

1 – Import Libraries

```
import os
import shutil
import zipfile
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
from PIL import Image
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.transforms as T
from torchvision import datasets, models
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")
```

#2 – Mount Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
import os

inner = '/content/drive/My Drive/INTEL_DATASET/'

print("Inside 'Intel dataset':")
for item in sorted(os.listdir(inner)):
    full = os.path.join(inner, item)
    if os.path.isdir(full):
        sub = sorted(os.listdir(full))
        print(f" {item}/ -> {sub[:8]}")
    else:
        print(f" {item} (file)")
```

```
Inside 'Intel dataset':
INTEL_PROCESSING.ipynb (file)
archive.zip (file)
```

3 Degin path and Unzip the dataset

```
import os

base_path = '/content/drive/My Drive/INTEL_DATASET'

for root, dirs, files in os.walk(base_path):
    level = root.replace(base_path, '').count(os.sep)
    if level > 3:
        continue
    indent = ' ' * 2 * level
    img_count = len([f for f in files if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp'))])
    print(f'{indent}{os.path.basename(root)}/ ({img_count} images)')
```

INTEL_DATASET/ (0 images)

import os, glob

```
# 1. Is Drive even mounted?
print("Drive mounted:", os.path.exists('/content/drive/My Drive'))

# 2. What's literally inside INTEL_DATASET, with exact names?
base = '/content/drive/My Drive/INTEL_DATASET'
print("\nExists:", os.path.exists(base))
if os.path.exists(base):
    print("Contents (exact names):")
    for item in os.listdir(base):
        print(" ", repr(item))

# 3. Find every zip anywhere under My Drive
print("\nAll zips under My Drive:")
for m in glob.glob('/content/drive/My Drive/**/*.zip', recursive=True):
    print(" ", repr(m))
```

Drive mounted: True

```
Exists: True
Contents (exact names):
'INTEL_PROCESSING.ipynb'
'archive.zip'
```

```
All zips under My Drive:
'/content/drive/My Drive/INTEL_DATASET/archive.zip'
'/content/drive/My Drive/Mobile Apps/QAWI/Prototype_1.zip'
'/content/drive/My Drive/Mobile Apps/QAWI/Prototype.zip'
'/content/drive/My Drive/MidOcean/Practical/PLANT_DISEASE/PlantVillage.zip'
```

import os, zipfile

```
zip_path = '/content/drive/My Drive/INTEL_DATASET/archive.zip'
extract_path = '/content/intel_extracted' # local Colab disk
```

```
os.makedirs(extract_path, exist_ok=True)
with zipfile.ZipFile(zip_path, 'r') as z:
    z.extractall(extract_path)
```

```
print('Extracted. Structure:')
for root, dirs, files in os.walk(extract_path):
    level = root.replace(extract_path, '').count(os.sep)
    if level > 3:
        continue
    indent = ' ' * 2 * level
```

```
img_count = len([f for f in files if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp'))])
print(f'{indent}{os.path.basename(root)}/{ (img_count) images}')
```

```
Extracted. Structure:
intel_extracted/ (0 images)
seg_test/ (0 images)
seg_train/ (0 images)
  buildings/ (437 images)
  glacier/ (553 images)
  sea/ (510 images)
  street/ (501 images)
  forest/ (474 images)
  mountain/ (525 images)
seg_train/ (0 images)
seg_train/ (0 images)
  buildings/ (2191 images)
  glacier/ (2404 images)
  sea/ (2274 images)
  street/ (2382 images)
  forest/ (2271 images)
  mountain/ (2512 images)
seg_pred/ (0 images)
seg_pred/ (7301 images)
```

#5 - Reduce Image Size

```
import os
from PIL import Image

dataset_root = '/content/intel_extracted/seg_train/seg_train'
output_root = '/content/intel_resized'
resize_size = (150, 150)
os.makedirs(output_root, exist_ok=True)

print('Source:', dataset_root)
for class_name in os.listdir(dataset_root):
    class_path = os.path.join(dataset_root, class_name)
    if not os.path.isdir(class_path):
        continue
    out_class = os.path.join(output_root, class_name)
    os.makedirs(out_class, exist_ok=True)
    count = 0
    for img_name in os.listdir(class_path):
        if img_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp')):
            try:
                with Image.open(os.path.join(class_path, img_name)) as img:
                    img = img.convert('RGB').resize(resize_size)
                    img.save(os.path.join(out_class, os.path.splitext(img_name)[0] + '.jpg'), 'JPEG')
                    count += 1
            except Exception as e:
                print(f'Failed: {img_name}: {e}')
    print(f' {class_name}: {count} resized')

print('Resizing complete')
classes = sorted(os.listdir(output_root))
print('Classes:', classes)
```

```
Source: /content/intel_extracted/seg_train/seg_train
buildings: 2191 resized
glacier: 2404 resized
sea: 2274 resized
street: 2382 resized
forest: 2271 resized
mountain: 2512 resized
Resizing complete
Classes: ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
```

#6 - split into 70/10/20 train/validation/test

```
import os, shutil
from sklearn.model_selection import train_test_split

source_root = '/content/intel_resized'
split_base = '/content/intel_split'

train_dir = os.path.join(split_base, 'train')
val_dir = os.path.join(split_base, 'validation')
test_dir = os.path.join(split_base, 'test')
for d in [train_dir, val_dir, test_dir]:
    os.makedirs(d, exist_ok=True)

train_ratio, val_ratio, test_ratio = 0.7, 0.1, 0.2

def split_and_copy(class_dir, dest_base):
    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_imgs, temp = train_test_split(images, test_size=(1 - train_ratio), random_state=42)
    val_imgs, test_imgs = train_test_split(temp, test_size=test_ratio/(val_ratio + test_ratio), random_state=42)
    for imgs, folder in zip([train_imgs, val_imgs, test_imgs], ['train', 'validation', 'test']):
        dest = os.path.join(dest_base, folder, os.path.basename(class_dir))
        os.makedirs(dest, exist_ok=True)
        for f in imgs:
            shutil.copy(f, dest)
    return len(train_imgs), len(val_imgs), len(test_imgs)

print('Splitting...')
for class_name in sorted(os.listdir(source_root)):
    full = os.path.join(source_root, class_name)
    if os.path.isdir(full):
        tr, va, te = split_and_copy(full, split_base)
        print(f' {class_name}: train={tr}, val={va}, test={te}')

print('Split complete')

# Verify totals
for split in ['train', 'validation', 'test']:
    total = sum(len(os.listdir(os.path.join(split_base, split, c)))
                 for c in os.listdir(os.path.join(split_base, split)))
    print(f'{split}: {total} images')
```

```

Splitting...
buildings: train=1533, val=219, test=439
forest: train=1589, val=227, test=455
glacier: train=1682, val=240, test=482
mountain: train=1758, val=251, test=503
sea: train=1591, val=227, test=456
street: train=1667, val=238, test=477
Split complete
train: 9820 images
validation: 1402 images
test: 2812 images

```

#7 - data loaders with augmentation

```

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

split_base = '/content/intel_split'

def get_data_loaders(data_dir, batch_size=32, train=False):
    if train:
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomRotation(15),
            T.Resize((224, 224)),
            T.ToTensor(),
            T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ])
        data = datasets.ImageFolder(os.path.join(data_dir, 'train'), transform=transform)
        return DataLoader(data, batch_size=batch_size, shuffle=True, num_workers=2), len(data)
    else:
        transform = T.Compose([
            T.Resize((224, 224)),
            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)

batch_size = 32
train_loader, train_len = get_data_loaders(split_base, batch_size, train=True)
val_loader, test_loader, val_len, test_len = get_data_loaders(split_base, batch_size, train=False)

classes = datasets.ImageFolder(os.path.join(split_base, 'train')).classes
num_classes = len(classes)
print('Classes:', classes)
print('num_classes:', num_classes)
print(f'Train: {train_len} | Val: {val_len} | Test: {test_len}')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Device:', device)

```

```

Classes: ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
num_classes: 6
Train: 9820 | Val: 1402 | Test: 2812
Device: cuda

```

#8 - Build the model

```

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

model = models.efficientnet_b0(weights='IMAGENET1K_V1')
model.classifier[1] = nn.Linear(model.classifier[1].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)

print('Model ready: EfficientNet-B0 (pretrained), output classes =', num_classes)

```

```

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, 201MB/s]
Model ready: EfficientNet-B0 (pretrained), output classes = 6

```

#9 - Training loop

```

from tqdm import tqdm

dataloaders = {"train": train_loader, "val": val_loader}
dataset_sizes = {"train": train_len, "val": val_len}

def train_model(model, criterion, optimizer, scheduler, num_epochs=5):
    for epoch in range(num_epochs):
        print(f"\nEpoch {epoch+1}/{num_epochs}\n" + "-"*10)
        for phase in ['train', 'val']:
            model.train() if phase == 'train' else model.eval()
            running_loss, running_corrects = 0.0, 0
            for inputs, labels in tqdm(dataloaders[phase], desc=phase):
                inputs, labels = inputs.to(device), labels.to(device)
                optimizer.zero_grad()
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    loss = criterion(outputs, labels)
                    _, preds = torch.max(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

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=5)

```

```

Epoch 1/5
-----
train: 100%|██████████| 307/307 [00:53<00:00, 5.76it/s]
train Loss: 0.4261 Acc: 0.8559
val: 100%|██████████| 44/44 [00:03<00:00, 12.35it/s]
val Loss: 0.3090 Acc: 0.8866

Epoch 2/5
-----
train: 100%|██████████| 307/307 [00:57<00:00, 5.37it/s]
train Loss: 0.2824 Acc: 0.9049
val: 100%|██████████| 44/44 [00:03<00:00, 12.63it/s]
val Loss: 0.3178 Acc: 0.8980

Epoch 3/5
-----
train: 100%|██████████| 307/307 [00:57<00:00, 5.37it/s]
train Loss: 0.2601 Acc: 0.9101
val: 100%|██████████| 44/44 [00:04<00:00, 9.81it/s]
val Loss: 0.3277 Acc: 0.8823

Epoch 4/5
-----
train: 100%|██████████| 307/307 [01:03<00:00, 4.87it/s]
train Loss: 0.2144 Acc: 0.9262
val: 100%|██████████| 44/44 [00:03<00:00, 13.81it/s]
val Loss: 0.2520 Acc: 0.9158

Epoch 5/5
-----
train: 100%|██████████| 307/307 [00:57<00:00, 5.33it/s]
train Loss: 0.1893 Acc: 0.9323
val: 100%|██████████| 44/44 [00:03<00:00, 13.17it/s]val Loss: 0.3369 Acc: 0.8951

```

#10 – evaluate on the test set

```

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report)

def evaluate_model(model_name, y_true, y_pred, classes):
    print(f"--- {model_name} ---")
    print("Accuracy :", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall   :", recall_score(y_true, y_pred, average='weighted'))
    print("F1-score :", f1_score(y_true, y_pred, average='weighted'))
    print("\nClassification Report:\n",
          classification_report(y_true, y_pred, target_names=classes))
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=classes, yticklabels=classes)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"Confusion Matrix – {model_name}")
    plt.show()

# Run predictions on the test set
y_true, y_pred = [], []
model.eval()
with torch.no_grad():
    for x, y in tqdm(test_loader, desc="testing"):
        out = model(x.to(device))
        preds = torch.argmax(torch.softmax(out, dim=1), dim=1).cpu().numpy()
        y_true.extend(y.numpy())
        y_pred.extend(preds)

evaluate_model("EfficientNet-B0", np.array(y_true), np.array(y_pred), classes)

```

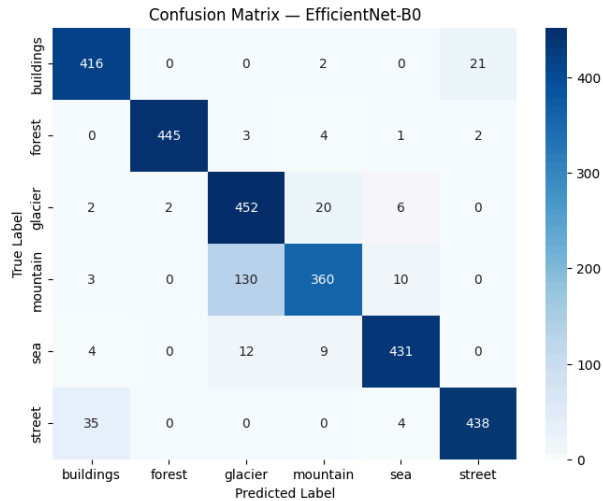
```

testing: 100%|██████████| 88/88 [00:10<00:00, 8.48it/s]
--- EfficientNet-B0 ---
Accuracy : 0.9039829302987198
Precision: 0.910864331133926
Recall   : 0.9039829302987198
F1-score : 0.9035281626048703

```

Classification Report:

	precision	recall	f1-score	support
buildings	0.90	0.95	0.93	439
forest	1.00	0.98	0.99	455
glacier	0.76	0.94	0.84	482
mountain	0.91	0.72	0.80	503
sea	0.95	0.95	0.95	456
street	0.95	0.92	0.93	477
accuracy			0.90	2812
macro avg	0.91	0.91	0.91	2812
weighted avg	0.91	0.90	0.90	2812



```

# Preview

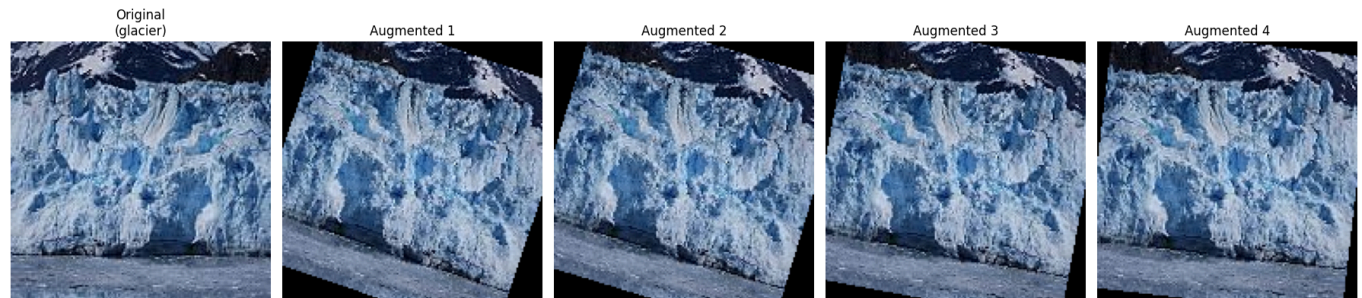
import matplotlib.pyplot as plt
import torchvision.transforms as T
from PIL import Image
import os, random

# Show ONE training image with several random augmentations applied
aug = T.Compose([
    T.RandomHorizontalFlip(p=1.0), # forced on for visible effect in preview
    T.RandomRotation(20),
])

# Grab a random training image
train_root = '/content/intel_split/train'
cls = random.choice(os.listdir(train_root))
cls_dir = os.path.join(train_root, cls)
img_path = os.path.join(cls_dir, random.choice(os.listdir(cls_dir)))
original = Image.open(img_path).convert('RGB')

fig, axes = plt.subplots(1, 5, figsize=(18, 4))
axes[0].imshow(original); axes[0].set_title(f'Original\n({cls})'); axes[0].axis('off')
for i in range(1, 5):
    augmented = aug(original)
    axes[i].imshow(augmented); axes[i].set_title(f'Augmented {i}'); axes[i].axis('off')
plt.tight_layout(); plt.show()

```



```

import torch, os

save_dir = '/content/drive/My Drive/INTEL_DATASET'
save_path = os.path.join(save_dir, 'efficientnet_intel.pth')

torch.save({
    'model_state_dict': model.state_dict(),
    'classes': classes,
    'num_classes': num_classes,
}, save_path)

```

```
print('Model saved to:', save_path)
print('Saved classes:', classes)

Model saved to: /content/drive/My Drive/INTEL_DATASET/efficientnet_intel.pth
Saved classes: ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
```

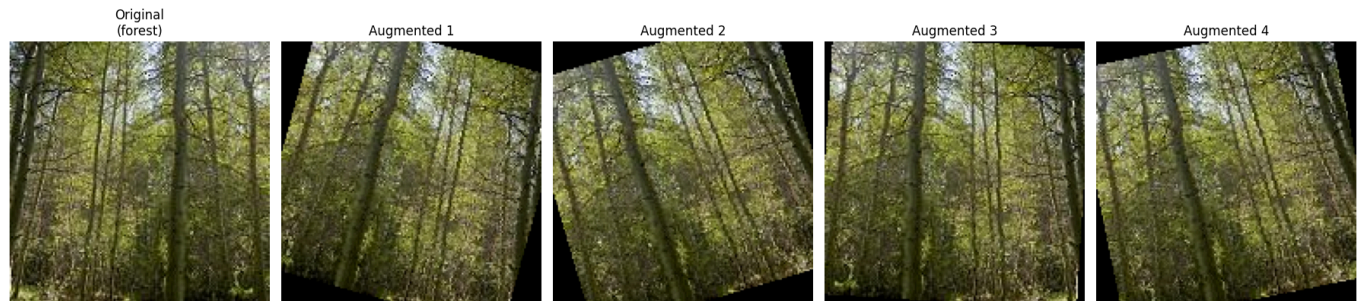
#Preview augmentation (shows flip + rotation on a sample image)

```
import matplotlib.pyplot as plt
import torchvision.transforms as T
from PIL import Image
import os, random

aug = T.Compose([
    T.RandomHorizontalFlip(p=1.0),
    T.RandomRotation(20),
])

train_root = '/content/intel_split/train'
cls = random.choice(os.listdir(train_root))
cls_dir = os.path.join(train_root, cls)
img_path = os.path.join(cls_dir, random.choice(os.listdir(cls_dir)))
original = Image.open(img_path).convert('RGB')

fig, axes = plt.subplots(1, 5, figsize=(18, 4))
axes[0].imshow(original); axes[0].set_title(f'Original\n({cls})'); axes[0].axis('off')
for i in range(1, 5):
    axes[i].imshow(aug(original)); axes[i].set_title(f'Augmented {i}'); axes[i].axis('off')
plt.tight_layout(); plt.show()
```



#Sample predictions (test images with predicted vs. true labels, green = correct, red = wrong):

```
import numpy as np, matplotlib.pyplot as plt, torch

mean = np.array([0.485, 0.456, 0.406]); std = np.array([0.229, 0.224, 0.225])

model.eval()
images, labels = next(iter(test_loader))
with torch.no_grad():
    preds = torch.argmax(model(images.to(device)), dim=1).cpu().numpy()

fig, axes = plt.subplots(3, 4, figsize=(14, 10))
for i, ax in enumerate(axes.flat):
    if i >= 12: break
    img = images[i].numpy().transpose(1, 2, 0)
    img = np.clip(std * img + mean, 0, 1)
    t, p = classes[labels[i]], classes[preds[i]]
    ax.imshow(img); ax.axis('off')
    ax.set_title(f"P: {p}\nT: {t}", color=('green' if t == p else 'red'), fontsize=11)
plt.tight_layout(); plt.show()
```

P: buildings
T: buildings



P: buildings
T: buildings



P: buildings
T: buildings



P: buildings
T: buildings



P: buildings

P: buildings

P: buildings

P: buildings

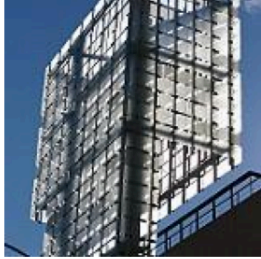
```
import numpy as np, matplotlib.pyplot as plt, torch, os, random
from PIL import Image
import torchvision.transforms as T

mean = np.array([0.485, 0.456, 0.406]); std = np.array([0.229, 0.224, 0.225])
tf = T.Compose([
    T.Resize((224, 224)), T.ToTensor(),
    T.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225])
])

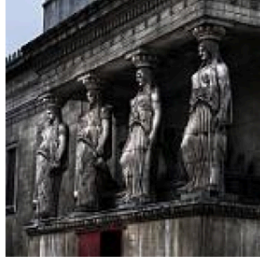
test_root = '/content/intel_split/test'
samples = []
for cls in classes:
    cls_dir = os.path.join(test_root, cls)
    for fname in random.sample(os.listdir(cls_dir), 2): # 2 per class = 12 total
        samples.append((os.path.join(cls_dir, fname), cls))

model.eval()
fig, axes = plt.subplots(3, 4, figsize=(14, 10))
for ax, (path, true_cls) in zip(axes.flat, samples):
    img = Image.open(path).convert('RGB')
    x = tf(img).unsqueeze(0).to(device)
    with torch.no_grad():
        pred_cls = classes[torch.argmax(model(x), dim=1).item()]
    ax.imshow(img); ax.axis('off')
    ax.set_title(f"P: {pred_cls}\nT: {true_cls}",
        color='green' if pred_cls == true_cls else 'red'), fontsize=11)
plt.tight_layout(); plt.show()
```

P: buildings
T: buildings



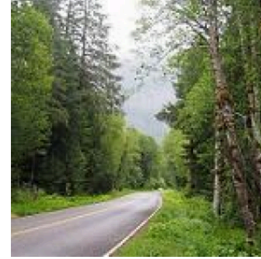
P: buildings
T: buildings



P: forest
T: forest



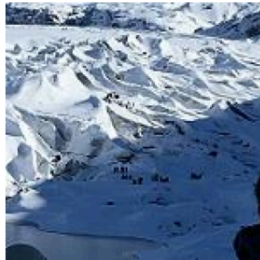
P: forest
T: forest



P: glacier
T: glacier



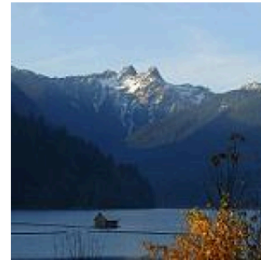
P: glacier
T: glacier



P: sea
T: mountain



P: glacier
T: mountain



P: sea
T: sea

P: sea
T: sea

P: street
T: street

P: street
T: street