Intel Classification using the resnet architecture

Pegah Givehchian - 99222089 - Seri 3 - Computational neural networks

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import OrderedDict
import cv2
import os
from PIL import Image
import torch
from torch import optim
from torch.autograd import Variable
from torch.utils.data import random_split, DataLoader
from torch import nn
import torch.nn.functional as F
from torchvision.utils import make_grid
from torchvision import transforms, models, datasets
from tqdm import trange, tqdm
```

Define the paths

In [2]:
```
train_dir = "../input/intel-image-classification/seg_train/seg_train/"
test_dir = "../input/intel-image-classification/seg_test/seg_test/"
pred_dir ="../input/intel-image-classification/seg_pred/seg_pred/"


pred_files = [os.path.join(pred_dir, f) for f in os.listdir(pred_dir)]
```

Define our classes

In [3]:
```
class_arr = os.listdir(train_dir)
print(enumerate([0,1]))
classes = {0: 'buildings',
         1: 'forest',
         2: 'glacier',
         3: 'mountain',
         4: 'sea',
         5: 'street'}
classes
```

```
<enumerate object at 0x7f99e52d2dc0>
```

Out[3]:
```
{0: 'buildings',
 1: 'forest',
 2: 'glacier',
 3: 'mountain',
 4: 'sea',
 5: 'street'}
```

Get the count of each category in our training and testing data:

In [4]:
```
def get_cat_count(dir):
    cat_count = {}
    for cat in class_arr:
        count = len(os.listdir(os.path.join(dir, cat)))
        cat_count[cat] =count
    return(cat_count)
```

In [5]:
```
get_cat_count(train_dir)
```

Out[5]:
```
{'mountain': 2512,
 'street': 2382,
 'buildings': 2191,
 'sea': 2274,
 'forest': 2271,
 'glacier': 2404}
```

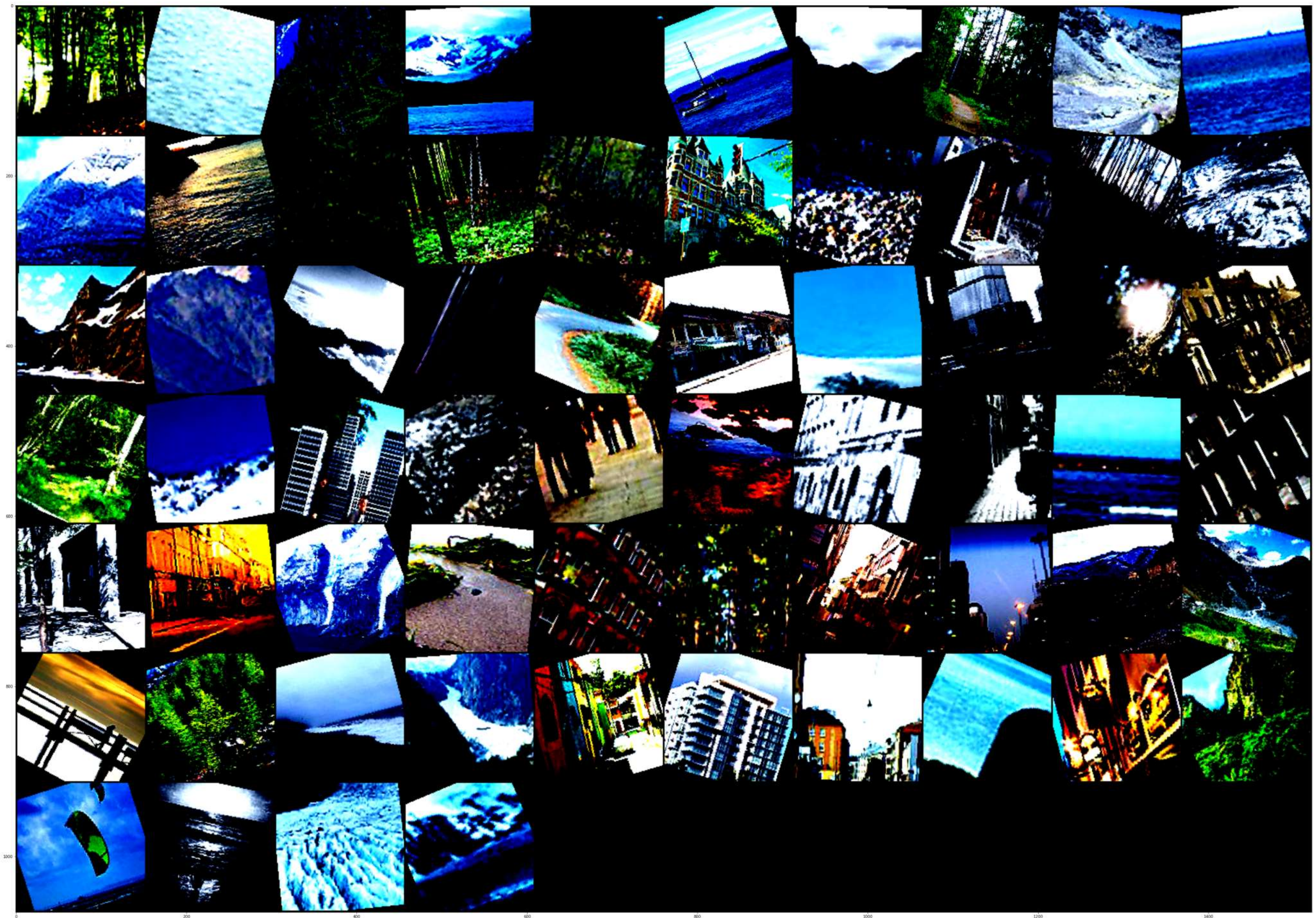
In [6]:
```
get_cat_count(test_dir)
```

Out[6]:
```
{'mountain': 525,
 'street': 501,
 'buildings': 437,
 'sea': 510,
 'forest': 474,
 'glacier': 553}
```

Transform data

In [7]:
```
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]

train_transform = transforms.Compose([transforms.Resize((150, 150)),
                                       transforms.RandomResizedCrop(150),
                                       transforms.RandomRotation(30),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       transforms.Normalize(torch.Tensor(mean), torch.Tensor(std)) # Normalize
                                       ])



test_transform = transforms.Compose([transforms.Resize((150, 150)),
                                      transforms.CenterCrop(150),
                                      transforms.ToTensor(),
                                      transforms.Normalize(torch.Tensor(mean),torch.Tensor(std))
                                      ])

tmp_ds = datasets.ImageFolder(train_dir, transform=train_transform)

train_ds, val_ds = random_split(tmp_ds, [10000, 4034],
                                 generator=torch.Generator().manual_seed(42))
test_ds = datasets.ImageFolder(test_dir, transform=test_transform)

train_loader = torch.utils.data.DataLoader(train_ds, batch_size=64, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_ds, batch_size=64)
test_loader = torch.utils.data.DataLoader(test_ds, batch_size=64)
```

In [8]:
```
def show_batch(loader):
    plt.figure(figsize=(60,60))
    batch = next(iter(loader))
    images, labels = batch
    grid = make_grid(images, nrow = 10)
    plt.imshow(np.transpose(grid, (1 ,2, 0)))
    plt.show()


show_batch(train_loader)
```



Choose Device

In [9]:
```
device =  torch.device('cuda' if torch.cuda.is_available else 'cpu')
device
```

Out[9]:
```
device(type='cuda')
```

We will be using the pretrained resnet50 model.

In [10]:
```
resnet = models.resnet50(pretrained=True)
# Freeze model params
for param in resnet.parameters():
    param.required_grad = False
# Pull final fc layer feature dimensions
in_features = resnet.fc.in_features
out_features = resnet.fc.out_features
print(f"model in features: {in_features}")
print(f"model out features: {out_features}")

#We have to transform the 1000 outputs to 6 for our classification
classifier = nn.Sequential(OrderedDict([('fc1', nn.Linear(in_features, 512)),
                                        ('relu', nn.ReLU()),
                                        ('drop', nn.Dropout(0.05)),
                                        ('fc2', nn.Linear(512, 6)),
                                        ]))

# Now we add our classifier layer to our resnet model and then push it to our device
resnet.classifier = classifier
resnet.to(device)
```

```
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676
ba61.pth

  0%|          | 0.00/97.8M [00:00<?, ?B/s]

model in features: 2048
model out features: 1000
```

Out[10]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

Define our loss function, optimizer and scheduler (we use the multistep learning rate)

In [11]:
```
criterion = nn.CrossEntropyLoss()
#initialize our learning rate to 0.001
optimizer = torch.optim.SGD(resnet.parameters(), lr=0.001)
scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[10, 15], gamma=0.05)
```

In [12]:

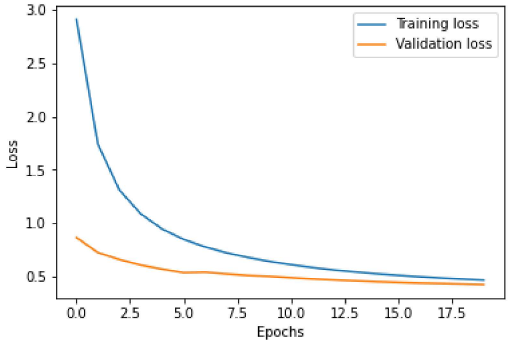
```
epochs = 20

tr_losses = []
avg_epoch_tr_loss = []
tr_accuracy = []

val_losses = []
avg_epoch_val_loss = []
val_accuracy = []
val_loss_min = np.Inf

resnet.train()
for epoch in range(epochs):
  for i, batch in enumerate(train_loader):
    data, label = batch
    data, label = data.to(device), label.to(device)
    optimizer.zero_grad()
    logit = resnet(data)
    loss = criterion(logit, label)
    loss.backward()
    optimizer.step()
    tr_losses.append(loss.item())
    tr_accuracy.append(label.eq(logit.argmax(dim=1)).float().mean())
  print(f'\nEpoch No: {epoch + 1},Training Loss: {torch.tensor(tr_losses).mean():.2f},Training Accuracy: {torch.tensor(tr_ac
  avg_epoch_tr_loss.append(torch.tensor(tr_losses).mean())


  resnet.eval()
  for i, batch in enumerate(val_loader):
    data, label = batch
    data, label = data.to(device), label.to(device)
    with torch.no_grad():
      logit = resnet(data)
    loss = criterion(logit, label)
    val_losses.append(loss.item())
    val_accuracy.append(label.eq(logit.argmax(dim=1)).float().mean())

  print(f'\nEpoch No: {epoch + 1}, Val Loss: {torch.tensor(val_losses).mean():.2f}, Val Accuracy: {torch.tensor(val_accuracy
  avg_epoch_val_loss.append(torch.tensor(val_losses).mean())

  if torch.tensor(val_losses).float().mean() <= val_loss_min:
    torch.save(resnet.state_dict(), './model_state.pt')
    val_loss_min = torch.tensor(val_losses).mean()
  scheduler.step()
```

```
Epoch No: 1,Training Loss: 2.91,Training Accuracy: 0.50

Epoch No: 1, Val Loss: 0.87, Val Accuracy: 0.77

Epoch No: 2,Training Loss: 1.74,Training Accuracy: 0.65

Epoch No: 2, Val Loss: 0.73, Val Accuracy: 0.79

Epoch No: 3,Training Loss: 1.31,Training Accuracy: 0.72

Epoch No: 3, Val Loss: 0.66, Val Accuracy: 0.80

Epoch No: 4,Training Loss: 1.09,Training Accuracy: 0.75

Epoch No: 4, Val Loss: 0.61, Val Accuracy: 0.81

Epoch No: 5,Training Loss: 0.95,Training Accuracy: 0.77

Epoch No: 5, Val Loss: 0.57, Val Accuracy: 0.81

Epoch No: 6,Training Loss: 0.85,Training Accuracy: 0.79

Epoch No: 6, Val Loss: 0.54, Val Accuracy: 0.82

Epoch No: 7,Training Loss: 0.78,Training Accuracy: 0.80

Epoch No: 7, Val Loss: 0.54, Val Accuracy: 0.82

Epoch No: 8,Training Loss: 0.72,Training Accuracy: 0.81

Epoch No: 8, Val Loss: 0.53, Val Accuracy: 0.82

Epoch No: 9,Training Loss: 0.68,Training Accuracy: 0.82

Epoch No: 9, Val Loss: 0.51, Val Accuracy: 0.83

Epoch No: 10,Training Loss: 0.64,Training Accuracy: 0.82

Epoch No: 10, Val Loss: 0.50, Val Accuracy: 0.83

Epoch No: 11,Training Loss: 0.61,Training Accuracy: 0.83

Epoch No: 11, Val Loss: 0.49, Val Accuracy: 0.83

Epoch No: 12,Training Loss: 0.59,Training Accuracy: 0.83

Epoch No: 12, Val Loss: 0.48, Val Accuracy: 0.84

Epoch No: 13,Training Loss: 0.56,Training Accuracy: 0.84

Epoch No: 13, Val Loss: 0.47, Val Accuracy: 0.84

Epoch No: 14,Training Loss: 0.54,Training Accuracy: 0.84

Epoch No: 14, Val Loss: 0.46, Val Accuracy: 0.84

Epoch No: 15,Training Loss: 0.53,Training Accuracy: 0.85

Epoch No: 15, Val Loss: 0.45, Val Accuracy: 0.84

Epoch No: 16,Training Loss: 0.51,Training Accuracy: 0.85

Epoch No: 16, Val Loss: 0.45, Val Accuracy: 0.85

Epoch No: 17,Training Loss: 0.50,Training Accuracy: 0.85

Epoch No: 17, Val Loss: 0.44, Val Accuracy: 0.85

Epoch No: 18,Training Loss: 0.49,Training Accuracy: 0.85

Epoch No: 18, Val Loss: 0.44, Val Accuracy: 0.85

Epoch No: 19,Training Loss: 0.48,Training Accuracy: 0.86

Epoch No: 19, Val Loss: 0.43, Val Accuracy: 0.85

Epoch No: 20,Training Loss: 0.47,Training Accuracy: 0.86

Epoch No: 20, Val Loss: 0.43, Val Accuracy: 0.85
```

We see that we have reached a decent accuracy, and the val accuracy and training accuracy are really close, which means that the model isn't overfitting.

Plot of the average loss in each epoch

In [18]:
```
plt.plot(avg_epoch_tr_loss, label='Training loss')
plt.plot(avg_epoch_val_loss, label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

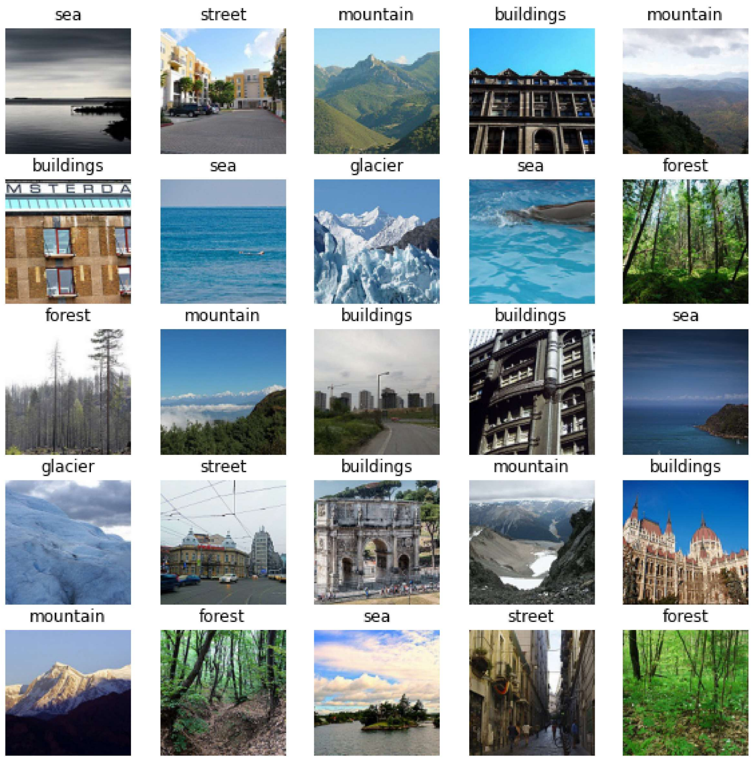


Now we make some predicitons to test our model.

In [33]:
```
resnet.load_state_dict(torch.load('./model_state.pt'))
resnet.eval()
#push our model to cuda
resnet = resnet.cuda()

def predictions(image):
    transform = test_transform(image)
    img = transform.unsqueeze(0).cuda()
    # push to gpu
    gpu_img = img.to(device)
    output = resnet(gpu_img)
    index = output.data.cpu().numpy().argmax()
    return index
```

In [36]:

```
resnet.eval()
plt.figure(figsize=(10,10))
for i, images in enumerate(pred_files):
    if i > 24:
      break
    img = Image.open(images)
    index = predictions(img)
    plt.subplot(5,5,i+1)
    plt.title(classes[index])
    plt.axis('off')
    plt.imshow(img)
```



We see that all the predictions are correct and the model has a decent accuracy.

In [ ]: