In [15]:
```
# Import the necessary libraries
import os
import torch
import torchvision
import torchvision.transforms as transforms
import pandas as pd
from PIL import Image
from torch.utils.data import Dataset, DataLoader
import csv
import glob
```

Create features

In [16]:
```
# Set the device to use for PyTorch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Dataloader

In [17]:
```
# Define a dataset that loads the images from a folder
class FolderDataset(Dataset):
    def __init__(self, folder_path, transform=None):
        self.folder_path = folder_path
        self.transform = transform
        self.files = os.listdir(folder_path)

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        img_path = os.path.join(self.folder_path, self.files[idx])
        image = Image.open(img_path)
        # Convert the image to a PyTorch tensor
        image = transforms.ToTensor()(image)
        # resize the image to 224x224
        image = transforms.Resize((224, 224))(image)
        if self.transform:
            image = self.transform(image)
        return image
```

In [18]:
```
# Create a dataset that loads the images from the "images" folder
image_dir = "static/data/jpg"
image_list = os.listdir(image_dir)
dataset = FolderDataset(image_dir)

# Create a dataloader for the dataset
dataloader = DataLoader(
    dataset, batch_size=1, shuffle=False, num_workers=10
)
```

Load model

In [19]:
```
# Load the pre-trained densenet model
model = torchvision.models.densenet121(pretrained=True)

# Set the model to evaluation mode
model.eval()
```

```
# Move the model to the specified device
model = model.to(device)
```

```
/home/temsfrog/anaconda3/envs/pfee-smith/lib/python3.10/site-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(
/home/temsfrog/anaconda3/envs/pfee-smith/lib/python3.10/site-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=DenseNet121_Weights.IMAGENET1K_V1`. You can also use `weights=DenseNet121_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

Csv to save features

In [20]:
```
header = ['image_id', "features"]
# create csv file
f_features = open('data/features_densenet_test.csv', 'w')
# initialize writer for csv
writer_features = csv.writer(f_features)
# write header
writer_features.writerow(header)
```

Out[20]:
```
19
```

Features extraction from images

In [21]:
```
import tqdm
# Extract features from the images in the dataset
for i, inputs in enumerate(tqdm.tqdm(dataloader)):
    # Move the input images to the specified device
    inputs = inputs.to(device)

    # Extract the features from the intermediate layer of the VGG19 model
    features = model.features(inputs)

    # Convert the features to a NumPy array
    features = features.detach().cpu().numpy()
    # Reshape the features to a 1D array
    features = features.reshape(features.shape[0], -1)
    # to string
    features = features[0].tolist()

    # write to csv
    writer_features.writerow([image_list[i], features])
```

```
  4%|▌          | 56/1491 [00:06<02:17, 10.46it/s]
```

In [ ]:
```
# Close the file
f_features.close()
```

Dimension reduction

In [ ]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:
```
# Load the features from the CSV file into a Pandas DataFrame
df = pd.read_csv("data/features_densenet_test.csv")
```

In [ ]:
```
# Load the dataset of images and obtain their features using the model
image_features = []
for i in range(len(df)):
    features = df.iloc[i, 1]
    features = np.array([float(x) for x in features[1:-1].split(",")])
    image_features.append(features)

# Perform dimensionality reduction on the list of image features using PCA
pca = PCA(n_components=128)
reduced_features = pca.fit_transform(image_features)
```

Test on image query

In [ ]:
```
# Define the query image and its features
def get_image_features(image):
  image = Image.open("data/images/" + image)

  # Convert the image to a PyTorch tensor
  image = transforms.ToTensor()(image)

  # resize the image to 224x224
  image = transforms.Resize((224, 224))(image)

  # Add a batch dimension to the image
  image = image.unsqueeze(0)

  # Move the image to the specified device
  image = image.to(device)

  # Obtain its features using the model
  query_features = model.features(image)

  # Convert the features to a NumPy array
  query_features = query_features.detach().cpu().numpy()

  return query_features[0]

def get_closest_images(query_image, nb_closest=50):
  query_features = get_image_features(query_image)
  # reshape to 2 dim
  query_features = query_features.reshape(1, -1)
  query_features_reduced = pca.transform(query_features)

  # Compare the query features to the reduced features and return the most similar
  similarity_scores = []
  for features in reduced_features:
    similarity = torch.nn.functional.cosine_similarity(torch.Tensor(query_features_
    similarity_scores.append(similarity)

  # Sort the similarity scores in descending order
  similarity_scores = np.array(similarity_scores)
  sorted_indices = np.argsort(similarity_scores)[::-1]

  # nb_closest similarity scores and convert tensor to int
  similarity_scores = similarity_scores[sorted_indices][:nb_closest]
  similarity_scores = [float(x) for x in similarity_scores]
```

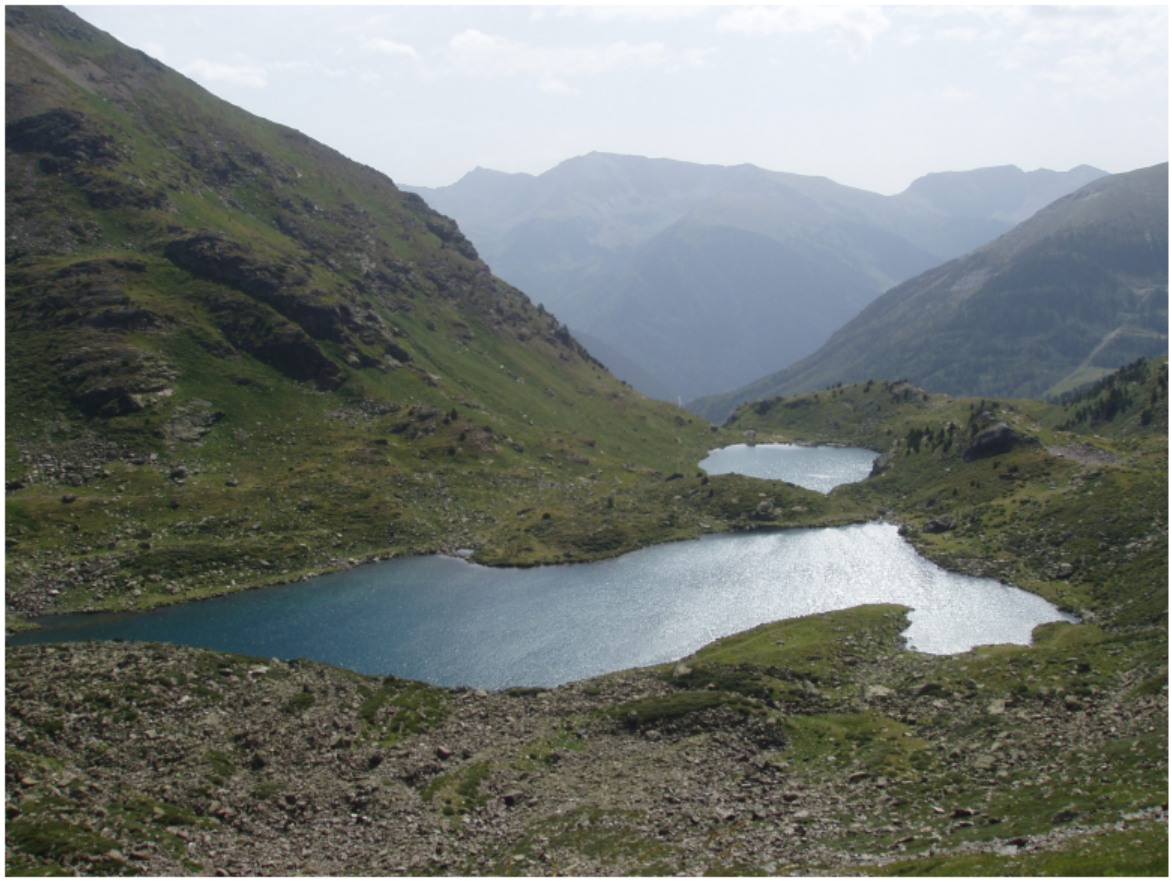
```
  # Get the top nb_closest most similar images
  most_similar_images = []
  for i in range(nb_closest):
    image_name = df.iloc[sorted_indices[i], 0]
    most_similar_images.append(image_name)

  return most_similar_images, similarity_scores

query_image = "111400.jpg"
most_similar_images, similarity_scores = get_closest_images(query_image, nb_closes
```

```
/tmp/ipykernel_16024/3040246366.py:38: FutureWarning: The input object of type 'Te
nsor' is an array-like implementing one of the corresponding protocols (`__array__
`, `__array_interface__` or `__array_struct__`); but not a sequence (or 0-D). In t
he future, this object will be coerced as if it was first converted using `np.arra
y(obj)`. To retain the old behaviour, you have to either modify the type 'Tensor',
or assign to an empty array created with `np.empty(correct_shape, dtype=object)`.
  similarity_scores = np.array(similarity_scores)
/tmp/ipykernel_16024/3040246366.py:38: VisibleDeprecationWarning: Creating an ndar
ray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or n
darrays with different lengths or shapes) is deprecated. If you meant to do this,
you must specify 'dtype=object' when creating the ndarray.
  similarity_scores = np.array(similarity_scores)
```

Plot input image

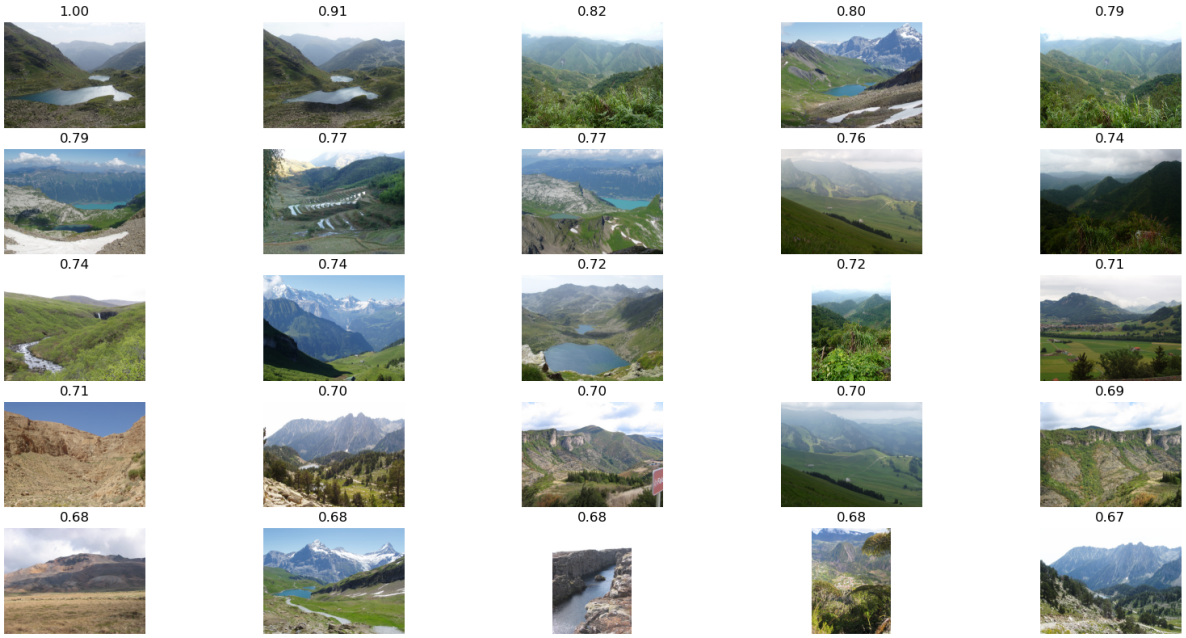
In [ ]:

```
# Display the query image with matplotlib
plt.figure(figsize=(10, 10))
plt.imshow(plt.imread("data/images/" + query_image))
plt.axis("off")
plt.show()
```



Plot first 50 most similar images with their similarity scores

In [ ]:
```
# Display the top 100 most similar images with matplotlib
plt.figure(figsize=(20, 20))
for i in range(25):
  plt.subplot(10, 5, i + 1)
  plt.imshow(plt.imread("data/images/" + most_similar_images[i]))
  # convert to int
  plt.title(f"{similarity_scores[i]:.2f}")
  plt.axis("off")

  ## first image is the query image so obviously it is the most similar
```



In [ ]: