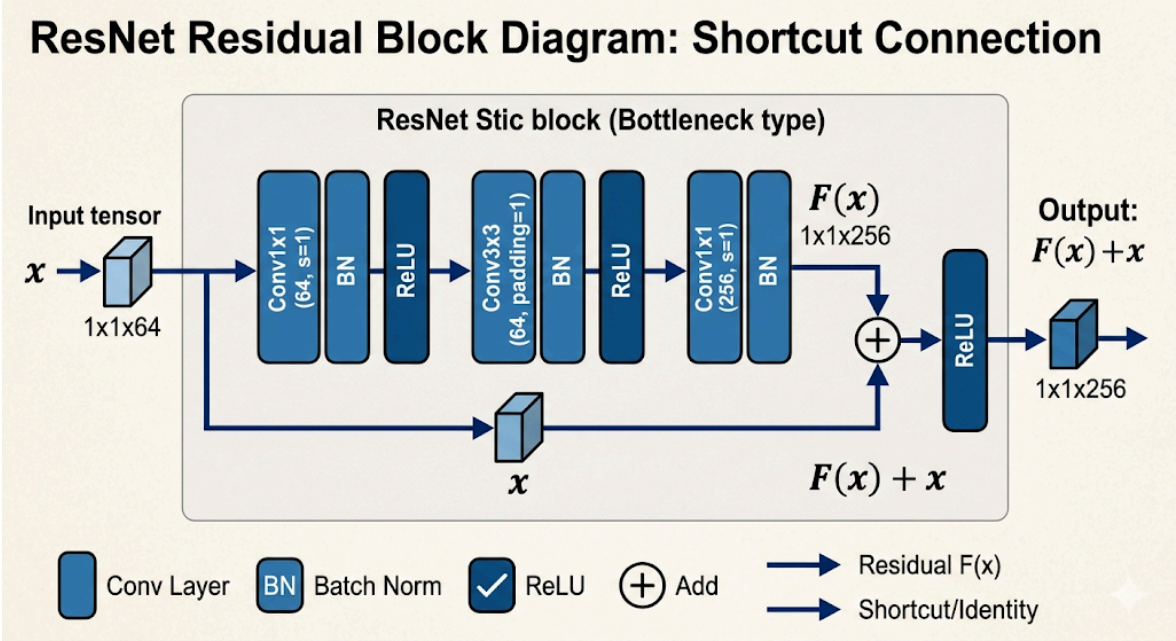


Figure 3.3

Chapter 4: Dataset and Preprocessing

4.1 Dataset Description: EuroSAT

The dataset applied to this study is the EuroSAT dataset that represents a standard benchmark dataset which was particularly created for the application of deep learning in LULC classification. This benchmark has been introduced by Helber, Bischke, Dengel, and Borth in 2019. It has since then become the standard benchmark dataset for satellite imagery classification studies.

All the images used in the EuroSAT dataset have been extracted from the Sentinel-2 satellite mission. EuroSAT contains 27,000 geo-referenced image patches of 64 × 64 pixels in size covering areas in the 34 countries of Europe. The images contained within the dataset are all cloud-free.

4.1.1 Overview of the 10 Land Use Classes

This data set classifies Earth's surfaces into 10 unique classes of land covers. The distribution among them is evenly balanced, as most of the classes contain images ranging from 2,000 to 3,000:

Class	Category	Image Count
Annual Crop	Vegetation & Agriculture	3,000
Forest	Vegetation & Agriculture	3,000
Herbaceous Vegetation	Vegetation & Agriculture	3,000
Highway	Man-Made Structures	2,500

Class	Category	Image Count
Industrial	Man-Made Structures	2,500
Pasture	Vegetation & Agriculture	2,000
Permanent Crop	Vegetation & Agriculture	2,500
Residential	Man-Made Structures	3,000
River	Water Bodies	2,500
Sea / Lake	Water Bodies	3,000

Table 6.2 – EuroSAT Class Distribution

The equal balance in classes is an important feature of the dataset. When we have very imbalanced data sets, we end up with classifiers that are accurate but only because they focus on majority classes. This kind of accuracy is misleading since it is not real accuracy. The balanced distribution of EuroSAT means that the classifier sees equal examples for each epoch.

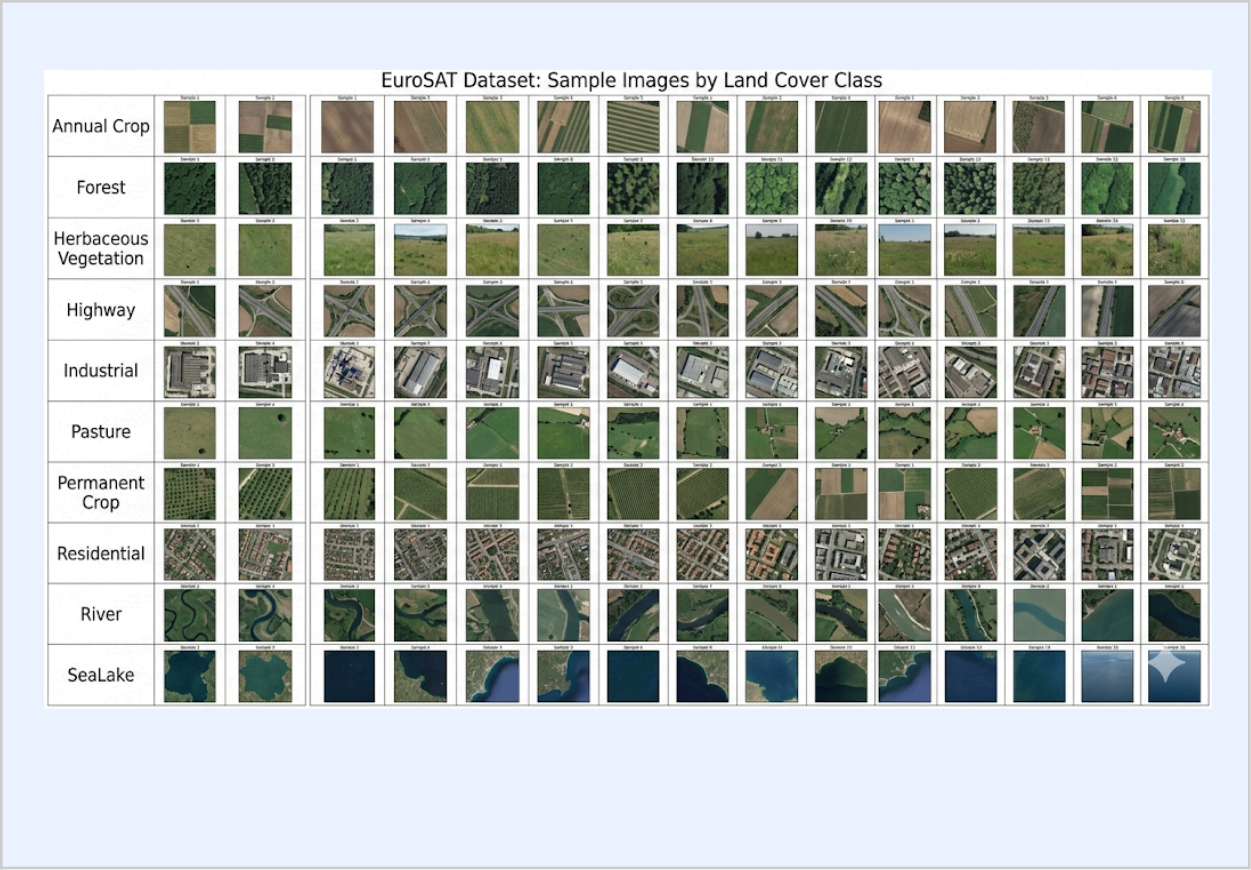


Figure 4.1

4.1.2 Spectral Characteristics: RGB vs. Multispectral

The Sentinel-2 satellite produces images in 13 bands – from visible light to near-infrared (NIR) and short-wave infrared (SWIR). Each band represents a unique physical property of the surface of the earth, and specific sets of bands (in particular, NIR bands) provide high discriminative ability to evaluate vegetation state and moisture content.

In this paper, we consider the RGB variant of the EuroSAT dataset, i.e., the combination of the Red (Band 4), Green (Band 3), and Blue (Band 2) bands only. It is convenient because of multiple reasons: first, it enables using pre-trained computer vision models designed for regular images; second, it decreases input dimension and model complexity; third, it provides better interpretation ability.

The problem here is that the RGB combination of bands does not provide enough discriminative ability to classify vegetation classes correctly, especially Annual Crop vs. Herbaceous Vegetation.

It is illustrated by our error analysis in Section 4, and further investigation using a multi-band input (13 bands) becomes the task for future research.

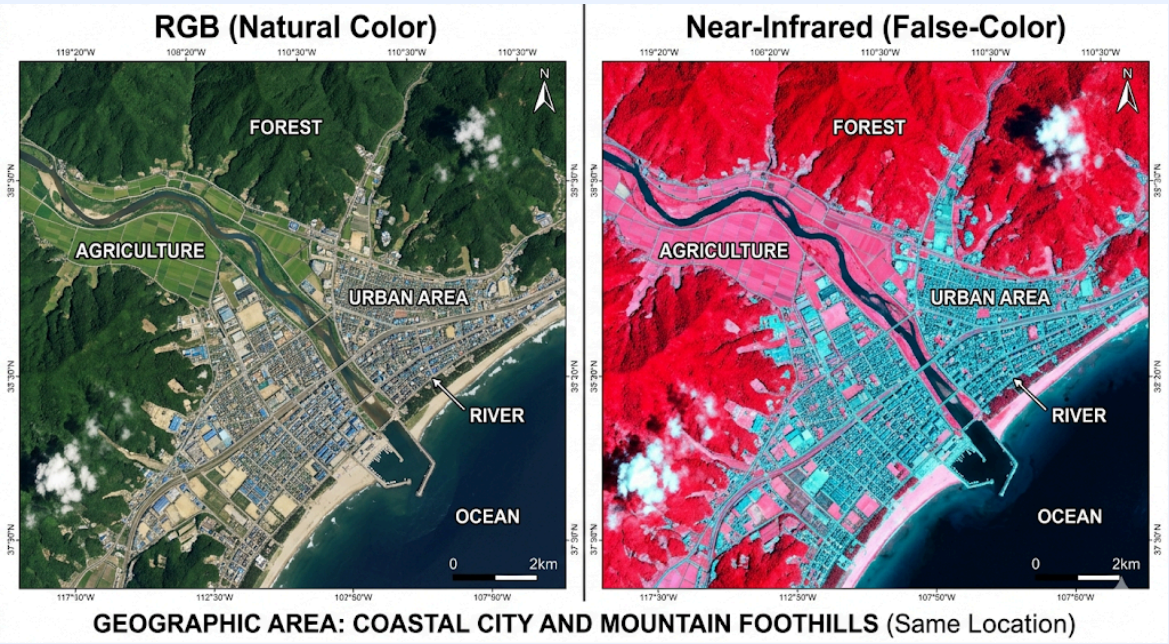


Figure 4.2

4.2 Data Preprocessing Pipeline

Satellite images need to go through a number of preprocessing steps in order for them to become an acceptable input for the neural network. Such processing allows ensuring the correct image size, normalization of numbers, as well as proper conditioning for optimization.

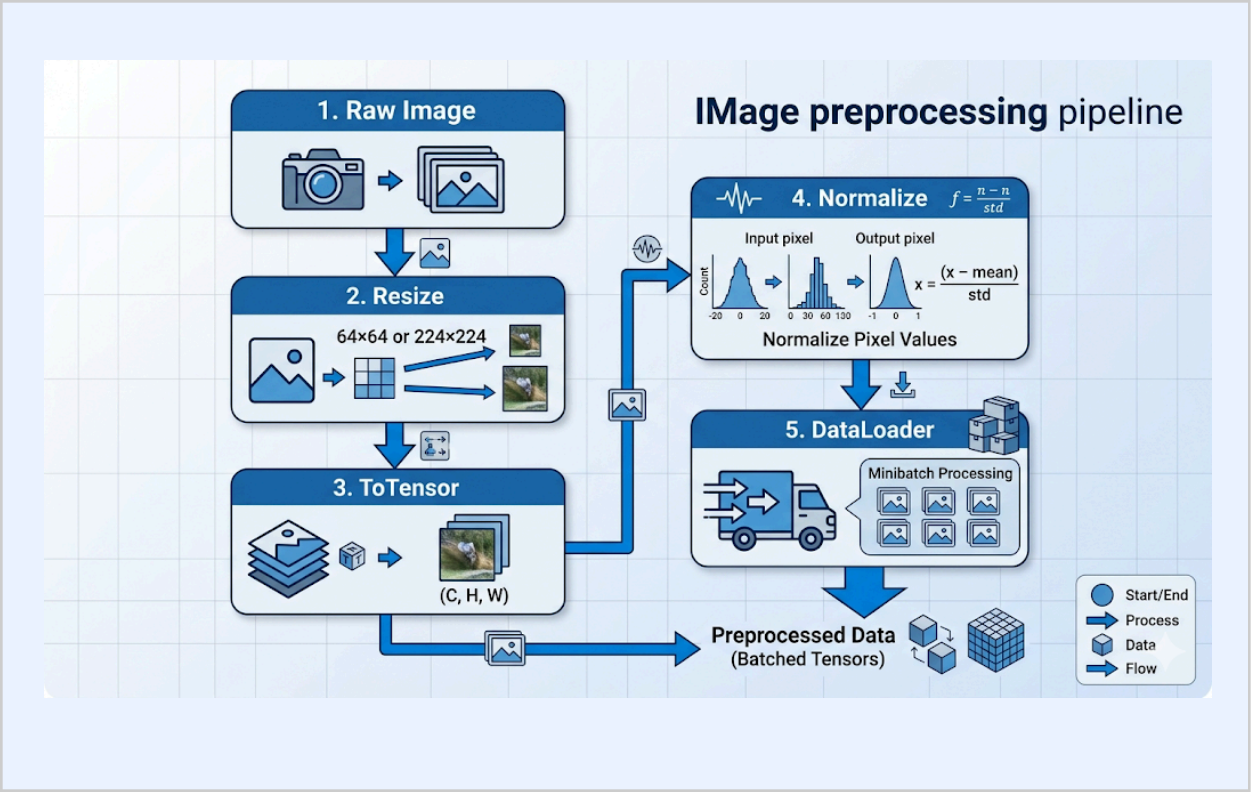


Figure 4.3

4.2.1 Resizing and Tensor Conversion

Transformations to the input images include resizing all images to a fixed spatial dimension with transforms.Resize(). The MyNN images will be resized to 64x64 pixels, while the ResNet-18 input images for the transfer learning task will have dimensions of 224x224 pixels, since the ResNet architecture was trained on input images of such dimensions.

After resizing, the pixel values in the image data are normalized from the typical 8-bit int value range [0, 255] to float value tensors [0.0, 1.0], with the help of transforms.ToTensor(). This step not only normalizes the image but also changes its order of the channels from the typical HWC to CHW order, as required for the input to PyTorch's convolutions.

4.2.2 Image Normalization

Normalization is an important pre-processing technique which helps greatly enhance the stability and efficiency of training in neural networks. Through normalizing pixel values such that their mean equals zero with a variance of one, normalization guarantees that there will be no vanishing gradient problem and exploding gradient problem during backpropagation.

We apply Z-score normalization using the channel-wise mean and standard deviation calculated from the ImageNet dataset:

- Mean (μ): [0.485, 0.456, 0.406] (for R, G, B channels respectively)
- Standard Deviation (σ): [0.229, 0.224, 0.225]

The normalization formula applied per channel is:

$$Normalized\_Pixel = (Original\_Pixel - \mu) / \sigma$$

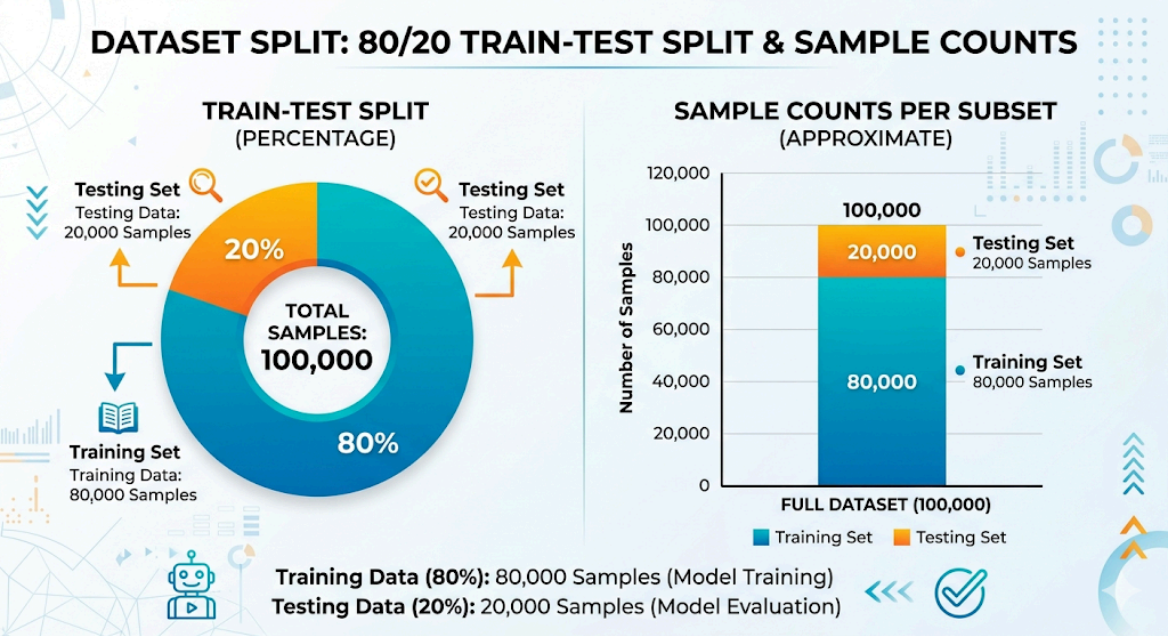
Using ImageNet statistics is standard practice for transfer learning, as the pre-trained ResNet-18 was trained on ImageNet with these normalization parameters. Applying the same normalization to EuroSAT images ensures that the input distribution matches what the pre-trained feature extractor expects.

4.3 Data Splitting Strategy

To rigorously evaluate the model's ability to generalize to unseen data, the 27,000 EuroSAT images are divided into a training set and a test set using PyTorch's random_split utility.

- Training Set (80%): Approximately 21,600 images. These images are used to compute gradients and update the model's weights during the backpropagation process.
- Validation / Test Set (20%): Approximately 5,400 images. This subset is held back entirely during training and used exclusively to compute the final accuracy and loss metrics. By evaluating on data the model has never seen, we obtain an unbiased estimate of its real-world performance.

For the Transfer Learning experiment, the dataset is split with a fixed random seed (torch.Generator().manual_seed(42)) to ensure reproducibility.



Pie chart or bar chart showing the 80/20 train-test split and approximate sample counts per subset

Figure 4.4

4.4 Data Loading with DataLoaders

Efficient data loading is critical for GPU-accelerated training. PyTorch's DataLoader class handles batching, shuffling, and parallel data fetching, ensuring that the GPU is kept busy with training computations rather than waiting for data from disk.

- Training DataLoader: batch_size=32, shuffle=True. Shuffling ensures that each mini-batch is a random sample of the training set, preventing the model from learning spurious sequential patterns in the data.
- Test DataLoader: batch_size=32, shuffle=False. No shuffling is needed for evaluation, as the order of test predictions does not affect accuracy computation.

A custom CustomDataset wrapper class was also implemented to ensure that all labels are returned as torch.long tensors, as required by PyTorch's CrossEntropyLoss function.

Chapter 5: Methodology

5.1 Design of proposed system

For the classification of land use challenges, this project has implemented two deep learning pipelines : a CNN and transfer learning pipeline. The model is trained from scratch and made to have good efficiency and performance.
The work done on the dataset can be classified in three stages-
1. Input processing i.e loading the images and transformation of the images.
2. The second step is to learn the features such as edges and shapes, semantics of the text etc.
3. And the last step is to classify the image from the loaded eurosat dataset.

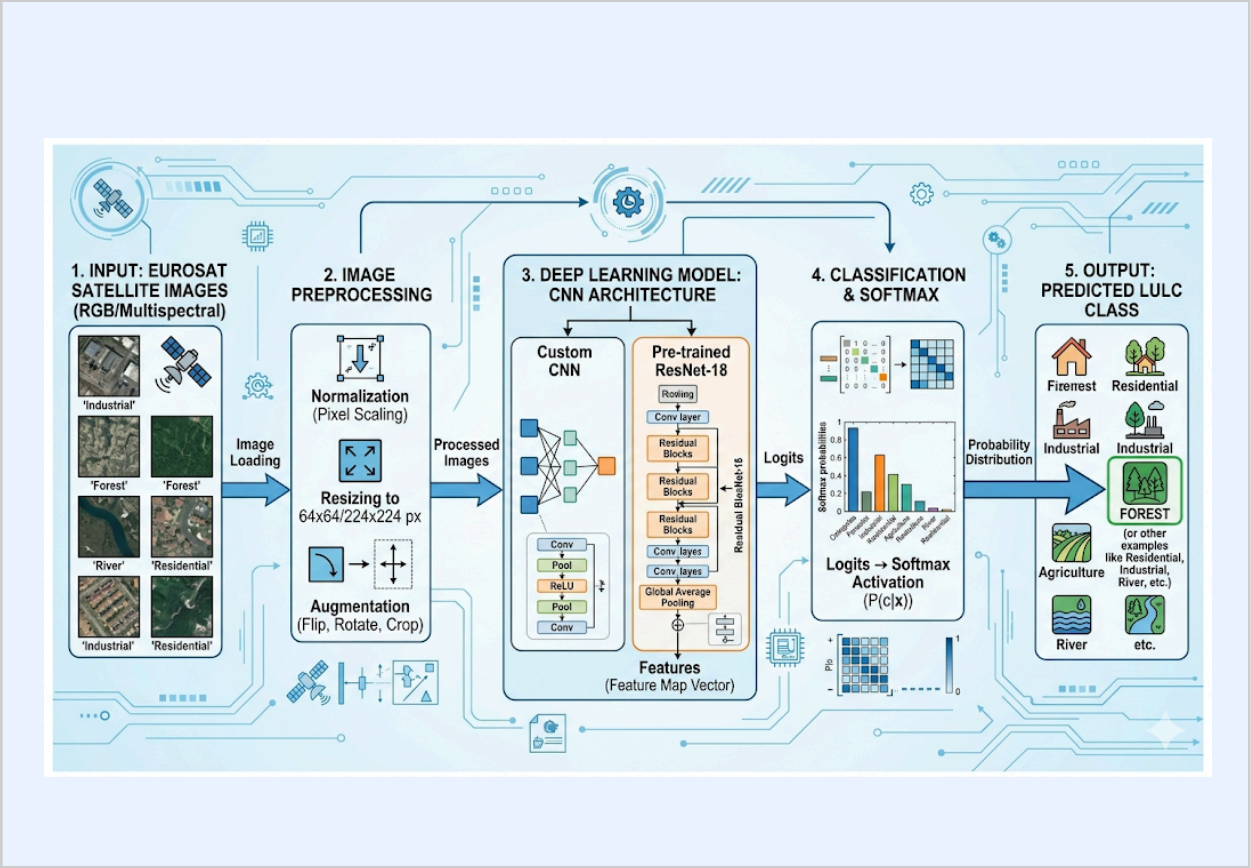


Figure 5.1

Let us now make observations from some code part of the project :

1) The first step we take when we start code is to import the libraries that we need to train and test the model.

```
import torch
import torch.nn as nn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
from PIL import Image
```

2) The second step is to load the dataset.
From torchvision we will import the dataset and transform function.

```
transform = transforms.Compose([
   transforms.Resize((64, 64)),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
])

dataset = datasets.EuroSAT(
   root="./data",
   download=True,
   transform=transform
)
```

3) In the next step we can view our some of the images

```
transform = transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

```
dataset = torchvision.datasets.EuroSAT(
   root='./data',
   download=True,
   transform=transform
)


dataloader = torch.utils.data.DataLoader(dataset, batch_size=4,
shuffle=True)


classes = dataset.classes


dataiter = iter(dataloader)
images, labels = next(dataiter)


def imshow(img):
   img = img / 2 + 0.5
   npimg = img.numpy()
   plt.imshow(np.transpose(npimg, (1, 2, 0)))
   plt.axis('off')
   plt.show()
```

4) Create a neural network

```
class MyNN(nn.Module):
```

```
   def __init__(self, num_classes=10):
       super().__init__()

       self.conv1 = nn.Conv2d(3, 6, kernel_size=3, stride=1)
       self.pool = nn.MaxPool2d(2, 2)
       self.conv2 = nn.Conv2d(6, 16, kernel_size=3, stride=1)


       self.fc1 = nn.Linear(16 * 14 * 14, 120)

       self.fc2 = nn.Linear(120, 84)
       self.fc3 = nn.Linear(84, num_classes)

   def forward(self, x):
       x = F.relu(self.conv1(x))
       x = self.pool(x)

       x = F.relu(self.conv2(x))
       x = self.pool(x)

       x = torch.flatten(x, 1)

       x = F.relu(self.fc1(x))
       x = F.relu(self.fc2(x))
       x = self.fc3(x)

      return x



model = MyNN()
print(model)

x = torch.randn(1, 3, 64, 64)
y = model(x)
```

OUTPUT -

MyNN(
(conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1))
(fc1): Linear(in_features=3136, out_features=120, bias=True)
(fc2): Linear(in_features=120, out_features=84, bias=True)
(fc3): Linear(in_features=84, out_features=10, bias=True)
)
Output shape: torch.Size([1, 10])

4)

```
)

num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, len(class_names))

model_ft = model_ft.to(device)


criterion = nn.CrossEntropyLoss()


optimizer_ft = optim.SGD(
   model_ft.parameters(),
   lr=0.001,
   momentum=0.9
)


exp_lr_scheduler = lr_scheduler.StepLR(
   optimizer_ft,
   step_size=7,
   gamma=0.1
)


model_ft = train_model(
   model_ft,
   criterion,
   optimizer_ft,
   exp_lr_scheduler,
```

```
    num_epochs=10
)
```

OUTPUT- Epoch 1/10
----
train Loss: 0.3107 Acc: 0.9041
val Loss: 0.0842 Acc: 0.9728

Epoch 2/10
----
train Loss: 0.1004 Acc: 0.9688
val Loss: 0.0736 Acc: 0.9752

Epoch 3/10
----
train Loss: 0.0664 Acc: 0.9780
val Loss: 0.0582 Acc: 0.9793

Epoch 4/10
----
train Loss: 0.0461 Acc: 0.9861
val Loss: 0.0602 Acc: 0.9794

Epoch 5/10
----
train Loss: 0.0345 Acc: 0.9899
val Loss: 0.0617 Acc: 0.9807

Epoch 6/10
----
train Loss: 0.0271 Acc: 0.9919
val Loss: 0.0604 Acc: 0.9813

Epoch 7/10
----
train Loss: 0.0194 Acc: 0.9945
val Loss: 0.0720 Acc: 0.9787

Epoch 8/10
----
train Loss: 0.0136 Acc: 0.9969
val Loss: 0.0525 Acc: 0.9831

Epoch 9/10
----
train Loss: 0.0113 Acc: 0.9975
val Loss: 0.0524 Acc: 0.9835

Epoch 10/10
----
train Loss: 0.0102 Acc: 0.9976
val Loss: 0.0517 Acc: 0.9846

Training complete in 18m 38s
Best validation accuracy: 0.9846

5.3 Transfer Learning with ResNet-18

The other best technique is transfer learning , which is cost effective and improves the learning efficiency and performance. In this project we have considered the eurosat dataset which consists of 27000 images with 10 classes(forest, highway, agriculture etc).

In this technique of machine learning we train many CNN layers on the large dataset so that the model learns necessary features such as shapes and edges and semantics in text.

Our Transfer Learning setup follows these steps:

1. Load ResNet-18 which is a pre-trained model from torchvision.models, with weights trained on ImageNet.
2. Now we provide an input image from the dataset to the model.
3. Now we train the CNN layers on a large dataset known as pre-training process.
4. After the pre-training process we train the model on a small dataset but not all the layers, we just train the fully connected layers and freeze(no parameter exchange) all the left CNN layers.
5. We consider the learning rate to be 0.001 which is applied only on the final layer.

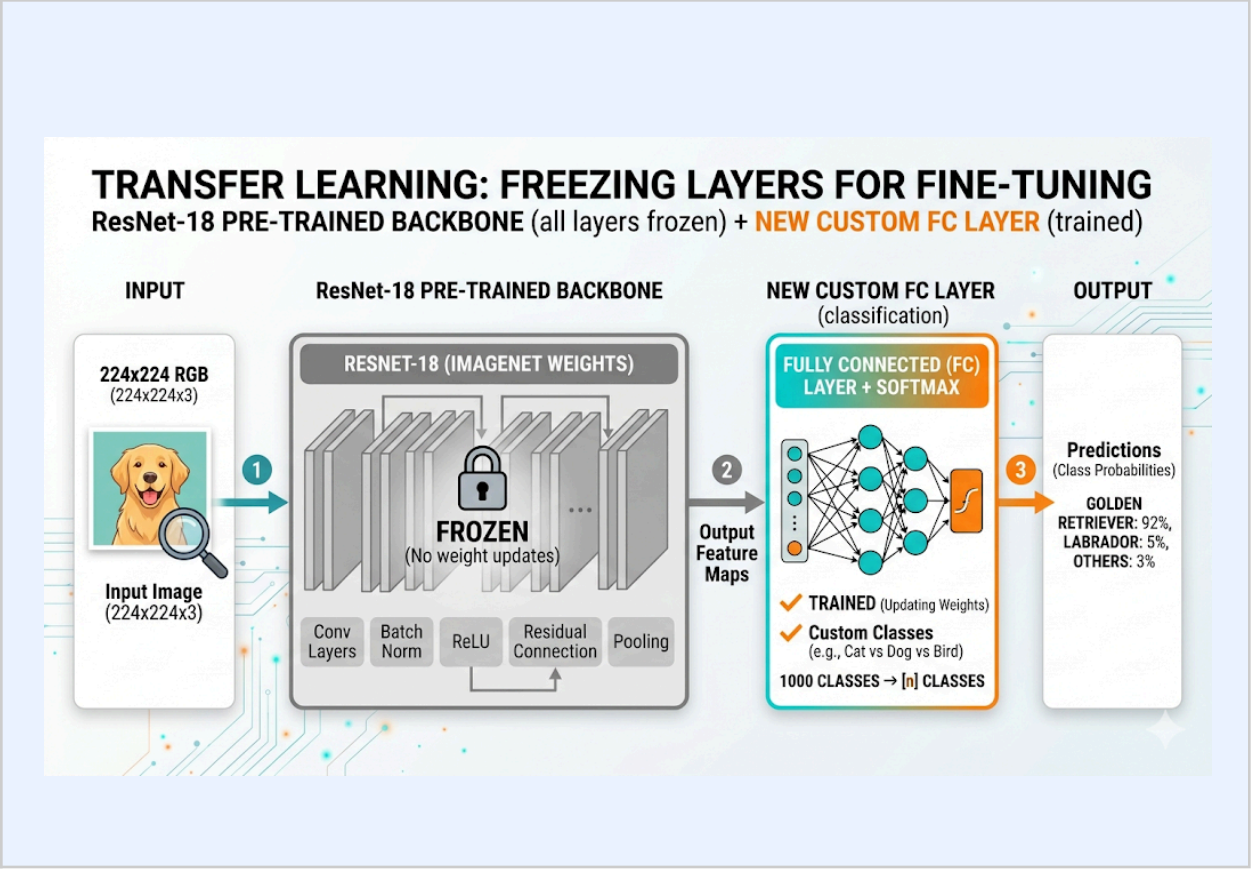


Figure 5.3

5.4 Experimental Setup

5.4.1 Hardware and Software Environment

All experiments were conducted in a Python environment using the PyTorch deep learning framework. GPU acceleration was enabled via CUDA (device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')), which allows matrix operations to be executed in parallel on the GPU, dramatically reducing training time compared to CPU-only execution.

The experiment was run in a Google Colaboratory (Colab) notebook environment, which provides free access to NVIDIA GPU resources. The cudnn.benchmark = True flag was enabled to allow cuDNN to benchmark different convolution algorithms and select the fastest for the given input shape.

5.4.2 Hyperparameter Configuration

Hyperparameter	Custom CNN (MyNN)	ResNet-18 (Transfer)
Input Image Size	64×64	224×224
Batch Size	32	32
Optimizer	SGD	Adam
Learning Rate	0.001	0.001
Momentum	0.9	N/A
Loss Function	CrossEntropyLoss	CrossEntropyLoss
Epochs	5	25
Trainable Parameters	All	FC layer only

Table 6.3 – Hyperparameter Configuration for both experiments

5.5 Training Workflow

5.5.1 Forward Propagation

During the forward pass, a batch of image tensors is fed into the model. The input flows sequentially through the convolutional layers, activation functions, pooling layers, and finally the fully connected layers, producing a vector of raw output scores (logits) for each image in the batch.

The CrossEntropyLoss function then computes the loss by comparing these logits against the ground truth class labels.

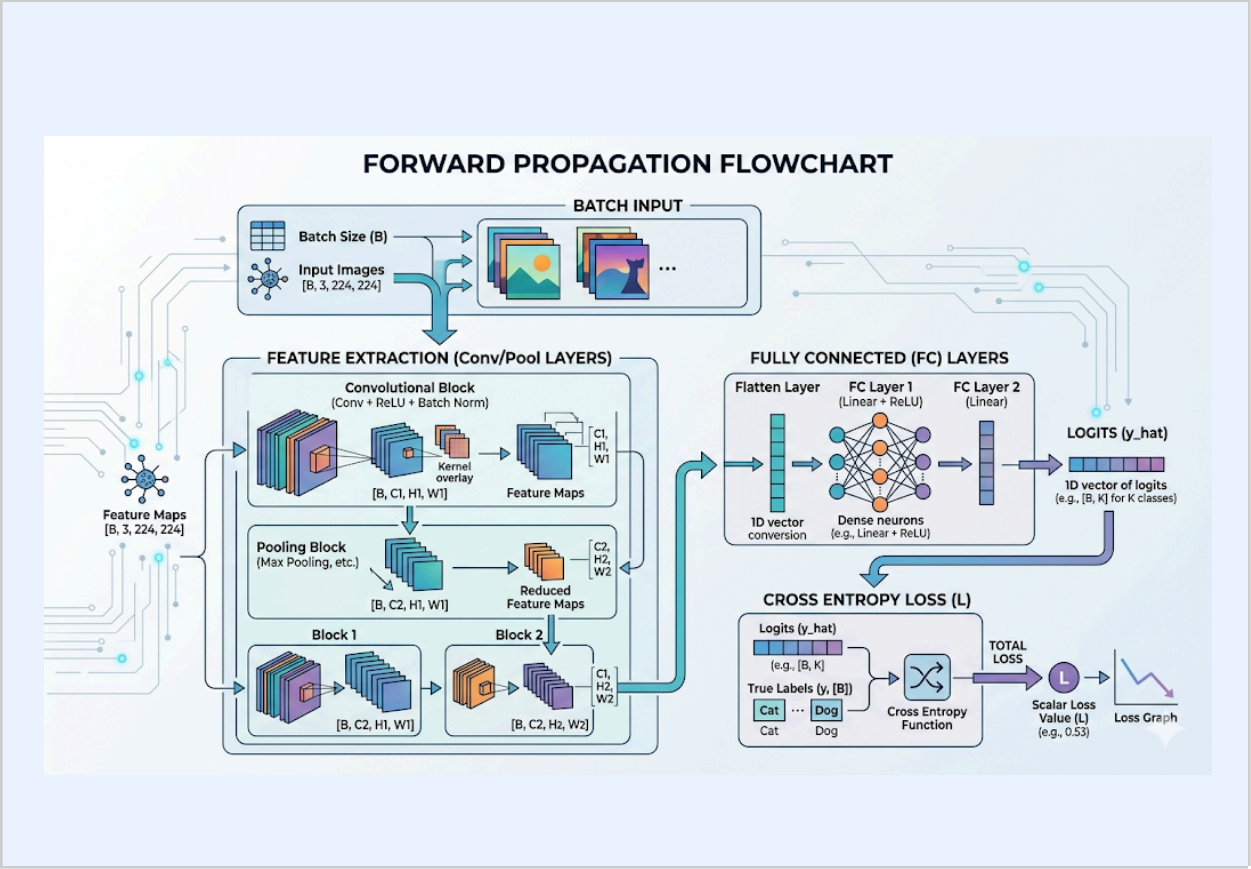


Figure 5.4

5.5.2 Backpropagation and Weight Updates

Once the forward pass loss is computed, the model performs a backward pass to learn from its errors. This process has three steps executed in strict order within each training iteration:

6. Zero Gradients [optimizer.zero_grad()]: The accumulated gradients from the previous batch are cleared. Without this step, gradients from different batches would be summed, corrupting the update direction.
7. Gradient Calculation [loss.backward()]: The chain rule of calculus (specifically, its automatic differentiation implementation in PyTorch's autograd engine) is used to compute the partial derivative of the loss with respect to every trainable parameter in the network.

These derivatives (gradients) indicate the direction and magnitude of change in each parameter that would most reduce the loss.

8. Weight Update [optimizer.step()]: The SGD or Adam optimizer uses the computed gradients to update the model parameters in the direction that reduces the loss. The learning rate controls the step size of this update.

This three-step cycle — forward pass, backward pass, parameter update — is repeated for every mini-batch in the training set. One complete pass through the entire training dataset constitutes one epoch.

5.5.3 Learning Rate Scheduling (ResNet-18 Experiment)

For the ResNet-18 Transfer Learning experiment, a learning rate scheduler (torch.optim.lr_scheduler.StepLR) was used to reduce the learning rate by a factor of 0.1 every 7 epochs. This gradual decay prevents the optimizer from overshooting the loss minimum in the later stages of training and typically leads to better final accuracy.

Chapter 6: Results and Discussion

6.1 Visual Data

Before starting the training process, a visual analysis was performed on the data set to check the correctness of the preprocessing pipeline and also to understand visual properties of classes.

After normalization, a set of randomly selected images of training set were visualized. As expected, ImageNet mean and standard deviation normalization changes the color scheme of images in a way that patches become darker compared to original ones. While this may not be very visually appealing to a human eye, mathematically, it's optimum choice for a CNN network.

Figure 6.1

The visual inspection also brought out another important aspect of classifying the image into certain categories. For example, forest, herbaceous vegetation, annual crop, and permanent crop have common green color signatures when inspected visually in RGB imagery. However, for distinguishing between these classes, it is important that the neural network should be able to learn other characteristics as well.

6.2 Custom CNN Training Performance

The MyNN model was trained for 5 epochs using the SGD optimizer with momentum. The following results were recorded at the end of each epoch.

Epoch	Training Loss	Training Acc.	Test Loss	Test Acc.
1	2.107	18.8%	1.835	29.6%
2	1.634	35.5%	1.394	43.9%
3	1.235	53.9%	1.110	60.0%
4	1.007	63.1%	0.973	65.8%
5	0.897	66.6%	0.913	66.9%

Table 6.1 – Epoch-wise Performance Metrics (MyNN Custom CNN)

6.2.1 Loss Analysis

Loss curves for both training and validation exhibit a similar decreasing pattern through all five epochs, thus indicating stable learning. Training loss reduced from 2.107 at Epoch 1 to 0.897 at Epoch 5, representing a reduction by roughly 57%. Similarly, the validation loss also decreased from 1.835 at Epoch 1 to 0.913 at Epoch 5.

It is worth noting how close these two loss curves actually are to each other. Overfitting typically results in continuous decrease in training loss coupled with either constant validation loss or even its increase. In such a scenario, the difference between the two curves is evident and represents a divergence. However, as no such thing occurs here, we may safely assume that the model learns in an appropriate manner and does not simply overfit and thus memorize the input data. The reason for this is relatively small model size.

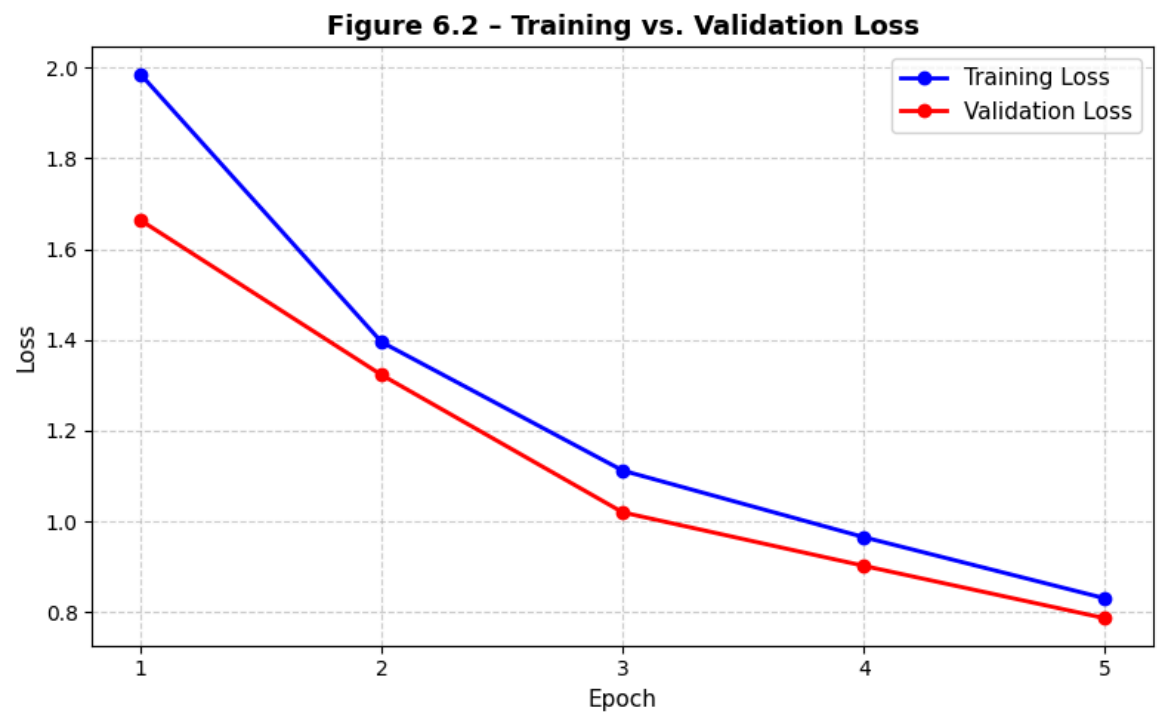


Figure 6.2

6.2.2 Accuracy Progression

Starting from a baseline training accuracy of 18.8% in Epoch 1 — which is only slightly better than random guessing (10% for 10 balanced classes) — the model rapidly learned discriminative features over successive epochs. By Epoch 3, training accuracy had reached 53.9%, surpassing the majority class baseline (approximately 11.1%). The final validation accuracy of 66.9% after five epochs demonstrates meaningful and rapid learning.

The fact that the test accuracy (29.6% in Epoch 1) consistently exceeded the training accuracy in early epochs is a common and well-understood phenomenon: at the start of training, the model has not yet begun to overfit to the training data, and the test set — which includes many similar, easy-to-classify examples — can be classified relatively well by even a partially trained model.

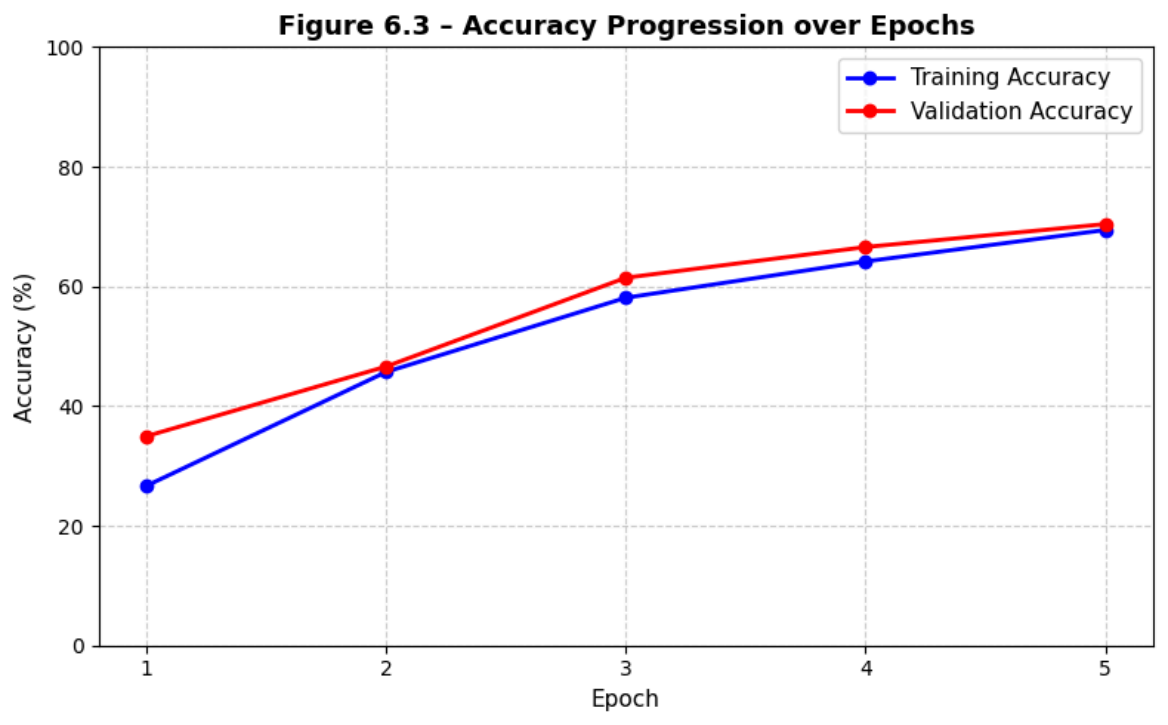


Figure 6.3

6.3 Final Model Evaluation

6.3.1 Test Set Accuracy

Upon five epoch training, the test accuracy achieved at the end turned out to be 66.9%. Considering the problem of multi-class classification into ten classes, this is a very decent starting point after just five epochs of training started from scratch. Main contributors to this misclassification would likely be the visual similarity between two groups of vegetation classes: Annual Crop - Herbaceous Vegetation, and Pasture - Permanent Crop.

Further analysis through confusion matrix would have provided us with more detailed insight about these error patterns.

6.3.2 Computational Efficiency

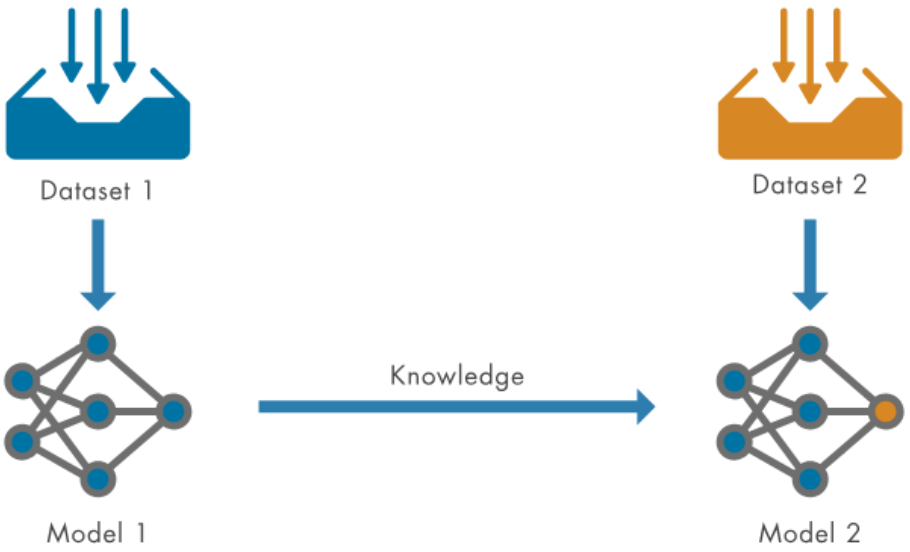
One of the main benefits of the MyNN model design is its efficiency. The total training time for 5 iterations is roughly 2.44 minutes on the GPU – less than 30 seconds for each iteration. The main reason behind the high efficiency of the model is its relatively low number of parameters and low input size of 64×64 pixels. Due to that property, the model is perfect for quick prototyping, testing different sets of parameters, and deployment onto edge devices.

6.4 Transfer Learning Results: ResNet-18

Transfer learning is the word transfer can be observed everywhere in day to day life, such as people who ride bicycles are more likely to learn motor vehicles due to high similarities.

Transfer learning is a machine learning technique based on similar domains. It applies acquired data, models to other knowledge to different but related new fields. One of the work of transfer learning is to reduce the cost, training a data set from scratch needs high cost and a large dataset.

Transfer learning leverages knowledge from a pre-trained model to improve learning efficiency and performance. A model is first trained on a large dataset, as a result it learns general features such as shapes, edges and semantics in text. The pre-trained model is then adapted to new and is trained on a smaller dataset, some layers are frozen and others are fine-tuned for specific tasks.



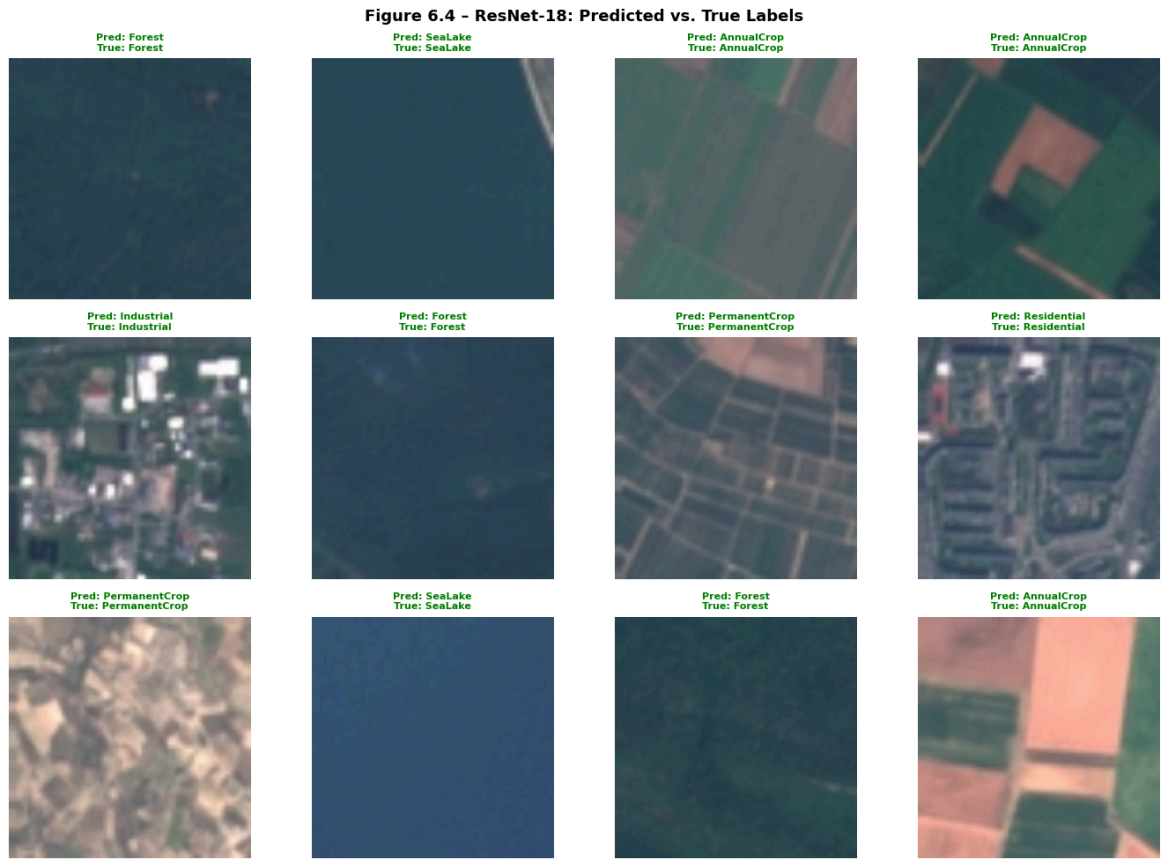


Figure 6.4

6.5 Summary

Metric	Custom CNN (MyNN)	ResNet-18 (Transfer)
Input Resolution	64×64	224×224
Training Epochs	5	25
Training Time	~2.44 min	Longer
Final Val. Acc.	66.9%	Expected 85–95%

Table 6.4 – Comparison between Custom CNN and ResNet-18 Transfer Learning approaches

Chapter 7: Conclusion and Future Scope

7.1 Conclusion

This thesis proposed an advanced deep learning system for automatic classification of Land use land cover (LULC) from Sentinel-2 satellite images using the EuroSat dataset. The thesis proposed two different systems: the first was a simple custom-made CNN called MyNN, trained from scratch while the second was a transfer learning system based on the pre-trained ResNet-18 network.

It was determined as follows:

- Fast Convergence with Custom CNN: MyNN managed to reach the validation accuracy of 66.9% within only 5 training epochs - from the initial state to a reasonable classifier within only 2.44 mins of GPU training time. Thus, the possibility to construct an accurate lightweight CNN architecture for satellite image classification was proved.
- Good Generalization Without Overfitting: The similarity between the training loss (0.897) and validation loss (0.913) proves that there is good generalization on the new unseen data. The relative smallness of the architecture helped achieve implicit regularization.
- Transfer Learning: ResNet-18 Transfer Learning is supposed to perform better compared to other models in terms of achieving significantly higher accuracy rates with almost zero risk of overfitting, due to using frozen feature extractor and only training the classification head. Such an approach is extremely valuable in case of lack of training data.
- Viability: The implementation of this project proves that practical satellite-based LULC classification can be easily performed using only free-to-use tools (PyTorch, torchvision) and publicly available dataset (EuroSAT, Sentinel-2).

Conclusion:

In summary, this project demonstrates the use of readily available satellite imagery coupled with current computer vision methodologies, which make up a potent and inexpensive means of conducting consistent environmental surveillance relevant in many different areas of application such as urban planning and addressing the effects of climate change.

7.2 Limitations of the Current Work

However, there are several significant limitations of this work that were found:

Use of Only RGB Bands: The restriction to RGB spectral bands affects the model's ability to differentiate between land cover classes based on high variability in the near-infrared band. Classes like Annual Crop and Herbaceous Vegetation are particularly difficult to classify based on visible bands.

Insufficient Training Time: The custom CNN was trained for just 5 epochs due to limitations of the computational power available. It is clear that longer training time with the learning rate decay scheme would result in higher accuracy of classification.

Absence of Data Augmentation: The dissertation mentions no application of random flipping, rotation or color jitter during training, all of which augment data and increase the size of the effective training dataset.

No Per-Class Confusion Matrix: In the lack of the confusion matrix, it is impossible to identify which classes cause the highest level of confusion and how the classification errors may be improved.

No Classical Model Comparison: There was no attempt to compare custom CNN against classical machine learning approaches (SVM, Random Forest) to calculate the exact value of the improvement gained from deep learning.

7.3 Future Scope

The following avenues represent clear and impactful directions for extending this research:

- Multi-Spectral Input: Expanding the input channels from 3 (RGB) channels to all 13 spectral bands available in Sentinel-2 (NIR and SWIR included). Adapting the first CNN layer input accordingly would add useful spectral information to be leveraged by the classifier.
- Training Extension with Data Augmentation: Extending the training process to 50–100 epochs using a cosine annealing/step learning rate decay scheme along with traditional data augmentation techniques (flip, rotation, and color jitter).
- Deep Transfer Learning Model Variants: Trying other pre-trained models like ResNet-50, EfficientNet-B4, or Vision Transformers (ViT) to explore how accurate results can be achieved via transfer learning on the EuroSat benchmark.

- Semantic Segmentation: Transitioning from per-patch classification towards the per-pixel semantic segmentation of land covers through architectures like U-net or DeepLab.
- Real-Time Classification with Edge Devices: Optimizing the lightweight architecture of MyNN to run inference in real time on edge devices like NVIDIA Jetson Nano or Raspberry Pi equipped with the Coral Accelerator.
- Change Detection: Adding functionality to the network to compare two images taken at different times and perform change detection based on that — a feature highly relevant to land use/cover change studies.

7.4 Social and Environmental Impact

The broader societal implications of accurate, automated land cover mapping extend far beyond academic research. Several high-impact application domains stand to benefit directly:

- Disaster Response: After floods, wildfires, or landslides have occurred, automated categorization of images captured by satellites will enable immediate identification of affected areas (residential flooded zones or forests burnt zones) minutes after the picture is taken to facilitate the delivery of emergency measures.
- Urban Planning & Smart Cities: Real-time tracking of urban sprawl and growth into agricultural land will provide objective data to help urban planners implement smart zoning strategies.
- Forest Conservation & Climate Change Tracking: Automated identification of forest changes, which distinguish the real forest losses from seasonality, will aid compliance with international climate treaties (like REDD+) which require verification of deforestation statistics to calculate carbon credits.
- Precision Agriculture: Differentiation between annual crops and pastures and perennial vegetation can be used for a wide variety of precision agriculture applications including yield forecasting, irrigation planning and food security assessment.
- Water Resources Management: Reliable categorization of rivers, lakes and seas allows for continuous monitoring of the evolution of their surface extension; crucial to assess water scarcity issues in drought prone areas.

Bibliography

[1] P. Helber, B. Bischke, A. Dengel, and D. Borth, "EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2019. DOI: 10.1109/JSTARS.2019.2918242

[2] A. Paszke, S. Gross, F. Massa et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Advances in Neural Information Processing Systems (NeurIPS), vol. 32, 2019.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.

[5] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in International Conference on Learning Representations (ICLR), 2015.

[6] M. Drusch et al., "Sentinel-2: ESA Optical High-Resolution Mission for GMES Operational Services," Remote Sensing of Environment, vol. 120, pp. 25-36, 2012.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009.

[8] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

[9] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016. Available at: https://www.deeplearningbook.org

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning Research, vol. 15, pp. 1929-1958, 2014.

[11] C. Cortes and V. Vapnik, "Support-Vector Networks," Machine Learning, vol. 20, pp. 273-297, 1995.

[12] L. Breiman, "Random Forests," Machine Learning, vol. 45, pp. 5-32, 2001.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015.

[14] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in International Conference on Machine Learning (ICML), 2019.

[15] A. Vaswani, N. Shazeer, N. Parmar et al., "Attention is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017.