

Student Name: Ammar Mohammad Wassas

Course Name: Practical_Image_Processing_and_Natural_Language_Processing_Second
Trimester 2026

Project 1

```
# Task #1: Download the image dataset
# Dataset Name: Intel Image Classification
# Kaggle Link: https://www.kaggle.com/datasets/puneet6060/intel-image-
classification

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

# Point to the Intel Image Classification folder and verify it exists
import os

base_path = '/content/drive/MyDrive/Colab Notebooks/Datasets/other
datasets/Intel Image Classification'

print("Exists:", os.path.exists(base_path))
print("Contents:", os.listdir(base_path))

Exists: True
Contents: ['seg_pred', 'seg_test', 'seg_train']

# Inspect the full folder structure and file counts per class
for root, dirs, files in os.walk(base_path):
    level = root.replace(base_path, '').count(os.sep)
    indent = ' ' * 2 * level
    print(f"{indent}{os.path.basename(root)}/ ({len(files)} files)")
    if level > 3:
        dirs[:] = []

Intel Image Classification/ (0 files)
  seg_pred/ (0 files)
    seg_pred/ (7301 files)
  seg_test/ (0 files)
    seg_test/ (0 files)
      buildings/ (437 files)
      glacier/ (553 files)
      mountain/ (525 files)
      forest/ (474 files)
      street/ (500 files)
      sea/ (510 files)
  seg_train/ (0 files)
    seg_train/ (0 files)
```

```
sea/ (2274 files)
glacier/ (2404 files)
street/ (2382 files)
mountain/ (2512 files)
buildings/ (2191 files)
forest/ (2271 files)
```

```
# Task #2: Splitting the Dataset
```

```
# Merge seg_train and seg_test into one flat structure per class
```

```
import os, shutil
```

```
from sklearn.model_selection import train_test_split
```

```
merged_dir = 'intel_merged'
```

```
os.makedirs(merged_dir, exist_ok=True)
```

```
source_dirs = [
```

```
    os.path.join(base_path, 'seg_train', 'seg_train'),
```

```
    os.path.join(base_path, 'seg_test', 'seg_test')
```

```
]
```

```
for source in source_dirs:
```

```
    source_label = 'train' if 'seg_train' in source else 'test'
```

```
    for class_name in os.listdir(source):
```

```
        class_path = os.path.join(source, class_name)
```

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

```
            continue
```

```
        merged_class_path = os.path.join(merged_dir, class_name)
```

```
        os.makedirs(merged_class_path, exist_ok=True)
```

```
        for fname in os.listdir(class_path):
```

```
            src_file = os.path.join(class_path, fname)
```

```
            dst_file = os.path.join(merged_class_path,
```

```
f"{source_label}_{fname}")
```

```
            shutil.copy(src_file, dst_file)
```

```
print("Merge complete.")
```

```
for class_name in os.listdir(merged_dir):
```

```
    count = len(os.listdir(os.path.join(merged_dir, class_name)))
```

```
    print(f"{class_name}: {count}")
```

```
Merge complete.
```

```
sea: 2784
```

```
forest: 2745
```

```
street: 2882
```

```
mountain: 3037
```

```
glacier: 2957
```

```
buildings: 2628
```

```
# Split merged dataset into train/validation/test (70/10/20)
```

```
original_dataset_dir = merged_dir
```

```
base_dir = 'intel_split'
```

```

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

for d in [train_dir, validation_dir, test_dir]:
    os.makedirs(d, exist_ok=True)

train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2

def split_and_copy_files(class_dir, dest_base_dir):
    all_images = [os.path.join(class_dir, f) for f in
os.listdir(class_dir)
                    if os.path.isfile(os.path.join(class_dir, f))]
    train_images, temp_images = train_test_split(
        all_images, test_size=(1 - train_ratio), random_state=42
    )
    validation_images, test_images = train_test_split(
        temp_images, test_size=test_ratio/(validation_ratio +
test_ratio), random_state=42
    )
    for images, folder_name in zip(
        [train_images, validation_images, test_images],
        ['train', 'validation', 'test']
    ):
        dest_dir = os.path.join(dest_base_dir, folder_name,
os.path.basename(class_dir))
        os.makedirs(dest_dir, exist_ok=True)
        for file in images:
            shutil.copy(file, dest_dir)

for class_name in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_name)
    if os.path.isdir(full_class_dir):
        split_and_copy_files(full_class_dir, base_dir)

print("Splitting complete.")

Splitting complete.

# Verify the split counts
for split in ['train', 'validation', 'test']:
    split_path = os.path.join(base_dir, split)
    total = 0
    for class_name in os.listdir(split_path):
        class_path = os.path.join(split_path, class_name)
        count = len(os.listdir(class_path))
        total += count
        print(f"{split}/{class_name}: {count}")
    print(f" → {split} total: {total}\n")

```

```
train/sea: 1948
train/forest: 1921
train/street: 2017
train/mountain: 2125
train/glacier: 2069
train/buildings: 1839
→ train total: 11919
```

```
validation/sea: 278
validation/forest: 274
validation/street: 288
validation/mountain: 304
validation/glacier: 296
validation/buildings: 263
→ validation total: 1703
```

```
test/sea: 558
test/forest: 550
test/street: 577
test/mountain: 608
test/glacier: 592
test/buildings: 526
→ test total: 3411
```

```
# Task #3: Data Augmentation
```

```
# Define data loaders with augmentation
```

```
import os
```

```
import torch
```

```
from torchvision import datasets, transforms as T
```

```
from torch.utils.data import DataLoader
```

```
def get_data_loaders(data_dir, batch_size, img_size=(224, 224),
train=False):
```

```
    """Prepare data loaders for training, validation, and testing."""
```

```
    if train:
```

```
        # Data augmentation for training
```

```
        transform = T.Compose([
```

```
            T.RandomHorizontalFlip(),
```

```
            T.RandomVerticalFlip(),
```

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

```
            T.Resize(img_size),
```

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

```
            T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
```

```
        ])
```

```
        train_data = datasets.ImageFolder(os.path.join(data_dir,
"train/"), transform=transform)
```

```
        train_loader = DataLoader(train_data, batch_size=batch_size,
shuffle=True, num_workers=2)
```

```

        return train_loader, len(train_data)
    else:
        # Validation and test data transformations
        transform = T.Compose([
            T.Resize(img_size),
            T.ToTensor(),
            T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
        ])
        val_data = datasets.ImageFolder(os.path.join(data_dir,
"validation/"), transform=transform)
        test_data = datasets.ImageFolder(os.path.join(data_dir,
"test/"), transform=transform)
        val_loader = DataLoader(val_data, batch_size=batch_size,
shuffle=False, num_workers=2)
        test_loader = DataLoader(test_data, batch_size=batch_size,
shuffle=False, num_workers=2)
        return val_loader, test_loader, len(val_data), len(test_data)

def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes

# Build the data loaders and confirm dataset sizes / device
dataset_path = "intel_split"
batch_size = 32

train_loader, train_data_len = get_data_loaders(dataset_path,
batch_size, train=True)
val_loader, test_loader, valid_data_len, test_data_len =
get_data_loaders(dataset_path, batch_size, train=False)

classes = get_classes(os.path.join(dataset_path, "train"))
num_classes = len(classes)

dataloaders = {"train": train_loader, "val": val_loader}
dataset_sizes = {"train": train_data_len, "val": valid_data_len}

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

print(f"Classes: {classes}")
print(f"Train: {train_data_len}, Validation: {valid_data_len}, Test:
{test_data_len}")
print(f"Device: {device}")

Classes: ['buildings', 'forest', 'glacier', 'mountain', 'sea',
'street']
Train: 11919, Validation: 1703, Test: 3411
Device: cuda

```

```
# Visualize a few augmented training samples
import matplotlib.pyplot as plt
import numpy as np

def denormalize(tensor):
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    img = tensor.numpy().transpose((1, 2, 0))
    img = std * img + mean
    return np.clip(img, 0, 1)

images, labels = next(iter(train_loader))
fig, axes = plt.subplots(1, 5, figsize=(15, 3))
for i in range(5):
    axes[i].imshow(denormalize(images[i]))
    axes[i].set_title(classes[labels[i]])
    axes[i].axis('off')
plt.show()
```



```
# Task #4: Training and testing models

# Custom CNN (Keras)
# TASK: Imports for Keras/TensorFlow models
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Data generators
def create_data_generators(train_dir, val_dir, test_dir,
img_size=(240, 240), batch_size=32):
    # Data augmentation for training dataset
    train_datagen = ImageDataGenerator(
        rescale=1./255,
```

```

        rotation_range=360,
        horizontal_flip=True,
        vertical_flip=True,
        width_shift_range=0.1,
        height_shift_range=0.1,
    )
    # Only rescale for validation and testing
    val_test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir, target_size=img_size, batch_size=batch_size,
        class_mode='categorical', shuffle=True
    )
    val_generator = val_test_datagen.flow_from_directory(
        val_dir, target_size=img_size, batch_size=batch_size,
        class_mode='categorical', shuffle=False
    )
    test_generator = val_test_datagen.flow_from_directory(
        test_dir, target_size=img_size, batch_size=batch_size,
        class_mode='categorical', shuffle=False
    )
    return train_generator, val_generator, test_generator

# Build the custom CNN model
def build_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu',
        input_shape=input_shape),
        MaxPooling2D(2, 2),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Set up directories, build generators and model, then train
train_dir = 'intel_split/train'

```

```

val_dir = 'intel_split/validation'
test_dir = 'intel_split/test'

train_gen, val_gen, test_gen = create_data_generators(train_dir,
val_dir, test_dir)

input_shape = (240, 240, 3)
num_classes = len(train_gen.class_indices)

model = build_cnn_model(input_shape, num_classes)

history = model.fit(train_gen, epochs=10, validation_data=val_gen)

Found 11919 images belonging to 6 classes.
Found 1703 images belonging to 6 classes.
Found 3411 images belonging to 6 classes.

/usr/local/lib/python3.12/dist-packages/keras/src/layers/
convolutional/base_conv.py:113: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/10
373/373 _____ 200s 513ms/step - accuracy: 0.4263 -
loss: 1.4174 - val_accuracy: 0.5285 - val_loss: 1.2095
Epoch 2/10
373/373 _____ 181s 484ms/step - accuracy: 0.5217 -
loss: 1.1922 - val_accuracy: 0.5843 - val_loss: 1.0949
Epoch 3/10
373/373 _____ 179s 480ms/step - accuracy: 0.5585 -
loss: 1.1358 - val_accuracy: 0.6389 - val_loss: 1.0075
Epoch 4/10
373/373 _____ 205s 488ms/step - accuracy: 0.5755 -
loss: 1.0940 - val_accuracy: 0.6377 - val_loss: 0.9745
Epoch 5/10
373/373 _____ 183s 491ms/step - accuracy: 0.6057 -
loss: 1.0339 - val_accuracy: 0.6442 - val_loss: 0.9372
Epoch 6/10
373/373 _____ 184s 493ms/step - accuracy: 0.6280 -
loss: 1.0016 - val_accuracy: 0.6812 - val_loss: 0.8759
Epoch 7/10
373/373 _____ 180s 482ms/step - accuracy: 0.6333 -
loss: 0.9765 - val_accuracy: 0.6524 - val_loss: 0.9521
Epoch 8/10
373/373 _____ 184s 493ms/step - accuracy: 0.6386 -
loss: 0.9523 - val_accuracy: 0.6665 - val_loss: 0.9384
Epoch 9/10

```

```
373/373 _____ 205s 551ms/step - accuracy: 0.6498 -  
loss: 0.9369 - val_accuracy: 0.6759 - val_loss: 0.8768  
Epoch 10/10
```

```
373/373 _____ 185s 496ms/step - accuracy: 0.6494 -  
loss: 0.9311 - val_accuracy: 0.6970 - val_loss: 0.8243
```

```
# Evaluate the CNN model
```

```
def evaluate_model(model_name, y_true, y_pred):  
    labels = list(test_gen.class_indices.keys())  
    accuracy = accuracy_score(y_true, y_pred)  
    precision = precision_score(y_true, y_pred, average='weighted')  
    recall = recall_score(y_true, y_pred, average='weighted')  
    f1 = f1_score(y_true, y_pred, average='weighted')  
    cm = confusion_matrix(y_true, y_pred)  
  
    print("Accuracy:", accuracy)  
    print("Precision :", precision)  
    print("Recall:", recall)  
    print("F1-score:", f1)  
    print("Classification Report:")  
    print(classification_report(y_true, y_pred, target_names=labels))
```

```
    plt.figure(figsize=(6, 5))  
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',  
xticklabels=labels, yticklabels=labels)  
    plt.xlabel("Predicted Label")  
    plt.ylabel("True Label")  
    plt.title("Confusion Matrix")  
    plt.show()
```

```
y_pred = model.predict(test_gen)  
y_true = test_gen.classes  
y_pred_classes = np.argmax(y_pred, axis=1)  
evaluate_model('CNN', y_true, y_pred_classes)
```

```
107/107 _____ 6s 57ms/step
```

```
Accuracy: 0.7024333040164175
```

```
Precision : 0.6981355801180967
```

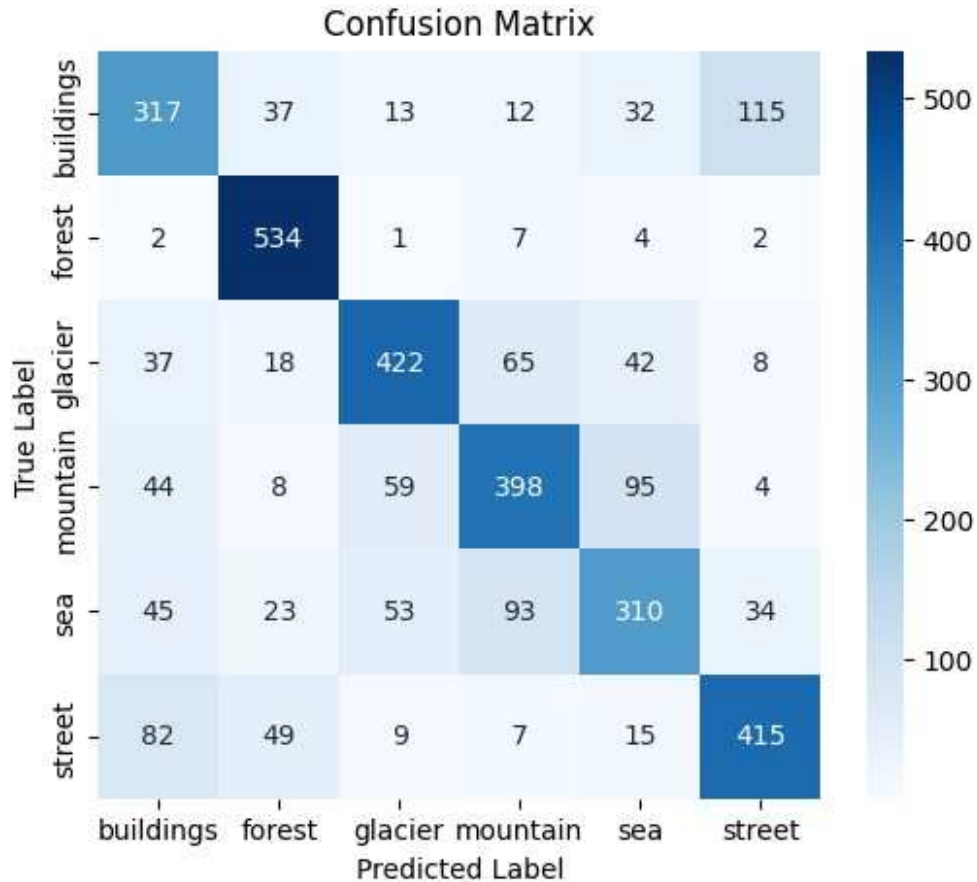
```
Recall: 0.7024333040164175
```

```
F1-score: 0.6984386175532151
```

```
Classification Report:
```

	precision	recall	f1-score	support
buildings	0.60	0.60	0.60	526
forest	0.80	0.97	0.88	550
glacier	0.76	0.71	0.73	592
mountain	0.68	0.65	0.67	608
sea	0.62	0.56	0.59	558
street	0.72	0.72	0.72	577

accuracy			0.70	3411
macro avg	0.70	0.70	0.70	3411
weighted avg	0.70	0.70	0.70	3411



```
# Model 2: ResNet50 (Keras)
# Imports for ResNet50
from tensorflow.keras.layers import Dense, Flatten, Dropout,
GlobalAveragePooling2D
from tensorflow.keras.applications import ResNet50

# Build ResNet50 model
def build_resnet8_model(input_shape, num_classes):
    base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False # Freeze the base model

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
```

```

        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Train ResNet50
model = build_resnet8_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)

Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/
resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 _____ 0s 0us/step
Epoch 1/5
373/373 _____ 233s 584ms/step - accuracy: 0.1968 -
loss: 1.7653 - val_accuracy: 0.2672 - val_loss: 1.7058
Epoch 2/5
373/373 _____ 196s 527ms/step - accuracy: 0.2122 -
loss: 1.7120 - val_accuracy: 0.2819 - val_loss: 1.6628
Epoch 3/5
373/373 _____ 197s 528ms/step - accuracy: 0.2199 -
loss: 1.6997 - val_accuracy: 0.2801 - val_loss: 1.6801
Epoch 4/5
373/373 _____ 195s 524ms/step - accuracy: 0.2273 -
loss: 1.6858 - val_accuracy: 0.2842 - val_loss: 1.6442
Epoch 5/5
373/373 _____ 198s 514ms/step - accuracy: 0.2315 -
loss: 1.6770 - val_accuracy: 0.2918 - val_loss: 1.6625

# Evaluate ResNet50
y_pred = model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('ResNet50', y_true, y_pred_classes)

107/107 _____ 22s 171ms/step
Accuracy: 0.2799765464673116
Precision : 0.19520564432089502
Recall: 0.2799765464673116
F1-score: 0.2107977884237209
Classification Report:

```

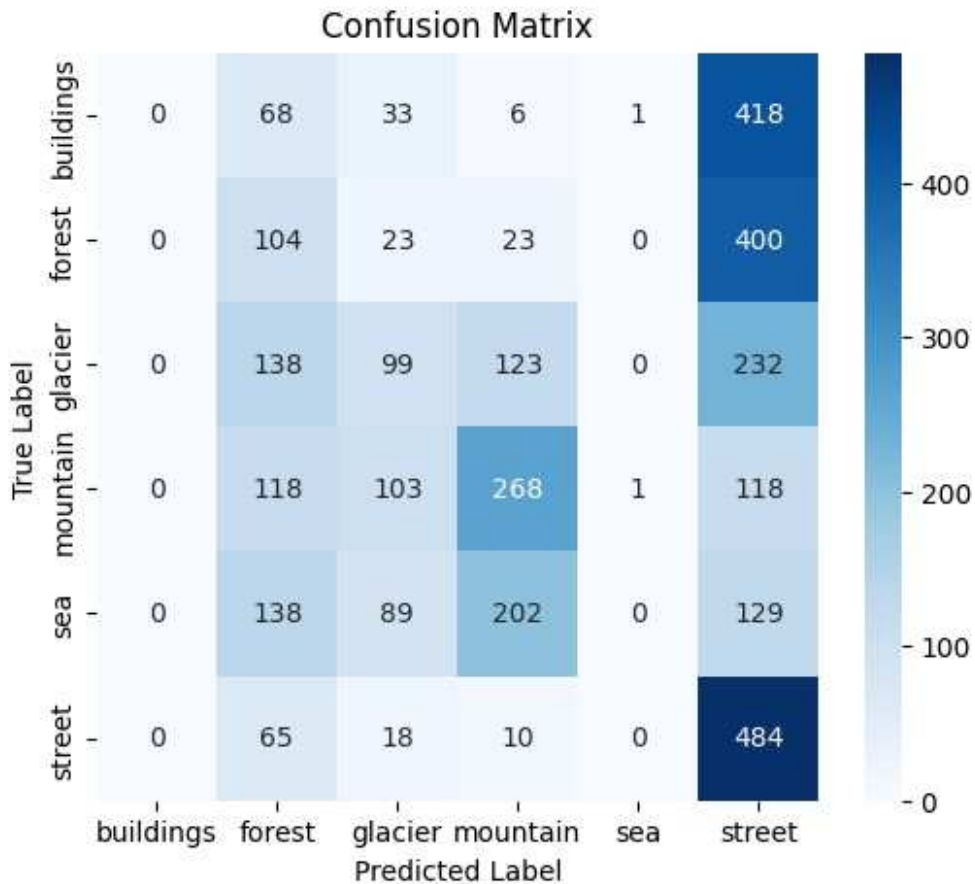
	precision	recall	f1-score	support
buildings	0.00	0.00	0.00	526
forest	0.16	0.19	0.18	550
glacier	0.27	0.17	0.21	592

mountain	0.42	0.44	0.43	608
sea	0.00	0.00	0.00	558
street	0.27	0.84	0.41	577
accuracy			0.28	3411
macro avg	0.19	0.27	0.20	3411
weighted avg	0.20	0.28	0.21	3411

```

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_
_classification.py:1565: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

```



```
# Model 3: EfficientNetB0 (Keras)
# Imports for EfficientNetB0
from tensorflow.keras.applications import import EfficientNetB0

# Build EfficientNetB0 model
def build_efficientnetb0_model(input_shape, num_classes):
    base_model = EfficientNetB0(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False # Freeze the base model

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
return model
```

```
# Train EfficientNetB0
```

```
model = build_efficientnetb0_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)
```

```
Downloading data from https://storage.googleapis.com/keras-
applications/efficientnetb0_notop.h5
```

```
16705208/16705208 _____ 0s 0us/step
```

```
Epoch 1/5
```

```
373/373 _____ 250s 578ms/step - accuracy: 0.1741 -
```

```
loss: 1.7951 - val_accuracy: 0.1785 - val_loss: 1.7906
```

```
Epoch 2/5
```

```
373/373 _____ 180s 483ms/step - accuracy: 0.1783 -
```

```
loss: 1.7909 - val_accuracy: 0.1785 - val_loss: 1.7906
```

```
Epoch 3/5
```

```
373/373 _____ 182s 488ms/step - accuracy: 0.1783 -
```

```
loss: 1.7908 - val_accuracy: 0.1785 - val_loss: 1.7906
```

```
Epoch 4/5
```

```
373/373 _____ 184s 494ms/step - accuracy: 0.1783 -
```

```
loss: 1.7908 - val_accuracy: 0.1785 - val_loss: 1.7906
```

```
Epoch 5/5
```

```
373/373 _____ 185s 496ms/step - accuracy: 0.1783 -
```

```
loss: 1.7908 - val_accuracy: 0.1785 - val_loss: 1.7906
```

```
# Evaluate EfficientNetB0
```

```
y_pred = model.predict(test_gen)
```

```
y_true = test_gen.classes
```

```
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
evaluate_model('EfficientNetB0', y_true, y_pred_classes)
```

```
107/107 _____ 27s 193ms/step
```

```
Accuracy: 0.178246848431545
```

```
Precision : 0.03177193897577818
```

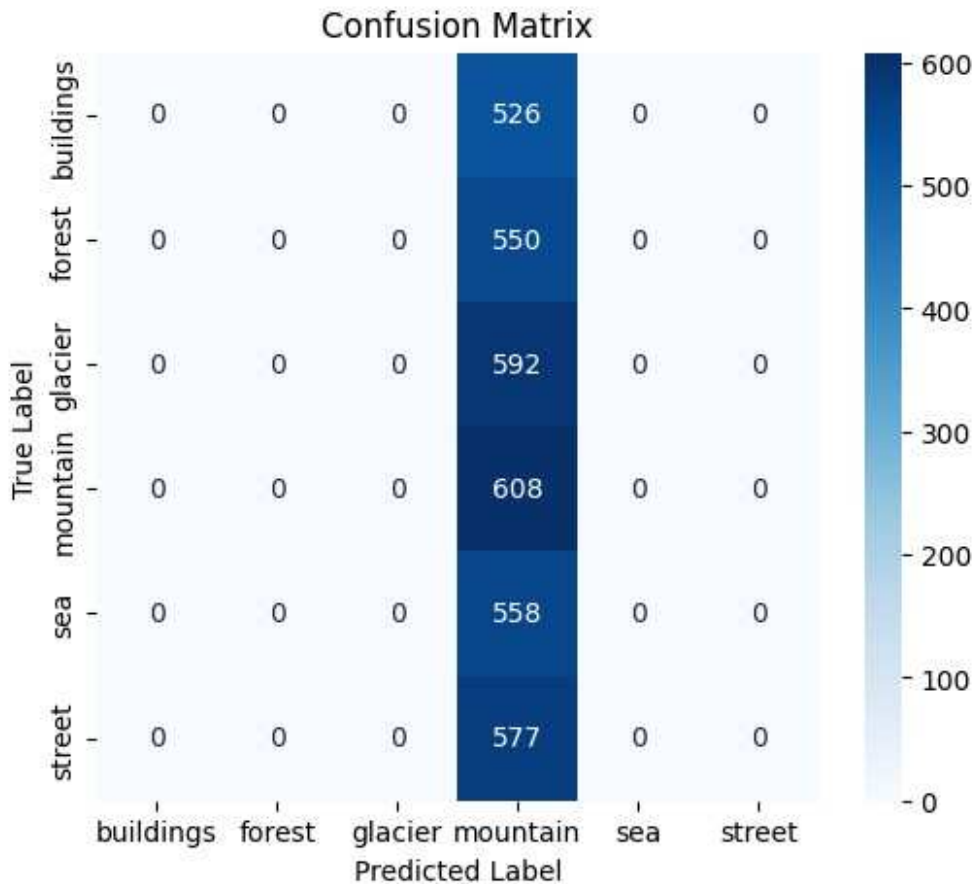
```
Recall: 0.178246848431545
```

```
F1-score: 0.05393087028931543
```

```
Classification Report:
```

	precision	recall	f1-score	support
buildings	0.00	0.00	0.00	526
forest	0.00	0.00	0.00	550
glacier	0.00	0.00	0.00	592
mountain	0.18	1.00	0.30	608
sea	0.00	0.00	0.00	558
street	0.00	0.00	0.00	577
accuracy			0.18	3411
macro avg	0.03	0.17	0.05	3411
weighted avg	0.03	0.18	0.05	3411

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```



```
# Model 4: InceptionV3 (Keras)
# Imports for InceptionV3
from tensorflow.keras.applications import InceptionV3

# Build InceptionV3 model
def build_inceptionv3_model(input_shape, num_classes):
    base_model = InceptionV3(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False # Freeze the base model

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
return model
```

Train InceptionV3

```
model = build_inceptionv3_model(input_shape, num_classes)
history = model.fit(train_gen, epochs=5, validation_data=val_gen)
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/

inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5

87910968/87910968 _____ 1s 0us/step

Epoch 1/5

373/373 _____ 234s 577ms/step - accuracy: 0.7211 -
loss: 0.7187 - val_accuracy: 0.8526 - val_loss: 0.3906

Epoch 2/5

373/373 _____ 187s 501ms/step - accuracy: 0.7801 -
loss: 0.5609 - val_accuracy: 0.8444 - val_loss: 0.3808

Epoch 3/5

373/373 _____ 204s 507ms/step - accuracy: 0.8029 -
loss: 0.5271 - val_accuracy: 0.8550 - val_loss: 0.3525

Epoch 4/5

373/373 _____ 190s 508ms/step - accuracy: 0.8080 -
loss: 0.5070 - val_accuracy: 0.8614 - val_loss: 0.3461

Epoch 5/5

373/373 _____ 186s 500ms/step - accuracy: 0.8117 -
loss: 0.5001 - val_accuracy: 0.8644 - val_loss: 0.3374

Evaluate InceptionV3

```
y_pred = model.predict(test_gen)
```

```
y_true = test_gen.classes
```

```
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
evaluate_model('InceptionV3', y_true, y_pred_classes)
```

107/107 _____ 25s 174ms/step

Accuracy: 0.8792143066549399

Precision : 0.8805006722274882

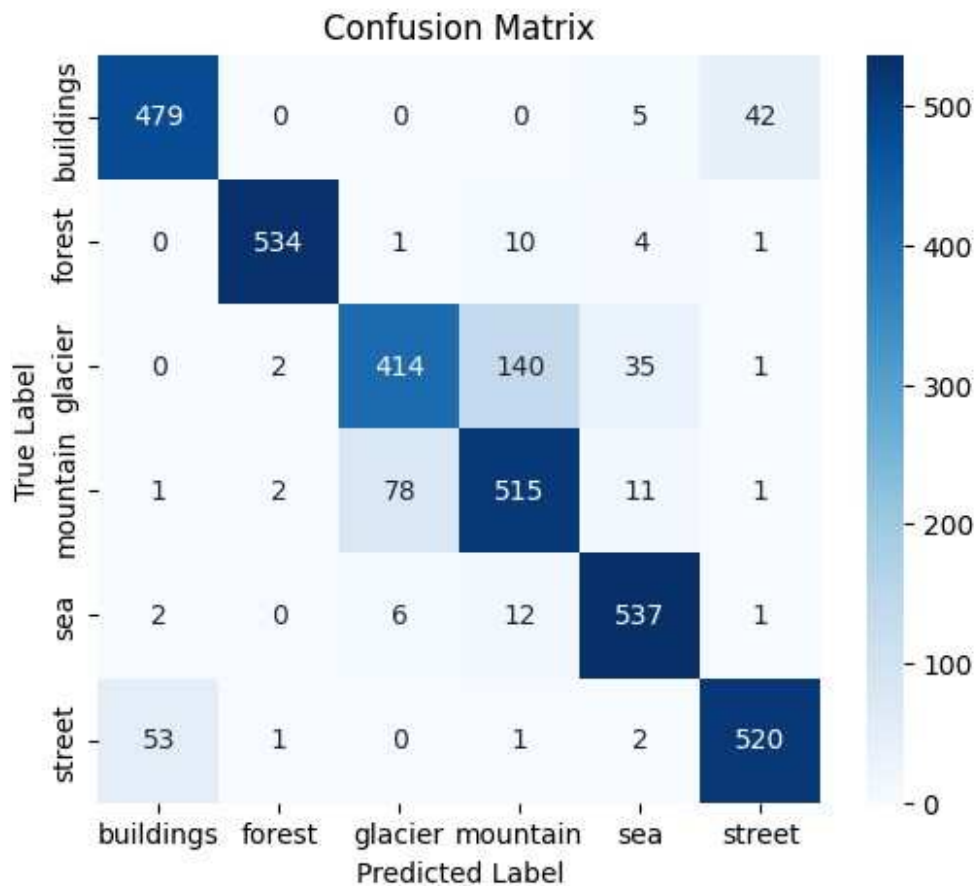
Recall: 0.8792143066549399

F1-score: 0.878279629312394

Classification Report:

	precision	recall	f1-score	support
buildings	0.90	0.91	0.90	526
forest	0.99	0.97	0.98	550
glacier	0.83	0.70	0.76	592
mountain	0.76	0.85	0.80	608
sea	0.90	0.96	0.93	558
street	0.92	0.90	0.91	577
accuracy			0.88	3411
macro avg	0.88	0.88	0.88	3411

weighted avg	0.88	0.88	0.88	3411
--------------	------	------	------	------



```
# Model 5: EfficientNetB0 (PyTorch)
# Imports for PyTorch models
import torch
from torchvision import models
from torch import nn, optim
from tqdm import tqdm
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report

# Shared training loop
def train_model(model, criterion, optimizer, scheduler,
num_epochs=20):
    """Train and validate the model."""
    for epoch in range(num_epochs):
        print(f"Epoch {epoch+1}/{num_epochs}")
        print('-' * 10)
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
```

```

        else:
            model.eval()

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in dataloaders[phase]:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                if isinstance(outputs, tuple) or hasattr(outputs,
"logits"):
                    main_outputs = outputs.logits if
hasattr(outputs, "logits") else outputs[0]
                    aux_outputs = outputs.aux_logits if
hasattr(outputs, "aux_logits") else None
                    loss = criterion(main_outputs, labels)
                    if aux_outputs is not None:
                        loss += 0.4 * criterion(aux_outputs,
labels)
                else:
                    main_outputs = outputs
                    loss = criterion(main_outputs, labels)
                _, preds = torch.max(main_outputs, 1)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        if phase == 'train':
            scheduler.step()

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() /
dataset_sizes[phase]
        print(f"{phase} Loss: {epoch_loss:.4f} Acc:
{epoch_acc:.4f}")

    return model

# Build, compile and train EfficientNetB0 (PyTorch)
model = models.efficientnet_b0(pretrained=True)
model.classifier[1] = nn.Linear(model.classifier[1].in_features,

```

```

num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)

model = train_model(model, criterion, optimizer, exp_lr_scheduler,
num_epochs=2)

/usr/local/lib/python3.12/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing
`weights=EfficientNet_B0_Weights.IMAGENET1K_V1`. You can also use
`weights=EfficientNet_B0_Weights.DEFAULT` to get the most up-to-date
weights.
  warnings.warn(msg)

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, 85.5MB/s]

Epoch 1/2
-----
train Loss: 0.4370 Acc: 0.8517
val Loss: 0.3139 Acc: 0.9049
Epoch 2/2
-----
train Loss: 0.3087 Acc: 0.8959
val Loss: 0.2930 Acc: 0.8943

# Evaluate EfficientNetB0 (PyTorch)
def evaluate_model(model_name, y_true, y_pred, classes):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

```

```

print("Accuracy:", accuracy)
print("Precision :", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Classification Report:")
print(classification_report(y_true, y_pred, target_names=classes))

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=classes, yticklabels=classes)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

```

```

y_true = []
y_proba = []
y_pred = []

```

```

with torch.no_grad():
    for x, y in tqdm(test_loader):
        output = model(x.to(device))
        y_true.extend(y.numpy())
        probabilities = torch.softmax(output, dim=1)
        y_proba.extend(probabilities.cpu().numpy())
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

```

```

y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)

```

```

evaluate_model("EfficientNetB0_PyTorch", y_true, y_pred, classes)

```

```

100%|██████████| 107/107 [00:15<00:00, 7.00it/s]

```

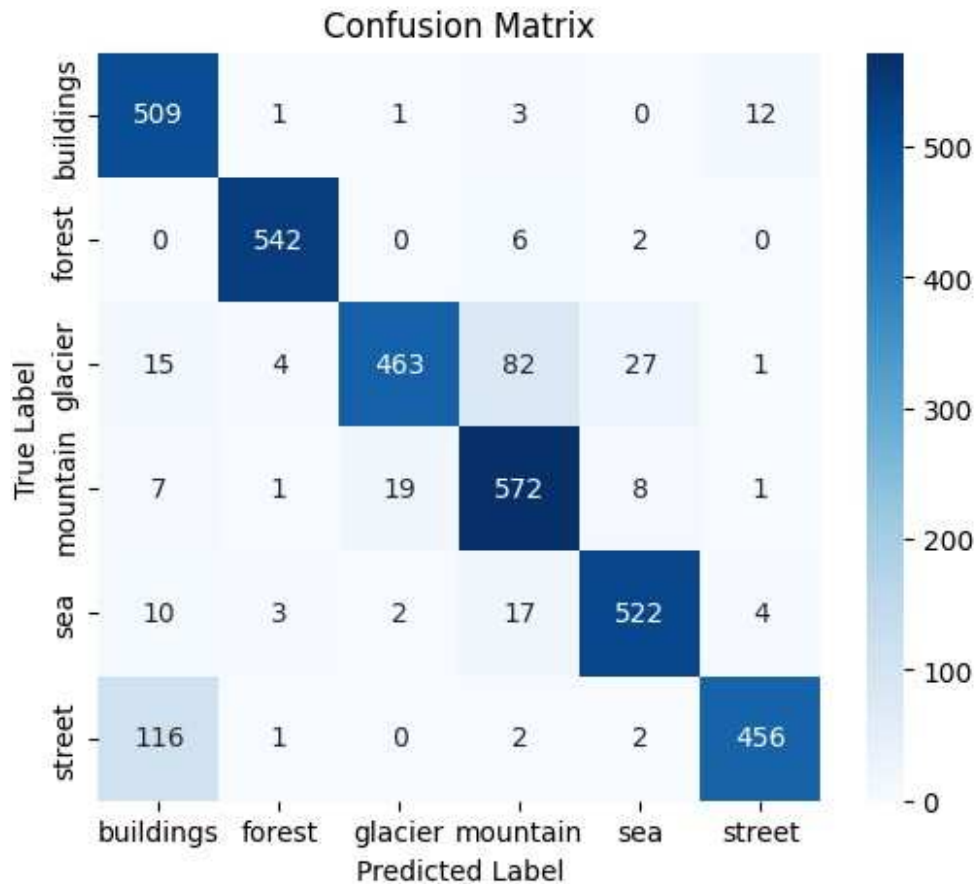
```

Accuracy: 0.8982703019642334
Precision : 0.9079229447984085
Recall: 0.8982703019642334
F1-score: 0.8980146814103588
Classification Report:

```

	precision	recall	f1-score	support
buildings	0.77	0.97	0.86	526
forest	0.98	0.99	0.98	550
glacier	0.95	0.78	0.86	592
mountain	0.84	0.94	0.89	608
sea	0.93	0.94	0.93	558
street	0.96	0.79	0.87	577

accuracy			0.90	3411
macro avg	0.91	0.90	0.90	3411
weighted avg	0.91	0.90	0.90	3411



```
# Model 6: ResNet18 (PyTorch)
# TASK: Build, compile and train ResNet18 (PyTorch)
model = models.resnet18(pretrained=False)
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)

model = train_model(model, criterion, optimizer, exp_lr_scheduler,
num_epochs=2)

Epoch 1/2
-----
```

```

/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)

```

```

train Loss: 1.0569 Acc: 0.5760
val Loss: 1.4656 Acc: 0.4974
Epoch 2/2
-----

```

```

train Loss: 0.8294 Acc: 0.6782
val Loss: 0.7819 Acc: 0.7093

```

Evaluate ResNet18 (PyTorch)

```

y_true = []
y_proba = []
y_pred = []

```

```

with torch.no_grad():
    for x, y in tqdm(test_loader):
        output = model(x.to(device))
        y_true.extend(y.numpy())
        probabilities = torch.softmax(output, dim=1)
        y_proba.extend(probabilities.cpu().numpy())
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

```

```

y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)

```

```

evaluate_model("Resnet18_PyTorch", y_true, y_pred, classes)

```

```

100%|██████████| 107/107 [00:09<00:00, 11.03it/s]

```

```

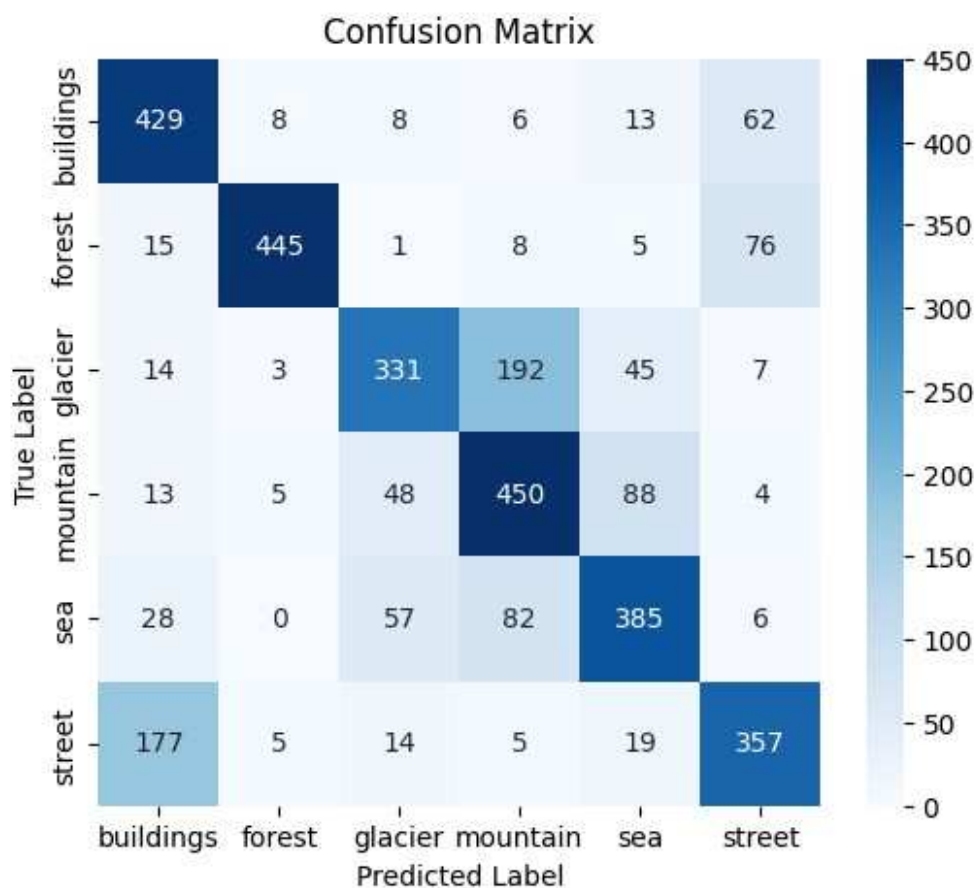
Accuracy: 0.702726473175022
Precision : 0.7163802943984956
Recall: 0.702726473175022
F1-score: 0.7034658234614137

```

Classification Report:

	precision	recall	f1-score	support
buildings	0.63	0.82	0.71	526
forest	0.95	0.81	0.88	550
glacier	0.72	0.56	0.63	592
mountain	0.61	0.74	0.67	608

sea	0.69	0.69	0.69	558
street	0.70	0.62	0.66	577
accuracy			0.70	3411
macro avg	0.72	0.71	0.71	3411
weighted avg	0.72	0.70	0.70	3411



```
# Model 7: DenseNet121 (PyTorch)
# Build, compile and train DenseNet121 (PyTorch)
model = models.densenet121(pretrained=False)
model.classifier = nn.Linear(model.classifier.in_features,
num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)
```

```
model = train_model(model, criterion, optimizer, exp_lr_scheduler,
num_epochs=1)
```

Epoch 1/1

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
```

```
train Loss: 0.9756 Acc: 0.6100
val Loss: 0.8428 Acc: 0.6864
```

Evaluate DenseNet121 (PyTorch)

```
y_true = []
y_proba = []
y_pred = []
```

```
with torch.no_grad():
    for x, y in tqdm(test_loader):
        output = model(x.to(device))
        y_true.extend(y.numpy())
        probabilities = torch.softmax(output, dim=1)
        y_proba.extend(probabilities.cpu().numpy())
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)
```

```
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)
```

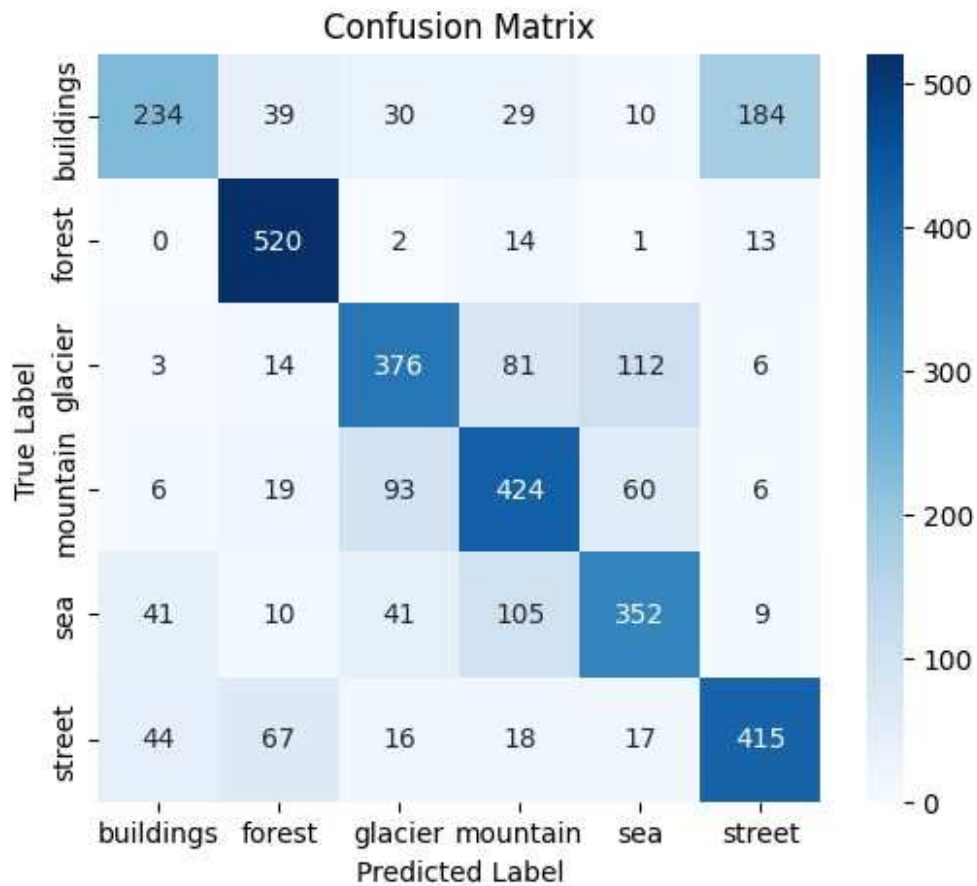
```
evaluate_model("Densenet121_PyTorch", y_true, y_pred, classes)
```

```
100%|██████████| 107/107 [00:13<00:00, 7.96it/s]
```

```
Accuracy: 0.6804456171210789
Precision : 0.6801445370071899
Recall: 0.6804456171210789
F1-score: 0.6735319710897989
Classification Report:
```

	precision	recall	f1-score	support
buildings	0.71	0.44	0.55	526
forest	0.78	0.95	0.85	550

glacier	0.67	0.64	0.65	592
mountain	0.63	0.70	0.66	608
sea	0.64	0.63	0.63	558
street	0.66	0.72	0.69	577
accuracy			0.68	3411
macro avg	0.68	0.68	0.67	3411
weighted avg	0.68	0.68	0.67	3411



```

# Model 8: VGG16 (PyTorch)
# Build, compile and train VGG16 (PyTorch)
model = models.vgg16(pretrained=False)
model.classifier[6] = nn.Linear(model.classifier[6].in_features,
num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)

```

```
model = train_model(model, criterion, optimizer, exp_lr_scheduler,
num_epochs=1)
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
```

```
warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
warnings.warn(msg)
```

Epoch 1/1

train Loss: 2.7350 Acc: 0.1738

val Loss: 1.7909 Acc: 0.1785

Evaluate VGG16 (PyTorch)

y_true = []

y_proba = []

y_pred = []

with torch.no_grad():

```
for x, y in tqdm(test_loader):
    output = model(x.to(device))
    y_true.extend(y.numpy())
    probabilities = torch.softmax(output, dim=1)
    y_proba.extend(probabilities.cpu().numpy())
    pred = torch.argmax(probabilities, axis=1).cpu().numpy()
    y_pred.extend(pred)
```

y_true = np.array(y_true)

y_proba = np.array(y_proba)

y_pred = np.array(y_pred)

evaluate_model("vgg16_PyTorch", y_true, y_pred, classes)

100%|██████████| 107/107 [00:18<00:00, 5.67it/s]

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificatio
n.py:1565: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
```

```
_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

Accuracy: 0.178246848431545

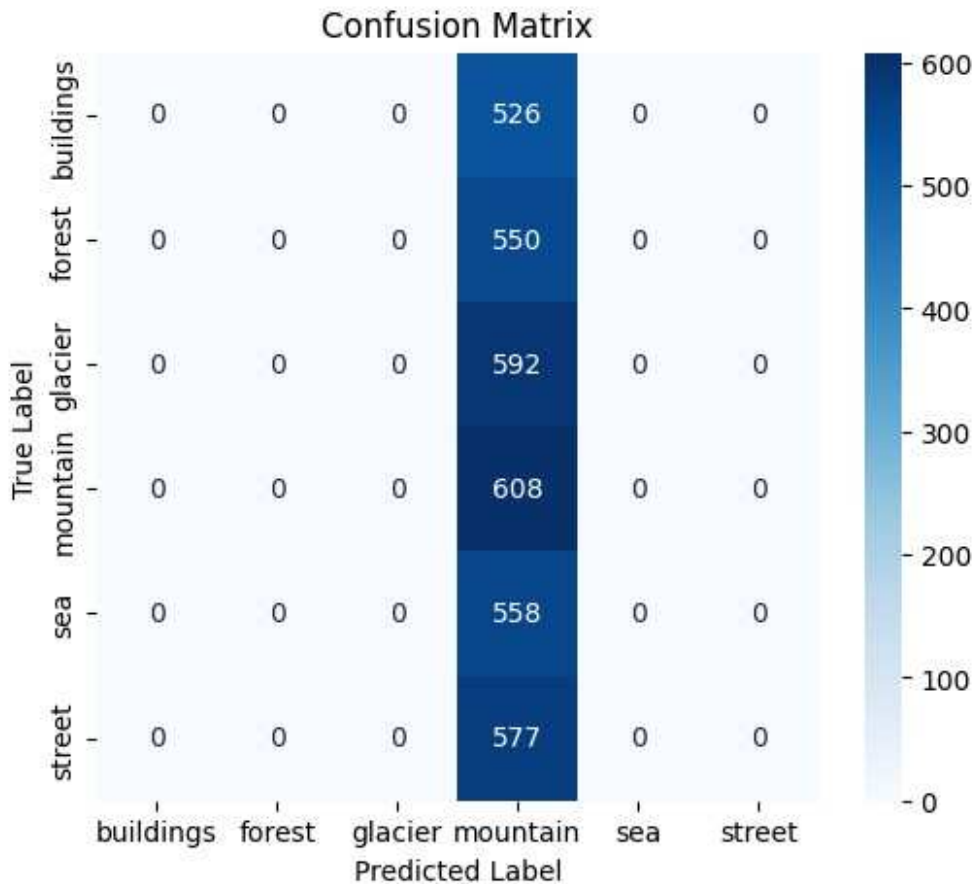
Precision : 0.03177193897577818

Recall: 0.178246848431545

F1-score: 0.05393087028931543

Classification Report:

	precision	recall	f1-score	support
buildings	0.00	0.00	0.00	526
forest	0.00	0.00	0.00	550
glacier	0.00	0.00	0.00	592
mountain	0.18	1.00	0.30	608
sea	0.00	0.00	0.00	558
street	0.00	0.00	0.00	577
accuracy			0.18	3411
macro avg	0.03	0.17	0.05	3411
weighted avg	0.03	0.18	0.05	3411



```
# Model 9: InceptionV3 (PyTorch)
# Build separate 299x299 data loaders for InceptionV3
inception_train_loader, inception_train_len = get_data_loaders(
    dataset_path, batch_size, img_size=(299, 299), train=True
)
inception_val_loader, inception_test_loader, inception_val_len,
inception_test_len = get_data_loaders(
    dataset_path, batch_size, img_size=(299, 299), train=False
)

inception_dataloaders = {"train": inception_train_loader, "val":
inception_val_loader}
inception_dataset_sizes = {"train": inception_train_len, "val":
inception_val_len}

# Build, compile and train InceptionV3 (PyTorch)
model = models.inception_v3(pretrained=False, aux_logits=True)
model.fc = nn.Linear(model.fc.in_features, num_classes)

if model.aux_logits:
    model.AuxLogits.fc = nn.Linear(model.AuxLogits.fc.in_features,
num_classes)
```

```

model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)

/usr/local/lib/python3.12/dist-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
/usr/local/lib/python3.12/dist-packages/torchvision/models/inception.p
y:43: FutureWarning: The default weight initialization of inception_v3
will be changed in future releases of torchvision. If you wish to keep
the old behavior (which leads to long initialization times due to
scipy/scipy#11299), please set init_weights=True.
  warnings.warn(

# Train InceptionV3 using the 299x299 loaders (temporarily swap in for
train_model's global references)
dataloaders = inception_dataloaders
dataset_sizes = inception_dataset_sizes

model = train_model(model, criterion, optimizer, exp_lr_scheduler,
num_epochs=1)

# Restore the original 224x224 loaders for subsequent models
dataloaders = {"train": train_loader, "val": val_loader}
dataset_sizes = {"train": train_data_len, "val": valid_data_len}

Epoch 1/1
-----
train Loss: 1.3867 Acc: 0.6011
val Loss: 0.7073 Acc: 0.7264

# Evaluate InceptionV3 (PyTorch) using the 299x299 test loader
y_true = []
y_proba = []
y_pred = []

with torch.no_grad():
    for x, y in tqdm(inception_test_loader):
        output = model(x.to(device))
        y_true.extend(y.numpy())

```

```

probabilities = torch.softmax(output, dim=1)
y_proba.extend(probabilities.cpu().numpy())
pred = torch.argmax(probabilities, axis=1).cpu().numpy()
y_pred.extend(pred)

y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)

```

```

evaluate_model("inception_v3_PyTorch", y_true, y_pred, classes)

```

```

100%|██████████| 107/107 [00:17<00:00, 6.03it/s]

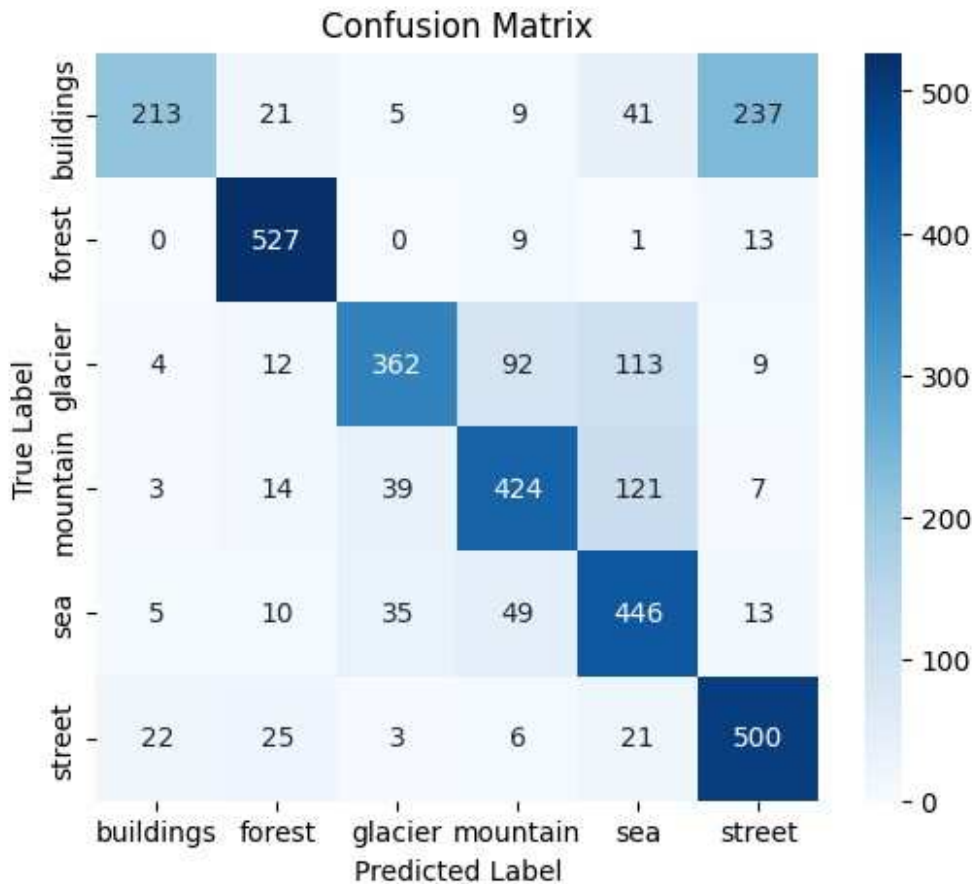
```

```

Accuracy: 0.7247141600703606
Precision : 0.7491000289008339
Recall: 0.7247141600703606
F1-score: 0.7160924014179216
Classification Report:

```

	precision	recall	f1-score	support
buildings	0.86	0.40	0.55	526
forest	0.87	0.96	0.91	550
glacier	0.82	0.61	0.70	592
mountain	0.72	0.70	0.71	608
sea	0.60	0.80	0.69	558
street	0.64	0.87	0.74	577
accuracy			0.72	3411
macro avg	0.75	0.72	0.72	3411
weighted avg	0.75	0.72	0.72	3411



```
# Model 10: ViT (Vision Transformer)
# Imports for ViT and Swin Transformer
from torchvision.models import vit_b_32
import pandas as pd
from sklearn.metrics import cohen_kappa_score, matthews_corrcoef,
ConfusionMatrixDisplay

# Evaluation function for transformer models
def evaluate_model_vit(model, dataloader):
    y_pred, y_true = [], []
    model.eval()
    with torch.no_grad():
        for inputs, labels in tqdm(dataloader):
            inputs = inputs.to(device)
            preds = torch.argmax(model(inputs), axis=1).cpu().numpy()
            y_pred.extend(preds)
            y_true.extend(labels.numpy())
    return y_true, y_pred

# Build ViT model
model = vit_b_32(pretrained=True)
n_inputs = model.heads.head.in_features
```

```

model.heads.head = nn.Sequential(
    nn.Linear(n_inputs, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, len(classes))
)
model = model.to(device)
print(model.heads.head)

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)

/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing
`weights=ViT_B_32_Weights.IMAGENET1K_V1`. You can also use
`weights=ViT_B_32_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)

Downloading: "https://download.pytorch.org/models/vit_b_32-
d86f8d99.pth" to /root/.cache/torch/hub/checkpoints/vit_b_32-
d86f8d99.pth

100%|██████████| 337M/337M [00:04<00:00, 78.6MB/s]

Sequential(
  (0): Linear(in_features=768, out_features=512, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.3, inplace=False)
  (3): Linear(in_features=512, out_features=6, bias=True)
)

# Train ViT
ViT_model = train_model(model, criterion, optimizer, scheduler,
num_epochs=3)

Epoch 1/3
-----
train Loss: 1.1083 Acc: 0.5496
val Loss: 0.9398 Acc: 0.6124
Epoch 2/3
-----

```

```
train Loss: 1.0048 Acc: 0.5971
val Loss: 0.9391 Acc: 0.6078
Epoch 3/3
```

```
-----
```

```
train Loss: 0.9356 Acc: 0.6299
val Loss: 1.0042 Acc: 0.6048
```

```
# Evaluate ViT
```

```
y_true, y_pred = evaluate_model_vit(ViT_model, test_loader)
print("Classification Report:")
print(classification_report(y_true, y_pred))
```

```
100%|██████████| 107/107 [00:12<00:00, 8.42it/s]
```

```
Classification Report:
```

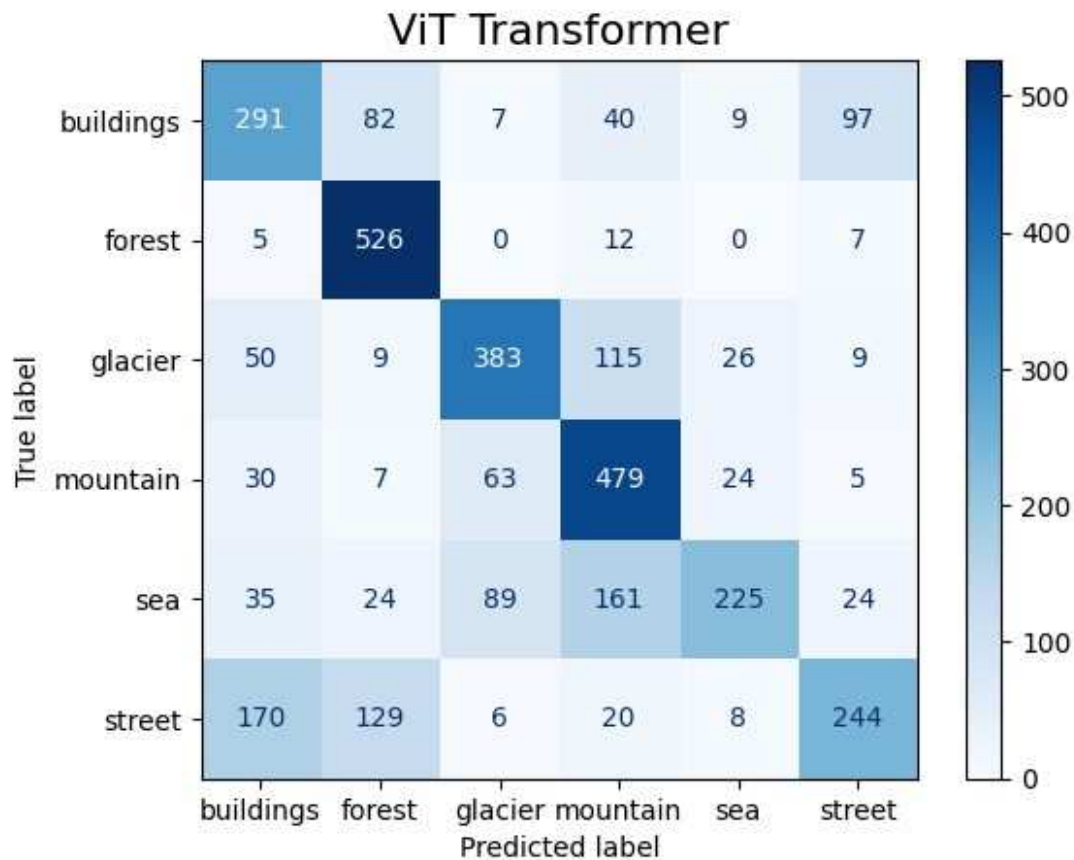
	precision	recall	f1-score	support
0	0.50	0.55	0.53	526
1	0.68	0.96	0.79	550
2	0.70	0.65	0.67	592
3	0.58	0.79	0.67	608
4	0.77	0.40	0.53	558
5	0.63	0.42	0.51	577
accuracy			0.63	3411
macro avg	0.64	0.63	0.62	3411
weighted avg	0.64	0.63	0.62	3411

```
# ViT confusion matrix and summary metrics table
```

```
disp =
ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_true,
y_pred), display_labels=classes)
disp.plot(cmap="Blues", values_format='')
plt.title("ViT Transformer", fontsize=16)
plt.show()
```

```
results = pd.DataFrame([{
    'Accuracy': round(accuracy_score(y_true, y_pred) * 100, 3),
    'Precision': round(precision_score(y_true, y_pred,
average='weighted') * 100, 3),
    'Recall': round(recall_score(y_true, y_pred, average='weighted') *
100, 3),
    'F1 Score': round(f1_score(y_true, y_pred, average='weighted') *
100, 3),
    'Kappa Statistic': round(cohen_kappa_score(y_true, y_pred) * 100,
3),
    'MCC': round(matthews_corrcoef(y_true, y_pred) * 100, 3)
```

```
}})
results
```



```
{
  "summary": {
    "name": "results",
    "rows": 1,
    "fields": [
      {
        "column": "Accuracy",
        "properties": {
          "dtype": "number",
          "std": null,
          "min": 62.973,
          "max": 62.973,
          "num_unique_values": 1,
          "samples": [62.973]
        },
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Precision",
        "properties": {
          "dtype": "number",
          "std": null,
          "min": 64.391,
          "max": 64.391,
          "num_unique_values": 1,
          "samples": [64.391]
        },
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Recall",
        "properties": {
          "dtype": "number",
          "std": null,
          "min": 62.973,
          "max": 62.973,
          "num_unique_values": 1,
          "samples": [62.973]
        },
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "F1 Score",
        "properties": {
          "dtype": "number",
          "std": null,
          "min": 61.684,
          "max": 61.684,
          "num_unique_values": 1,
          "samples": [61.684]
        },
        "semantic_type": "",
        "description": ""
      }
    ]
  }
}
```

```

{"num_unique_values": 1, "samples": [61.684], "semantic_type": "", "description": ""}, {"column": "Kappa Statistic", "properties": {"dtype": "number", "std": null, "min": 55.537, "max": 55.537}, "num_unique_values": 1, "samples": [55.537], "semantic_type": "", "description": ""}, {"column": "MCC", "properties": {"dtype": "number", "std": null, "min": 56.154, "max": 56.154}, "num_unique_values": 1, "samples": [56.154], "semantic_type": "", "description": ""}]}, {"type": "dataframe", "variable_name": "results"}

```

Model 11: Swin Transformer

Load Swin Transformer via torch.hub

```
HUB_URL = "SharanSMenon/swin-transformer-hub:main"
```

```
MODEL_NAME = "swin_tiny_patch4_window7_224"
```

```
model = torch.hub.load(HUB_URL, MODEL_NAME, pretrained=True)
```

```
model = model.to(device)
```

```
criterion = nn.CrossEntropyLoss().to(device)
```

```
optimizer = optim.AdamW(model.parameters(), lr=1e-4)
```

```
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)
```

The repository SharanSMenon_swin-transformer-hub does not belong to the list of trusted repositories and as such cannot be downloaded. Do you trust this repository and wish to add it to the trusted list of repositories (y/N)?y

Downloading:

"https://github.com/SharanSMenon/swin-transformer-hub/zipball/main" to /root/.cache/torch/hub/main.zip

```
/usr/local/lib/python3.12/dist-packages/timm/models/layers/
__init__.py:49: FutureWarning: Importing from timm.models.layers is
deprecated, please import via timm.layers
```

```
warnings.warn(f"Importing from {__name__} is deprecated, please
import via timm.layers", FutureWarning)
```

```
/usr/local/lib/python3.12/dist-packages/torch/functional.py:505:
UserWarning: torch.meshgrid: in an upcoming release, it will be
required to pass the indexing argument. (Triggered internally at
/pytorch/aten/src/ATen/native/TensorShape.cpp:4381.)
```

```
return _VF.meshgrid(tensors, **kwargs) # type: ignore[attr-defined]
```

Downloading:

"https://github.com/SwinTransformer/storage/releases/download/v1.0.0/swin_tiny_patch4_window7_224.pth" to /root/.cache/torch/hub/checkpoints/swin_tiny_patch4_window7_224.pth

100%|██████████| 109M/109M [00:02<00:00, 52.6MB/s]

Train Swin Transformer

```
Swin_model = train_model(model, criterion, optimizer, scheduler,  
num_epochs=2)
```

Epoch 1/2

train Loss: 0.4129 Acc: 0.8643

val Loss: 0.2326 Acc: 0.9084

Epoch 2/2

train Loss: 0.2179 Acc: 0.9214

val Loss: 0.1873 Acc: 0.9360

Evaluate Swin Transformer, matching Lecture 5 exactly

```
y_true, y_pred = evaluate_model_vit(Swin_model, test_loader)
```

```
print("Classification Report:")
```

```
print(classification_report(y_true, y_pred))
```

```
disp =
```

```
ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_true,  
y_pred), display_labels=classes)
```

```
disp.plot(cmap="Blues", values_format='')
```

```
plt.title("Swin Transformer", fontsize=16)
```

```
plt.show()
```

```
results = pd.DataFrame([
```

```
    'Accuracy': round(accuracy_score(y_true, y_pred) * 100, 3),
```

```
    'Precision': round(precision_score(y_true, y_pred,
```

```
average='weighted') * 100, 3),
```

```
    'Recall': round(recall_score(y_true, y_pred, average='weighted') *  
100, 3),
```

```
    'F1 Score': round(f1_score(y_true, y_pred, average='weighted') *  
100, 3),
```

```
    'Kappa Statistic': round(cohen_kappa_score(y_true, y_pred) * 100,  
3),
```

```
    'MCC': round(matthews_corrcoef(y_true, y_pred) * 100, 3)
```

```
]])
```

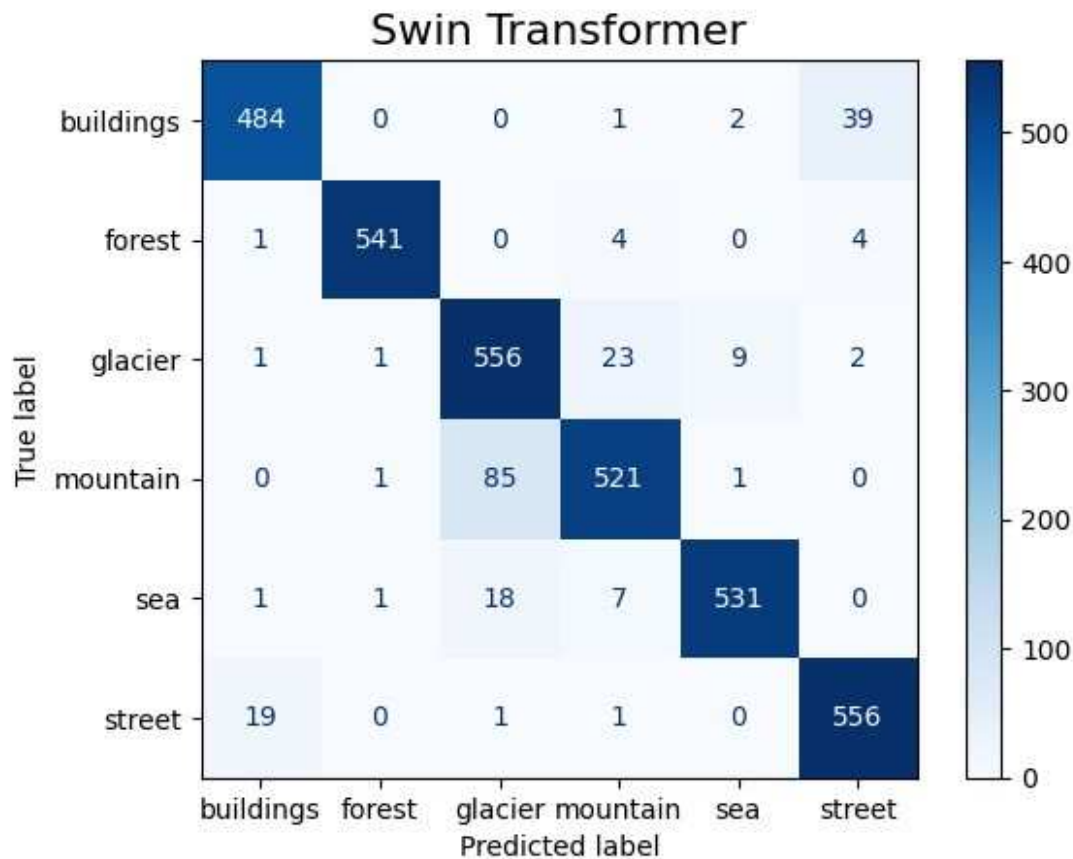
```
results
```

100%|██████████| 107/107 [00:18<00:00, 5.90it/s]

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.92	0.94	526
1	0.99	0.98	0.99	550
2	0.84	0.94	0.89	592
3	0.94	0.86	0.89	608
4	0.98	0.95	0.96	558

	5	0.93	0.96	0.94	577
accuracy				0.93	3411
macro avg		0.94	0.94	0.94	3411
weighted avg		0.94	0.93	0.94	3411



```
{
  "summary": {
    "name": "results",
    "rows": 1,
    "fields": [
      {
        "column": "Accuracy",
        "properties": {
          "dtype": "number",
          "std": null,
          "min": 93.492,
          "max": 93.492,
          "num_unique_values": 1,
          "samples": [93.492]
        },
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Precision",
        "properties": {
          "dtype": "number",
          "std": null,
          "min": 93.726,
          "max": 93.726,
          "num_unique_values": 1,
          "samples": [93.726]
        },
        "semantic_type": "",
        "description": ""
      },
      {
        "column": "Recall",
        "properties": {
          "dtype": "number",
          "std": null,
          "min": 93.492,
          "max": 93.492,
          "num_unique_values": 1,
          "samples": [93.492]
        },
        "semantic_type": "",
        "description": ""
      }
    ]
  }
}
```

```

],\n      \"semantic_type\": \"\", \n      \"description\": \"\"\n}\n  },\n  {\n    \"column\": \"F1 Score\", \n    \"properties\": {\n      \"dtype\": \"number\", \n      \"std\": \nnull,\n      \"min\": 93.517, \n      \"max\": 93.517, \n      \"num_unique_values\": 1, \n      \"samples\": [\n        93.517\n      ], \n      \"semantic_type\": \"\", \n      \"description\": \"\"\n}\n  },\n  {\n    \"column\": \"Kappa Statistic\", \n    \"properties\": {\n      \"dtype\": \"number\", \n      \"std\": \nnull,\n      \"min\": 92.186, \n      \"max\": 92.186, \n      \"num_unique_values\": 1, \n      \"samples\": [\n        92.186\n      ], \n      \"semantic_type\": \"\", \n      \"description\": \"\"\n}\n  },\n  {\n    \"column\": \"MCC\", \n    \"properties\": {\n      \"dtype\": \"number\", \n      \"std\": null, \n      \"min\": 92.226, \n      \"max\": 92.226, \n      \"num_unique_values\": 1, \n      \"samples\": [\n        92.226\n      ], \n      \"semantic_type\": \"\", \n      \"description\": \"\"\n}\n  }\n]\n} \", \"type\": \"dataframe\", \"variable_name\": \"results\"}

```

Model 12: Optuna Hyperparameter Tuning

Install Optuna

```
!pip install optuna --quiet
```

```

0.0/425.6 kB ? eta -:-:-
425.6/425.6 kB 13.4 MB/s eta
0:00:00

```

```

0.0/264.7 kB ? eta -:-:-
264.7/264.7 kB 25.8 MB/s eta
0:00:00

```

Imports for Optuna

```
import optuna
```

Define model + hyperparameter search space

```

def define_model(trial):
    lr = trial.suggest_loguniform("lr", 1e-5, 1e-2)
    weight_decay = trial.suggest_loguniform("weight_decay", 1e-6, 1e-
2)

    step_size = trial.suggest_int("step_size", 2, 5)
    gamma = trial.suggest_uniform("gamma", 0.85, 0.99)

    model = models.efficientnet_b0(pretrained=True)
    model.classifier[1] = nn.Linear(model.classifier[1].in_features,
num_classes)
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=lr,
weight_decay=weight_decay)
    scheduler = optim.lr_scheduler.StepLR(optimizer,
step_size=step_size, gamma=gamma)

```

```

        return model, criterion, optimizer, scheduler

# Optuna training objective
best_model = None

def train_model_optuna(trial):
    global best_model
    model, criterion, optimizer, scheduler = define_model(trial)
    num_epochs = 2
    best_val_acc = 0.0

    for epoch in range(num_epochs):
        for phase in ['train', 'val']:
            model.train() if phase == 'train' else model.eval()
            running_loss, running_corrects = 0.0, 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    if isinstance(outputs, tuple) or hasattr(outputs,
"logits"):
                        main_outputs = outputs.logits if
hasattr(outputs, "logits") else outputs[0]
                        aux_outputs = outputs.aux_logits if
hasattr(outputs, "aux_logits") else None
                        loss = criterion(main_outputs, labels)
                        if aux_outputs is not None:
                            loss += 0.4 * criterion(aux_outputs,
labels)
                    else:
                        main_outputs = outputs
                        loss = criterion(main_outputs, labels)
                        _, preds = torch.max(main_outputs, 1)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            if phase == 'train':
                scheduler.step()

        epoch_acc = running_corrects.double() /

```

```

dataset_sizes[phase]

    if phase == 'val':
        if epoch_acc.item() > best_val_acc:
            best_val_acc = epoch_acc.item()
            best_model = model
        trial.report(best_val_acc, epoch)
        if trial.should_prune():
            raise optuna.exceptions.TrialPruned()

    return best_val_acc

```

Run Optuna study

```

study = optuna.create_study(direction="maximize")
study.optimize(train_model_optuna, n_trials=4)

```

```

final_model = best_model

```

```

[I 2026-07-07 08:37:02,510] A new study created in memory with name:
no-name-6df88dc0-6ab3-4fc7-969d-9c876b4399c9
/tmp/ipykernel_428/2657320632.py:3: FutureWarning: suggest_loguniform
has been deprecated in v3.0.0. This feature will be removed in v6.0.0.
See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float(..., log=True) instead.

```

```

    lr = trial.suggest_loguniform("lr", 1e-5, 1e-2)
/tmp/ipykernel_428/2657320632.py:4: FutureWarning: suggest_loguniform
has been deprecated in v3.0.0. This feature will be removed in v6.0.0.
See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float(..., log=True) instead.

```

```

    weight_decay = trial.suggest_loguniform("weight_decay", 1e-6, 1e-2)
/tmp/ipykernel_428/2657320632.py:6: FutureWarning: suggest_uniform has
been deprecated in v3.0.0. This feature will be removed in v6.0.0. See
https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float instead.

```

```

    gamma = trial.suggest_uniform("gamma", 0.85, 0.99)
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:2
08: UserWarning: The parameter 'pretrained' is deprecated since 0.13
and may be removed in the future, please use 'weights' instead.

```

```

    warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing
`weights=EfficientNet_B0_Weights.IMAGENET1K_V1`. You can also use
`weights=EfficientNet_B0_Weights.DEFAULT` to get the most up-to-date
weights.

```

```

    warnings.warn(msg)
[I 2026-07-07 08:39:32,959] Trial 0 finished with value:
0.923076923076923 and parameters: {'lr': 0.0001518200101355044,
'weight_decay': 0.00013680799491207568, 'step_size': 2, 'gamma':

```

```
0.9146186632024905}. Best is trial 0 with value: 0.923076923076923.
[I 2026-07-07 08:42:01,388] Trial 1 finished with value:
0.9260129183793305 and parameters: {'lr': 0.0005927433424636198,
'weight_decay': 6.380188026212757e-05, 'step_size': 3, 'gamma':
0.9187795526133273}. Best is trial 1 with value: 0.9260129183793305.
[I 2026-07-07 08:44:26,596] Trial 2 finished with value:
0.8960657662947739 and parameters: {'lr': 0.0019570021981886797,
'weight_decay': 3.29789960323497e-05, 'step_size': 4, 'gamma':
0.8545942752861322}. Best is trial 1 with value: 0.9260129183793305.
[I 2026-07-07 08:46:53,040] Trial 3 finished with value:
0.9330593071051086 and parameters: {'lr': 9.116982895648952e-05,
'weight_decay': 0.004878794242569644, 'step_size': 2, 'gamma':
0.898269477017864}. Best is trial 3 with value: 0.9330593071051086.
```

```
# Evaluate the best Optuna model
```

```
y_true, y_pred = evaluate_model_vit(final_model, test_loader)
```

```
print("Classification Report:")
```

```
print(classification_report(y_true, y_pred))
```

```
disp =
```

```
ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_true,
y_pred), display_labels=classes)
```

```
disp.plot(cmap="Blues", values_format='')
```

```
plt.title("Optuna-Tuned EfficientNet", fontsize=16)
```

```
plt.show()
```

```
100%|██████████| 107/107 [00:10<00:00, 10.01it/s]
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.95	0.94	0.94	526
1	0.99	0.99	0.99	550
2	0.94	0.86	0.90	592
3	0.90	0.93	0.91	608
4	0.94	0.97	0.96	558
5	0.93	0.96	0.95	577
accuracy			0.94	3411
macro avg	0.94	0.94	0.94	3411
weighted avg	0.94	0.94	0.94	3411

