Intel Image Classification Using CNN

A Final Project

Presented to Mr. Louie Cervantes

College of Information and Communications Technology

West Visayas State University

La Paz, Iloilo City

In Fulfillment of the Requirements

for the Alternative Learning Assessment in the Course

Intelligent Systems (CCS 229)

By

Christian Angelo Panique

BSCS 3A - AI

May 2025

Table of Contents

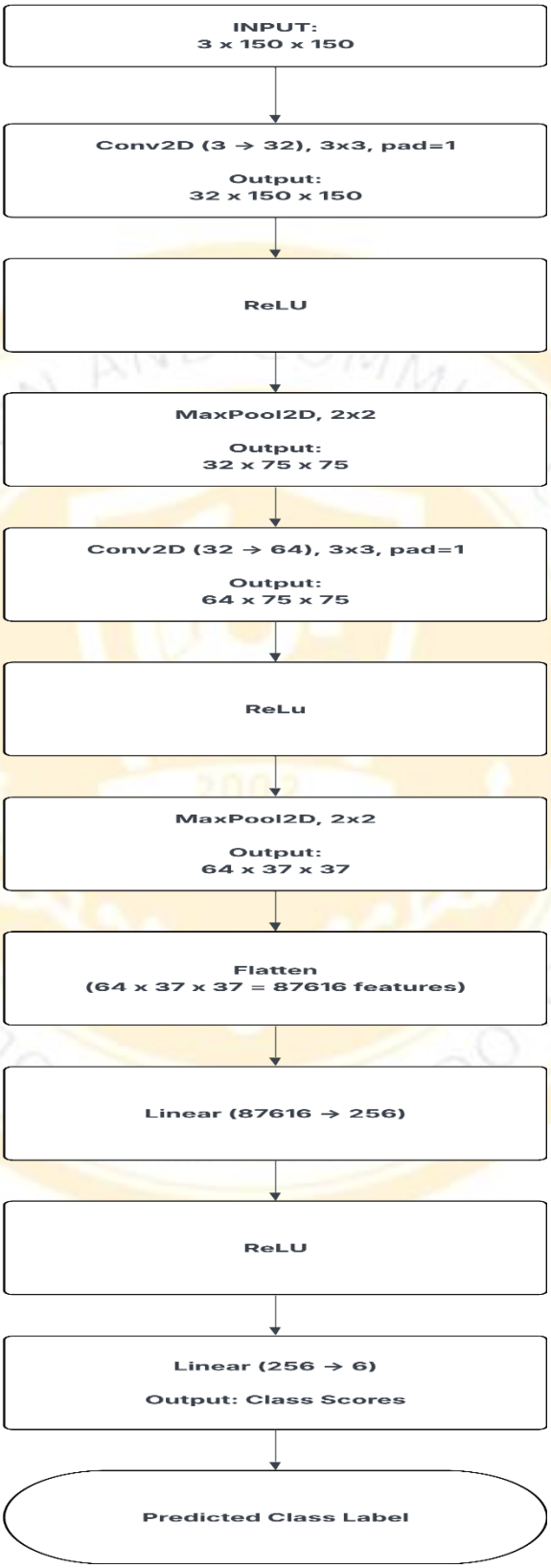
I. Project Overview & Objectives

This project implements a convolutional neural network (CNN) to classify natural scene images from the Intel Image Classification dataset. The model is trained to predict six classes: buildings, forest, glacier, mountain, sea, and street.

Objectives:

- Develop a CNN-based image classifier.
- Train and evaluate the model using PyTorch.
- Deploy the model via a Streamlit web application.
- Analyze model performance and generate classification metrics.

II. Model Architecture Diagram



III. Dataset

Dataset: Intel Image Classification
Source: Kaggle (https://www.kaggle.com/datasets/puneet6060/intel-image-classification)

Dataset Objective:
The dataset is designed to support image classification tasks where the goal is to classify scenes into one of six natural or man-made environments. It is suitable for training convolutional neural networks (CNNs) and testing their ability to generalize across diverse image textures, lighting conditions, and landscapes.

Number of Classes:

The dataset contains **6 scene categories**:

Label	Description
Buildings	Images of man-made structures like houses and skyscrapers
Forest	Dense tree-covered landscapes, forests, and jungles
Glacier	Ice-covered terrain, often with snow and mountains
Mountain	Rocky and hilly terrain, usually with minimal vegetation
Sea	Images of water bodies such as oceans, beaches, and seas
Street	Urban roads, streets, and cityscapes

Dataset Structure:
The dataset is organized in a folder-based hierarchy compatible with PyTorch's ImageFolder class. Each subfolder represents a class.

seg_train/

├── buildings/

├── forest/

├── glacier/

├── mountain/

├── sea/

└── street/

Data Split Used in Training:
We used only the seg_train directory for both training and validation. The images were split using an **80/20 ratio**:

- **Training Set**: 80% of the images randomly selected from each class.
- **Validation Set**: 20% held out for evaluating model generalization.

This ensures that class distribution is preserved across both sets.

Image Preprocessing & Augmentation:
The following preprocessing steps were applied using torchvision.transforms:

For Training:

- Resize to (150, 150)
- Random horizontal flip
- Normalize with mean and std for ImageNet

For Validation & Inference:

- Resize to (150, 150)
- Normalize (same as training)

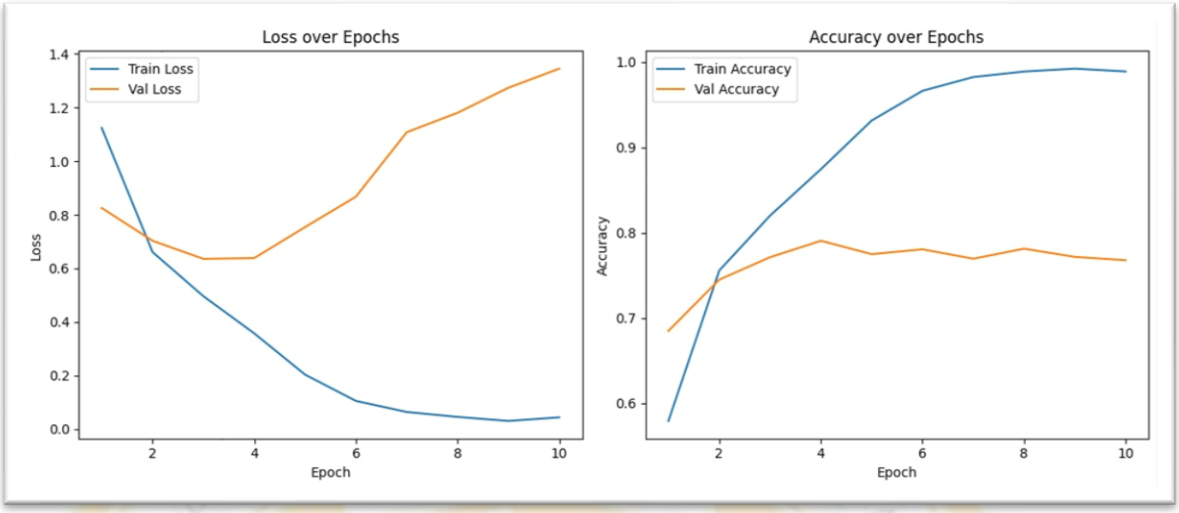
These augmentations help the model generalize better and prevent overfitting.

Class Distribution (approximate):

Class	# Images
Buildings	~2190
Forest	~2270
Glacier	~2400
Mountain	~2500
Sea	~2270
Street	~2380

Total: ~14,000 images in the training set

IV. Training Logs & Charts

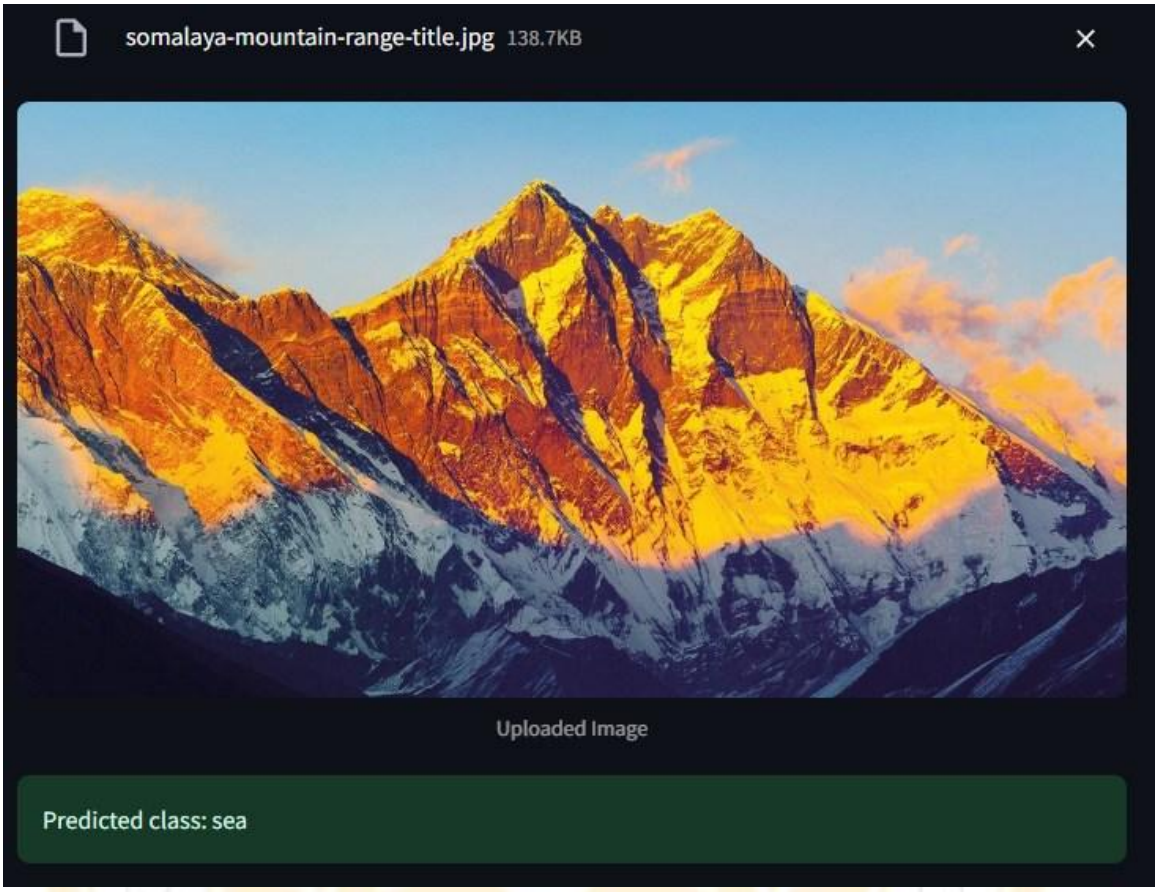



```
Classification Report:
              precision    recall  f1-score   support

   buildings       0.75      0.71      0.73       421
      forest       0.93      0.89      0.91       455
     glacier       0.73      0.76      0.74       492
    mountain       0.73      0.70      0.72       503
         sea       0.68      0.75      0.71       456
      street       0.80      0.79      0.80       480

    accuracy                           0.77      2807
   macro avg       0.77      0.77      0.77      2807
weighted avg       0.77      0.77      0.77      2807
```

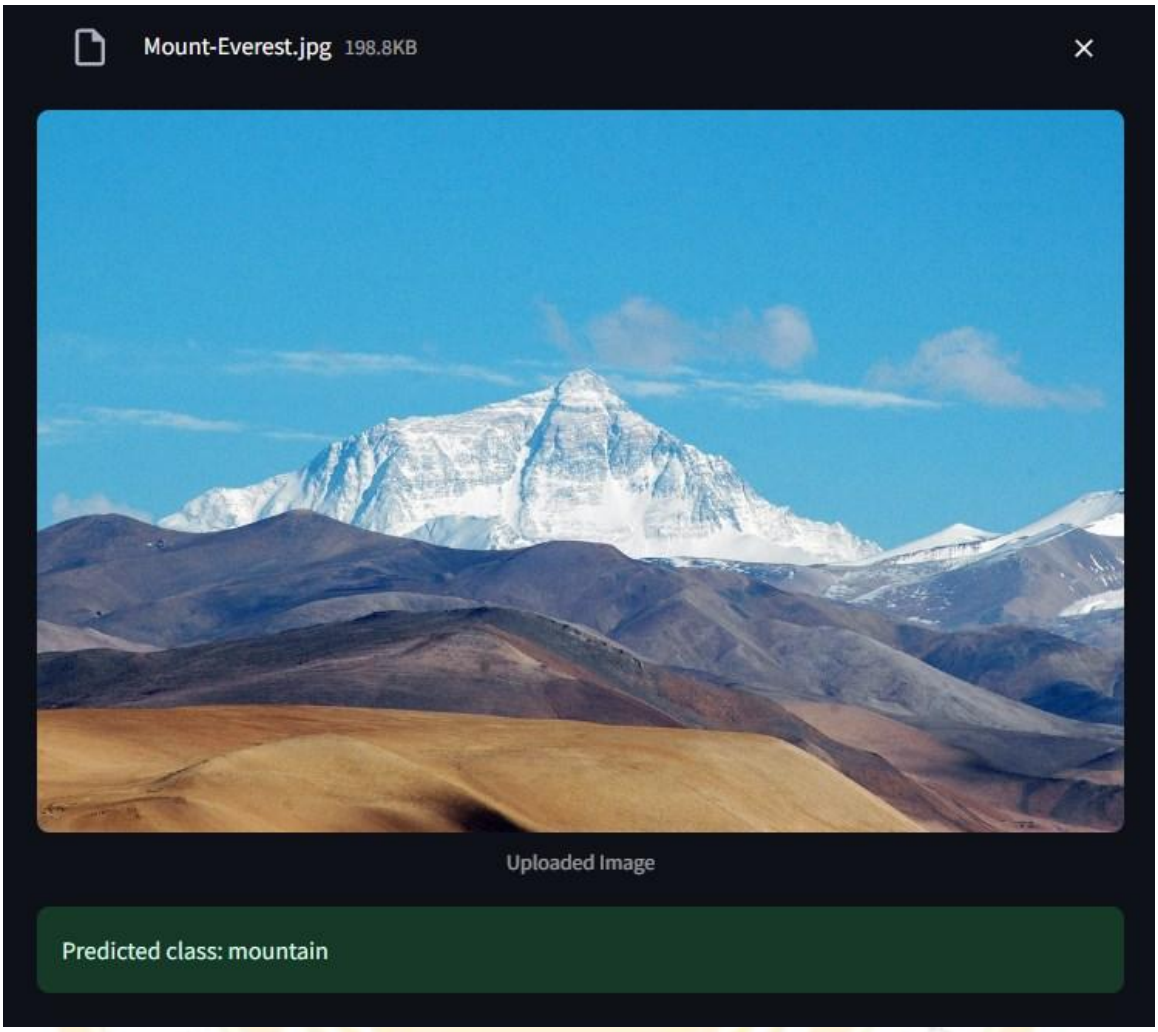
```
Classes: ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
Epoch 1/10 – Train loss: 1.1243, Train acc: 0.5789, Val loss: 0.8257, Val acc: 0.6847
Epoch 2/10 – Train loss: 0.6613, Train acc: 0.7557, Val loss: 0.7036, Val acc: 0.7449
Epoch 3/10 – Train loss: 0.4962, Train acc: 0.8200, Val loss: 0.6359, Val acc: 0.7713
Epoch 4/10 – Train loss: 0.3572, Train acc: 0.8743, Val loss: 0.6389, Val acc: 0.7905
Epoch 5/10 – Train loss: 0.2028, Train acc: 0.9313, Val loss: 0.7544, Val acc: 0.7748
Epoch 6/10 – Train loss: 0.1045, Train acc: 0.9661, Val loss: 0.8679, Val acc: 0.7805
Epoch 7/10 – Train loss: 0.0629, Train acc: 0.9822, Val loss: 1.1082, Val acc: 0.7695
Epoch 8/10 – Train loss: 0.0446, Train acc: 0.9887, Val loss: 1.1808, Val acc: 0.7813
Epoch 9/10 – Train loss: 0.0295, Train acc: 0.9921, Val loss: 1.2739, Val acc: 0.7716
Epoch 10/10 – Train loss: 0.0430, Train acc: 0.9888, Val loss: 1.3453, Val acc: 0.7677
```

The training logs indicate that the model learned effectively during the early epochs, with validation accuracy peaking at 79.05% by Epoch 4. However, as training progressed, the training accuracy continued to rise reaching over 99% while the validation accuracy plateaued and even slightly declined, signaling clear signs of overfitting. The final validation accuracy settled at 77%, showing a moderate level of generalization. Among the classes, "forest" performed the best with an F1-score of 0.91, while "sea" was the most challenging with an F1-score of 0.71, likely due to visual similarities with other natural scenes. Overall, the model demonstrates strong potential, but performance could be further improved through techniques such as regularization, early stopping, enhanced data augmentation, and adaptive learning rate scheduling.

V. Sample Predictions



As we can see here, the model failed to identify the Somalayan Mountain Range. It classified the mountain as the sea; the model struggles with images that has similar features with other classes.



Here, it correctly predicted Mount Everest as a mountain.

VI. Limitations & Improvements

Limitations:

- Slight overfitting visible after epoch 5.
- Model accuracy plateaus due to increasing validation loss.
- Some class confusion (e.g., sea vs. glacier vs. mountain).

Future Improvements:

- Implement learning rate scheduling and early stopping.
- Use data augmentation strategies to improve generalization.
- Experiment with pre-trained models (ResNet, MobileNet).
- Increase training epochs with model checkpointing.

VII. References

- Intel Image Classification Dataset: https://www.kaggle.com/datasets/puneet6060/intel-image-classification

- PyTorch Documentation: https://pytorch.org/docs/stable/index.html

- Streamlit Documentation: https://docs.streamlit.io/

- scikit-learn Classification Report: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html