

Section 1: Import Libraries

```
# =====  
# SECTION 1: IMPORT LIBRARIES  
# Intel Image Classification Project  
# =====  
  
import os  
from pathlib import Path  
from collections import Counter  
  
import matplotlib.pyplot as plt  
from PIL import Image  
  
from google.colab import drive  
  
print("="*70)  
print("LIBRARIES IMPORTED SUCCESSFULLY")  
print("="*70)  
  
=====
```

LIBRARIES IMPORTED SUCCESSFULLY

```
=====
```

Section 2: Mount Google Drive

```
# =====  
# SECTION 2: MOUNT GOOGLE DRIVE  
# =====  
  
drive.mount('/content/drive')  
  
print("\nGoogle Drive mounted successfully.")  
  
Mounted at /content/drive  
  
Google Drive mounted successfully.
```

Section 3: Dataset Inspection

```

# =====
# SECTION 3: DEFINE DATASET PATH
# نظرا لعدد الصور الكبير تم اختيار مجلد الفحص في الداتا سيت ليكون هو الداتا سيت الخاص
# بنا في هذا المشروع
# =====

DATASET_PATH = "/content/drive/MyDrive/seg_test"

print("Dataset Path:")
print(DATASET_PATH)

Dataset Path:
/content/drive/MyDrive/seg_test

# =====
# VERIFY DATASET
# =====

if os.path.exists(DATASET_PATH):
    print("Dataset found successfully.")
else:
    raise FileNotFoundError("Dataset folder not found!")

Dataset found successfully.

# =====
# DATASET OVERVIEW
# =====

image_extensions = (".jpg", ".jpeg", ".png", ".bmp")

classes = sorted(
    [
        folder for folder in os.listdir(DATASET_PATH)
        if os.path.isdir(os.path.join(DATASET_PATH, folder))
    ]
)

print("="*70)
print("DATASET SUMMARY")
print("="*70)

total_images = 0

for cls in classes:

    class_path = os.path.join(DATASET_PATH, cls)

    num_images = len(
        [
            img for img in os.listdir(class_path)
            if img.lower().endswith(image_extensions)
        ]
    )

```

```

    )

    total_images += num_images

    print(f"{cls:<12} : {num_images:,} images")

print("-"*70)
print(f"Number of Classes : {len(classes)}")
print(f"Total Images      : {total_images:,}")

=====
DATASET SUMMARY
=====
buildings      : 437 images
forest         : 474 images
glacier        : 553 images
mountain       : 525 images
sea            : 510 images
street         : 501 images
-----
Number of Classes : 6
Total Images      : 3,000

# =====
# VISUALIZE DATASET DISTRIBUTION
# =====

counts = []

for cls in classes:

    class_path = os.path.join(DATASET_PATH, cls)

    count = len(
        [
            img for img in os.listdir(class_path)
            if img.lower().endswith(image_extensions)
        ]
    )

    counts.append(count)

plt.figure(figsize=(8,5))

plt.bar(classes, counts)

plt.title("Images per Class")

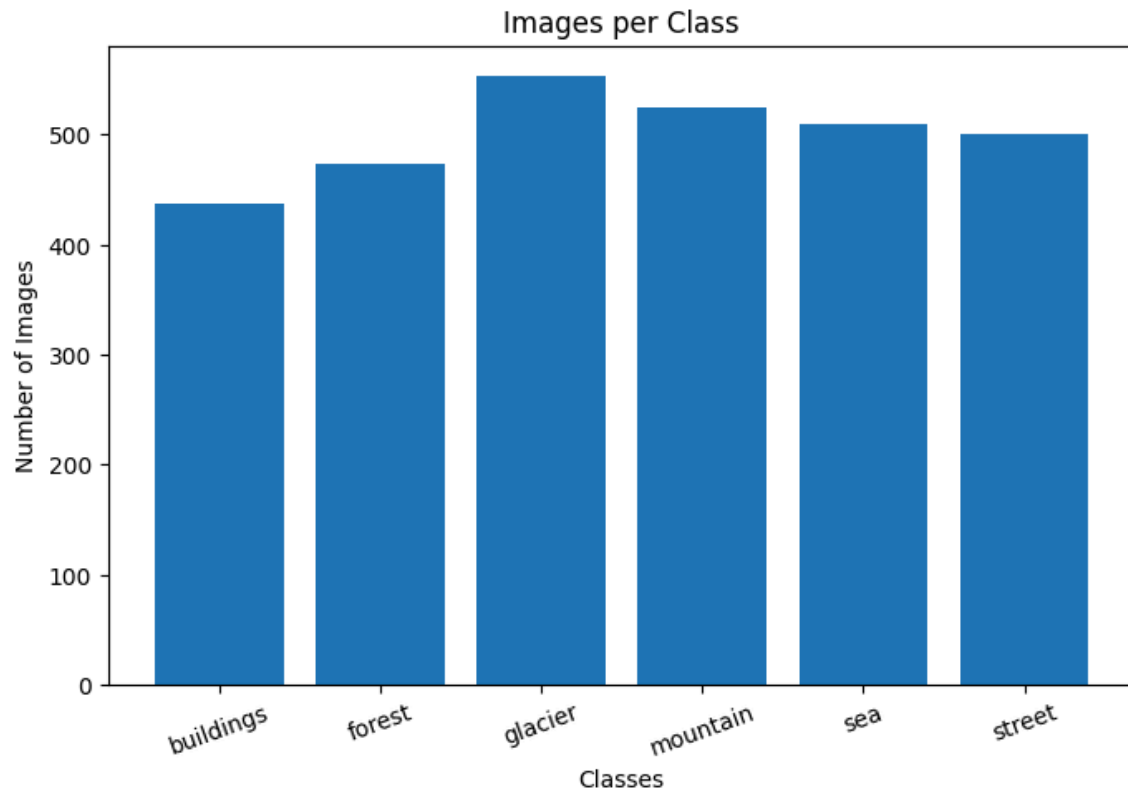
plt.xlabel("Classes")

plt.ylabel("Number of Images")

plt.xticks(rotation=20)

```

```
plt.show()
```



Section 4: Image Resizing

```
# =====  
# SECTION 2: IMAGE RESIZING  
# =====  
  
from PIL import Image  
from tqdm import tqdm  
import os  
  
# -----  
# Input & Output Paths  
# -----  
INPUT_PATH = "/content/drive/MyDrive/seg_test"  
OUTPUT_PATH = "/content/drive/MyDrive/resized_dataset"  
  
# Image size  
IMAGE_SIZE = (224, 224)  
  
# Supported image formats  
IMAGE_EXTENSIONS = (".jpg", ".jpeg", ".png", ".bmp")  
  
# Create output folder  
os.makedirs(OUTPUT_PATH, exist_ok=True)
```

```

print("=" * 70)
print("RESIZING IMAGES")
print("=" * 70)

total_processed = 0

# Loop through each class
for class_name in sorted(os.listdir(INPUT_PATH)):

    input_class_path = os.path.join(INPUT_PATH, class_name)

    if not os.path.isdir(input_class_path):
        continue

    output_class_path = os.path.join(OUTPUT_PATH, class_name)
    os.makedirs(output_class_path, exist_ok=True)

    images = [
        img for img in os.listdir(input_class_path)
        if img.lower().endswith(IMAGE_EXTENSIONS)
    ]

    print(f"\nProcessing class: {class_name}")

    for image_name in tqdm(images):

        input_image = os.path.join(input_class_path, image_name)
        output_image = os.path.join(output_class_path, image_name)

        try:
            with Image.open(input_image) as img:

                img = img.convert("RGB")
                img = img.resize(IMAGE_SIZE)

                img.save(output_image)

                total_processed += 1

        except Exception as e:
            print(f"Error: {image_name} --> {e}")

print("\n" + "=" * 70)
print("IMAGE RESIZING COMPLETED")
print("=" * 70)
print(f"Output Folder : {OUTPUT_PATH}")
print(f"Image Size      : {IMAGE_SIZE}")
print(f"Images Saved     : {total_processed}")

```

```

=====
RESIZING IMAGES

```

```
=====
```

Processing class: buildings

100%|██████████| 437/437 [02:08<00:00, 3.40it/s]

Processing class: forest

100%|██████████| 474/474 [02:19<00:00, 3.39it/s]

Processing class: glacier

100%|██████████| 553/553 [02:39<00:00, 3.47it/s]

Processing class: mountain

100%|██████████| 525/525 [02:29<00:00, 3.52it/s]

Processing class: sea

100%|██████████| 510/510 [02:28<00:00, 3.43it/s]

Processing class: street

100%|██████████| 501/501 [02:23<00:00, 3.48it/s]

```
=====
```

IMAGE RESIZING COMPLETED

```
=====
```

Output Folder : /content/drive/MyDrive/resized_dataset
Image Size : (224, 224)
Images Saved : 3000

```
# =====  
# COMPARE DATASET SIZE BEFORE & AFTER RESIZING  
# =====
```

```
import os
```

```
ORIGINAL_PATH = "/content/drive/MyDrive/seg_test"  
RESIZED_PATH = "/content/drive/MyDrive/resized_dataset"
```

```
def get_folder_size(folder):  
    total_size = 0
```

```
    for dirpath, _, filenames in os.walk(folder):
```

```

        for filename in filenames:
            filepath = os.path.join(dirpath, filename)

            if os.path.isfile(filepath):
                total_size += os.path.getsize(filepath)

    return total_size

def convert_size(size_bytes):

    units = ["Bytes", "KB", "MB", "GB"]

    size = float(size_bytes)

    for unit in units:

        if size < 1024:
            return f"{size:.2f} {unit}"

        size /= 1024

    return f"{size:.2f} TB"

original_size = get_folder_size(ORIGINAL_PATH)
resized_size = get_folder_size(RESIZED_PATH)

saved_size = original_size - resized_size
reduction = (saved_size / original_size) * 100

print("=" * 70)
print("DATASET SIZE COMPARISON")
print("=" * 70)

print(f"Original Dataset Size : {convert_size(original_size)}")
print(f"Resized Dataset Size : {convert_size(resized_size)}")
print(f"Saved Space : {convert_size(saved_size)}")
print(f"Reduction Percentage : {reduction:.2f}%")

print("=" * 70)

=====
DATASET SIZE COMPARISON
=====
Original Dataset Size : 43.28 MB
Resized Dataset Size : 32.29 MB
Saved Space : 10.99 MB
Reduction Percentage : 25.40%
=====

# =====
# VERIFY RESIZED DATASET

```

```
# =====

print("=" * 70)
print("RESIZED DATASET SUMMARY")
print("=" * 70)

total = 0

for cls in sorted(os.listdir(OUTPUT_PATH)):

    class_path = os.path.join(OUTPUT_PATH, cls)

    if not os.path.isdir(class_path):
        continue

    count = len(os.listdir(class_path))

    total += count

    print(f"{cls:<12}: {count}")

print("-" * 70)
print(f"Total Images : {total}")

=====
RESIZED DATASET SUMMARY
=====
buildings      : 437
forest         : 474
glacier        : 553
mountain       : 525
sea            : 510
street         : 501
-----
Total Images : 3000
```

Section 5: Dataset Splitting

```
# =====
# SECTION 5: DATASET SPLITTING
# =====

import os
import shutil
from sklearn.model_selection import train_test_split

# -----
# Paths
# -----
```

```

SOURCE_DATASET = "/content/drive/MyDrive/resized_dataset"
OUTPUT_DATASET = "/content/drive/MyDrive/split_dataset"

TRAIN_DIR = os.path.join(OUTPUT_DATASET, "train")
VAL_DIR   = os.path.join(OUTPUT_DATASET, "validation")
TEST_DIR  = os.path.join(OUTPUT_DATASET, "test")

# -----
# Split Ratios
# -----

TRAIN_RATIO = 0.70
VAL_RATIO   = 0.15
TEST_RATIO  = 0.15

# -----
# Create Output Folders
# -----

for folder in [TRAIN_DIR, VAL_DIR, TEST_DIR]:
    os.makedirs(folder, exist_ok=True)

print("=" * 70)
print("DATASET SPLITTING")
print("=" * 70)

summary = []

# -----
# Split Each Class
# -----

for class_name in sorted(os.listdir(SOURCE_DATASET)):

    class_path = os.path.join(SOURCE_DATASET, class_name)

    if not os.path.isdir(class_path):
        continue

    images = [
        f for f in os.listdir(class_path)
        if f.lower().endswith((".jpg", ".jpeg", ".png", ".bmp"))
    ]

    # Train / Temp
    train_imgs, temp_imgs = train_test_split(
        images,
        train_size=TRAIN_RATIO,
        random_state=42,
        shuffle=True
    )

```

```

# Validation / Test
val_imgs, test_imgs = train_test_split(
    temp_imgs,
    test_size=TEST_RATIO / (VAL_RATIO + TEST_RATIO),
    random_state=42,
    shuffle=True
)

splits = {
    "train": train_imgs,
    "validation": val_imgs,
    "test": test_imgs
}

for split_name, split_images in splits.items():

    destination = os.path.join(
        OUTPUT_DATASET,
        split_name,
        class_name
    )

    os.makedirs(destination, exist_ok=True)

    for image in split_images:

        shutil.copy2(
            os.path.join(class_path, image),
            os.path.join(destination, image)
        )

    summary.append([
        class_name,
        len(train_imgs),
        len(val_imgs),
        len(test_imgs)
    ])

print("Dataset splitting completed successfully.\n")

# -----
# Display Summary
# -----

print("=" * 70)
print(f"{'Class':<15}{'Train':>10}{'Validation':>15}{'Test':>10}")
print("=" * 70)

train_total = 0
val_total = 0
test_total = 0

```

```

for row in summary:

    train_total += row[1]
    val_total += row[2]
    test_total += row[3]

    print(f"{row[0]:<15}{row[1]:>10}{row[2]:>15}{row[3]:>10}")

print("=" * 70)
print(f"{'TOTAL':<15}{train_total:>10}{val_total:>15}{test_total:>10}")
print("=" * 70)

print(f"Total Images : {train_total + val_total + test_total}")

=====
DATASET SPLITTING
=====
Dataset splitting completed successfully.

=====
Class          Train      Validation      Test
=====
buildings      305          66             66
forest         331          71             72
glacier        387          83             83
mountain       367          79             79
sea            357          76             77
street         350          75             76
=====
TOTAL          2097         450            453
=====
Total Images : 3000

```

Section 6: Data Augmentation

```

# =====
# SECTION 6: DATA AUGMENTATION & DATA LOADERS
# =====

import torch
from torchvision import datasets, transforms as T
from torch.utils.data import DataLoader

# -----
# Dataset Path
# -----

DATASET_PATH = "/content/drive/MyDrive/split_dataset"

# -----

```

```
# Hyperparameters
```

```
# -----
```

```
IMAGE_SIZE = 224
```

```
BATCH_SIZE = 32
```

```
# -----
```

```
# Training Transform (Data Augmentation)
```

```
# -----
```

```
train_transform = T.Compose([
```

```
    T.RandomHorizontalFlip(p=0.5),
```

```
    T.RandomRotation(10),
```

```
    T.RandomResizedCrop(
```

```
        IMAGE_SIZE,
```

```
        scale=(0.9, 1.0)
```

```
    ),
```

```
    T.ToTensor(),
```

```
    T.Normalize(
```

```
        mean=[0.485, 0.456, 0.406],
```

```
        std=[0.229, 0.224, 0.225]
```

```
    )
```

```
])
```

```
# -----
```

```
# Validation & Test Transform
```

```
# -----
```

```
test_transform = T.Compose([
```

```
    T.Resize((IMAGE_SIZE, IMAGE_SIZE)),
```

```
    T.ToTensor(),
```

```
    T.Normalize(
```

```
        mean=[0.485, 0.456, 0.406],
```

```
        std=[0.229, 0.224, 0.225]
```

```
    )
```

```
])
```

```
# -----
```

```
# Load Datasets
```

```
# -----
```

```
train_dataset = datasets.ImageFolder(
```

```

        root=f"{DATASET_PATH}/train",
        transform=train_transform
    )

validation_dataset = datasets.ImageFolder(
    root=f"{DATASET_PATH}/validation",
    transform=test_transform
)

test_dataset = datasets.ImageFolder(
    root=f"{DATASET_PATH}/test",
    transform=test_transform
)

# -----
# Data Loaders
# -----

train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2
)

validation_loader = DataLoader(
    validation_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2
)

test_loader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2
)

# -----
# Classes
# -----

classes = train_dataset.classes
num_classes = len(classes)

# -----
# Device
# -----

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

```

```

# -----
# Summary
# -----

print("="*70)
print("DATA LOADERS CREATED SUCCESSFULLY")
print("="*70)

print(f"Classes           : {classes}")
print(f"Number of Classes : {num_classes}")
print()

print(f"Training Images   : {len(train_dataset)}")
print(f"Validation Images  : {len(validation_dataset)}")
print(f"Testing Images    : {len(test_dataset)}")
print()

print(f"Batch Size        : {BATCH_SIZE}")
print(f"Device            : {device}")

print("="*70)

=====
DATA LOADERS CREATED SUCCESSFULLY
=====
Classes           : ['buildings', 'forest', 'glacier', 'mountain', 'sea',
'street']
Number of Classes : 6

Training Images   : 2097
Validation Images  : 450
Testing Images    : 453

Batch Size        : 32
Device            : cuda
=====

```

Section 7: EfficientNet-B0 Transfer Learning

```

# =====
# SECTION 7: BUILD EFFICIENTNET-B0 MODEL
# =====

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models

```

```

# -----
# Load Pretrained EfficientNet-B0
# -----

model = models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.DEFAULT)

# -----
# Freeze Feature Extractor
# -----

for param in model.features.parameters():
    param.requires_grad = False

# -----
# Replace Classification Layer
# -----

num_features = model.classifier[1].in_features

model.classifier[1] = nn.Linear(
    num_features,
    num_classes
)

# -----
# Move Model to Device
# -----

model = model.to(device)

# -----
# Loss Function
# -----

criterion = nn.CrossEntropyLoss()

# -----
# Optimizer
# -----

optimizer = optim.Adam(
    model.classifier.parameters(),
    lr=0.001
)

# -----
# Learning Rate Scheduler
# -----

scheduler = optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,

```

```

        gamma=0.1
    )

# -----
# Display Model Information
# -----

print("="*70)
print("MODEL CONFIGURATION")
print("="*70)

print("Model                : EfficientNet-B0")
print("Transfer Learning     : Enabled")
print("Frozen Layers          : Feature Extractor")
print("Trainable Layer         : Final Classifier")
print(f"Number of Classes       : {num_classes}")
print(f"Loss Function           : {criterion}")
print(f"Optimizer                : Adam")
print(f"Learning Rate            : 0.001")
print("="*70)

Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-
7f5810bc.pth" to
/root/.cache/torch/hub/checkpoints/efficientnet_b0_rwightman-7f5810bc.pth

100%|██████████| 20.5M/20.5M [00:00<00:00, 149MB/s]

```

```

=====
MODEL CONFIGURATION
=====
Model                : EfficientNet-B0
Transfer Learning     : Enabled
Frozen Layers          : Feature Extractor
Trainable Layer         : Final Classifier
Number of Classes     : 6
Loss Function         : CrossEntropyLoss()
Optimizer             : Adam
Learning Rate         : 0.001
=====

```

Section 8: MODEL TRAINING

```

# =====
# SECTION 8: TRAIN EFFICIENTNET-B0
# =====

import copy
import time

EPOCHS = 10

```

```

print("=" * 70)
print("TRAINING STARTED")
print("=" * 70)

best_accuracy = 0.0
best_model_weights = copy.deepcopy(model.state_dict())

history = {
    "train_loss": [],
    "train_acc": [],
    "val_loss": [],
    "val_acc": []
}

start_time = time.time()

for epoch in range(EPOCHS):

    print(f"\nEpoch [{epoch+1}/{EPOCHS}]")
    print("-" * 70)

    # =====
    # Training
    # =====
    model.train()

    running_loss = 0.0
    running_correct = 0
    total = 0

    for images, labels in train_loader:

        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running_loss += loss.item() * images.size(0)

        _, predicted = torch.max(outputs, 1)

        running_correct += (predicted == labels).sum().item()

```

```

        total += labels.size(0)

train_loss = running_loss / total
train_acc = running_correct / total

# =====
# Validation
# =====
model.eval()

val_loss = 0.0
val_correct = 0
val_total = 0

with torch.no_grad():

    for images, labels in validation_loader:

        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)

        loss = criterion(outputs, labels)

        val_loss += loss.item() * images.size(0)

        _, predicted = torch.max(outputs, 1)

        val_correct += (predicted == labels).sum().item()

        val_total += labels.size(0)

val_loss /= val_total
val_acc = val_correct / val_total

scheduler.step()

history["train_loss"].append(train_loss)
history["train_acc"].append(train_acc)
history["val_loss"].append(val_loss)
history["val_acc"].append(val_acc)

print(f"Train Loss : {train_loss:.4f}")
print(f"Train Acc : {train_acc:.4f}")
print(f"Val Loss : {val_loss:.4f}")
print(f"Val Acc : {val_acc:.4f}")

if val_acc > best_accuracy:

    best_accuracy = val_acc
    best_model_weights = copy.deepcopy(model.state_dict())

```

```

torch.save(best_model_weights, "best_efficientnet_b0.pth")

print("Best model saved.")

elapsed = time.time() - start_time

model.load_state_dict(best_model_weights)

print("\n" + "=" * 70)
print("TRAINING FINISHED")
print("=" * 70)
print(f"Best Validation Accuracy : {best_accuracy:.4f}")
print(f"Training Time           : {elapsed/60:.2f} minutes")

=====
TRAINING STARTED
=====

Epoch [1/10]
-----
Train Loss : 1.1105
Train Acc  : 0.6743
Val Loss   : 0.6690
Val Acc    : 0.8400
Best model saved.

Epoch [2/10]
-----
Train Loss : 0.6114
Train Acc  : 0.8283
Val Loss   : 0.5080
Val Acc    : 0.8622
Best model saved.

Epoch [3/10]
-----
Train Loss : 0.5034
Train Acc  : 0.8393
Val Loss   : 0.4337
Val Acc    : 0.8733
Best model saved.

Epoch [4/10]
-----
Train Loss : 0.4524
Train Acc  : 0.8646
Val Loss   : 0.4315
Val Acc    : 0.8800
Best model saved.

Epoch [5/10]
-----
Train Loss : 0.4643

```

Train Acc : 0.8550
Val Loss : 0.4311
Val Acc : 0.8800

Epoch [6/10]

Train Loss : 0.4454
Train Acc : 0.8708
Val Loss : 0.4191
Val Acc : 0.8800

Epoch [7/10]

Train Loss : 0.4584
Train Acc : 0.8560
Val Loss : 0.4233
Val Acc : 0.8800

Epoch [8/10]

Train Loss : 0.4526
Train Acc : 0.8627
Val Loss : 0.4192
Val Acc : 0.8889
Best model saved.

Epoch [9/10]

Train Loss : 0.4447
Train Acc : 0.8593
Val Loss : 0.4249
Val Acc : 0.8822

Epoch [10/10]

Train Loss : 0.4348
Train Acc : 0.8689
Val Loss : 0.4228
Val Acc : 0.8800

=====
TRAINING FINISHED
=====

Best Validation Accuracy : 0.8889
Training Time : 2.35 minutes

Section 9: MODEL EVALUATION

```
# =====  
# SECTION 9: MODEL EVALUATION  
# =====
```

```

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

import numpy as np
import matplotlib.pyplot as plt

# -----
# Evaluation Mode
# -----

model.eval()

all_predictions = []
all_labels = []

with torch.no_grad():

    for images, labels in test_loader:

        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)

        _, predictions = torch.max(outputs, 1)

        all_predictions.extend(predictions.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# -----
# Performance Metrics
# -----

accuracy = accuracy_score(all_labels, all_predictions)

precision = precision_score(
    all_labels,
    all_predictions,
    average="weighted"
)

recall = recall_score(
    all_labels,
    all_predictions,
    average="weighted"
)

```

```

)

f1 = f1_score(
    all_labels,
    all_predictions,
    average="weighted"
)

print("="*70)
print("MODEL EVALUATION")
print("="*70)

print(f"Test Accuracy : {accuracy:.4f}")
print(f"Precision      : {precision:.4f}")
print(f"Recall          : {recall:.4f}")
print(f"F1-Score        : {f1:.4f}")

print("="*70)

# -----
# Classification Report
# -----

print("\nClassification Report\n")

print(
    classification_report(
        all_labels,
        all_predictions,
        target_names=classes
    )
)

# -----
# Confusion Matrix
# -----

cm = confusion_matrix(all_labels, all_predictions)

plt.figure(figsize=(8,6))

plt.imshow(cm, interpolation="nearest", cmap="Blues")

plt.title("Confusion Matrix")

plt.colorbar()

tick_marks = np.arange(len(classes))

plt.xticks(tick_marks, classes, rotation=45)

plt.yticks(tick_marks, classes)

```

```

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(
            j,
            i,
            cm[i, j],
            ha="center",
            va="center",
            color="black"
        )

```

```

plt.xlabel("Predicted Label")
plt.ylabel("True Label")

```

```

plt.tight_layout()

```

```

plt.show()

```

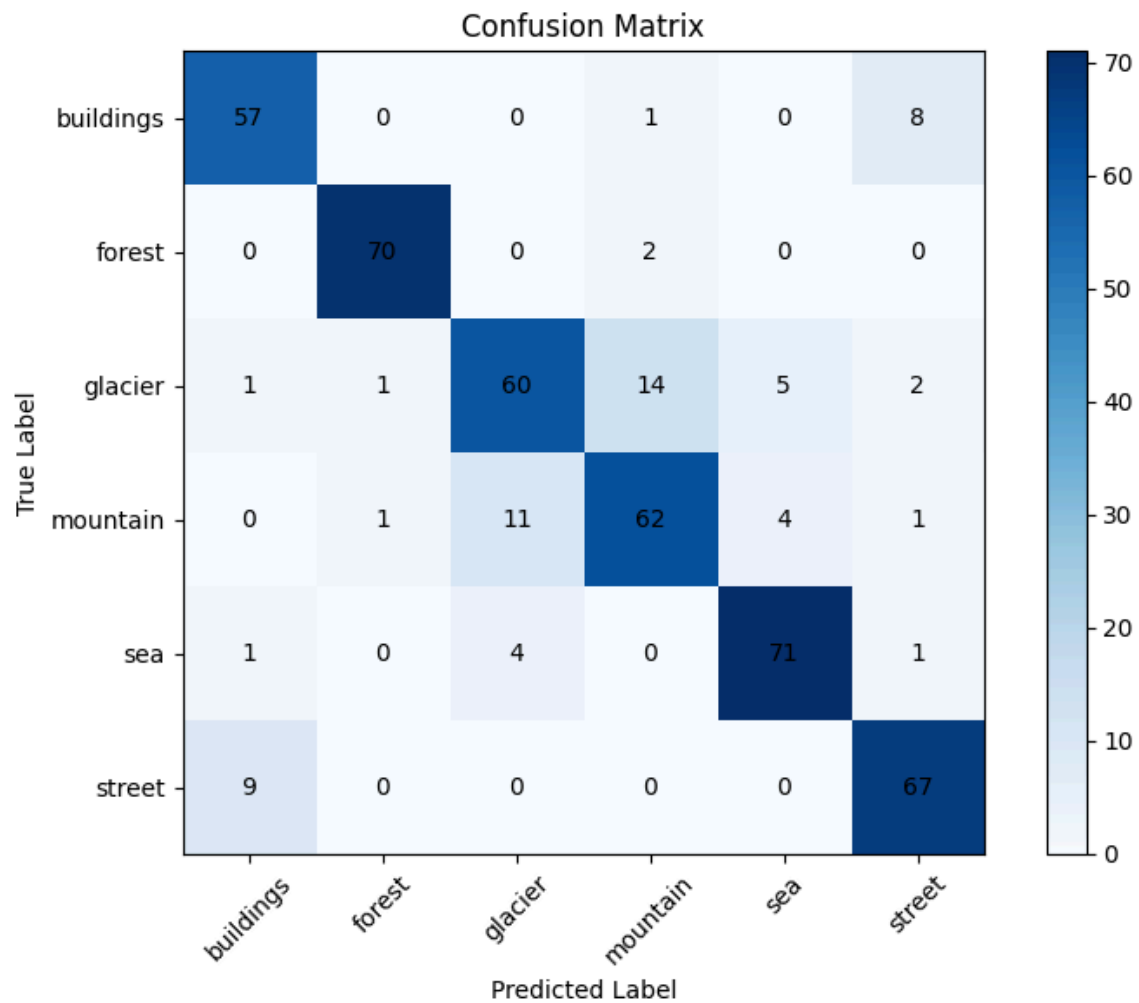
```

=====
MODEL EVALUATION
=====
Test Accuracy : 0.8543
Precision      : 0.8532
Recall         : 0.8543
F1-Score       : 0.8533
=====

```

Classification Report

	precision	recall	f1-score	support
buildings	0.84	0.86	0.85	66
forest	0.97	0.97	0.97	72
glacier	0.80	0.72	0.76	83
mountain	0.78	0.78	0.78	79
sea	0.89	0.92	0.90	77
street	0.85	0.88	0.86	76
accuracy			0.85	453
macro avg	0.86	0.86	0.86	453
weighted avg	0.85	0.85	0.85	453



SECTION 10: TRAINING CURVES

```
# =====
# SECTION 10: TRAINING CURVES
# =====

import matplotlib.pyplot as plt

# -----
# Accuracy Curve
# -----

plt.figure(figsize=(8,5))

plt.plot(
    history["train_acc"],
    marker="o",
    label="Training Accuracy"
)

plt.plot(
```

```
        history["val_acc"],
        marker="o",
        label="Validation Accuracy"
    )

plt.title("Training vs Validation Accuracy")

plt.xlabel("Epoch")

plt.ylabel("Accuracy")

plt.legend()

plt.grid(True)

plt.show()

# -----
# Loss Curve
# -----

plt.figure(figsize=(8,5))

plt.plot(
    history["train_loss"],
    marker="o",
    label="Training Loss"
)

plt.plot(
    history["val_loss"],
    marker="o",
    label="Validation Loss"
)

plt.title("Training vs Validation Loss")

plt.xlabel("Epoch")

plt.ylabel("Loss")

plt.legend()

plt.grid(True)

plt.show()
```

