```python
# Standard libraries
import os
import random
from tqdm.notebook import tqdm

# Data manipulation and visualization
import matplotlib.pyplot as plt
from PIL import Image
import seaborn as sns
import pandas as pd
import numpy as np

# Deep Learning libraries
import torch
import torchvision
import torchsummary
from torch.utils import data
from torchvision import datasets, models, transforms

# Set seed for reproducibility
SEED = 42
np.random.seed(SEED)

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

from torch.utils.data import DataLoader, Dataset
from glob import glob
from itertools import chain
import random

class EuroSATDataset(Dataset):

    def __init__(self, root, train_flag = True):

        self.train_flag = train_flag
        self.root = root
        self.images_paths = [glob(f'{root}/{folder}/*.jpg') for folder
in os.listdir(f"{root}")]
        self.images_paths =
list(chain.from_iterable(self.images_paths))
        random.shuffle(self.images_paths)

        # комментарий преподавателя: на текущей машине очень мало
мощностей, максимально обрезаем данные
        # для обучения и валидации в соотношении 100/30

        self.count = {True: 100}
        self.count[False] = 30
```

```
        self.classes_names = {class_name:label for label, class_name
in enumerate(os.listdir(f"{root}"))}
        self.labels =
[self.classes_names[os.path.basename(os.path.dirname(path))] for path
in self.images_paths]

        # применяем аугментациб данных. В качестве ДЗ расширите список
применяемых преобразований.
        self.transform_train = transforms.Compose([
            transforms.RandomResizedCrop((224, 224)),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(30),
            transforms.ColorJitter(brightness=0.2, contrast=0.2),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
                                                  ])
        self.transform_test =  transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
        ])


    def __len__(self):
        return self.count[self.train_flag]

    def __getitem__(self, index):
        if (self.train_flag):
            index = index % self.count[True]
        else:
            index = self.count[True] + index % self.count[False]
        image_path = self.images_paths[index]
        image = Image.open(image_path).convert('RGB')
        label = self.labels[index]

        if self.train_flag:
            image = self.transform_train(image)
        else:
            image = self.transform_test(image)

        return image.float().to(device),
torch.tensor([label]).float().to(device)

data =  EuroSATDataset('./EuroSAT/2750', train_flag=True)
len(data)

import random
```

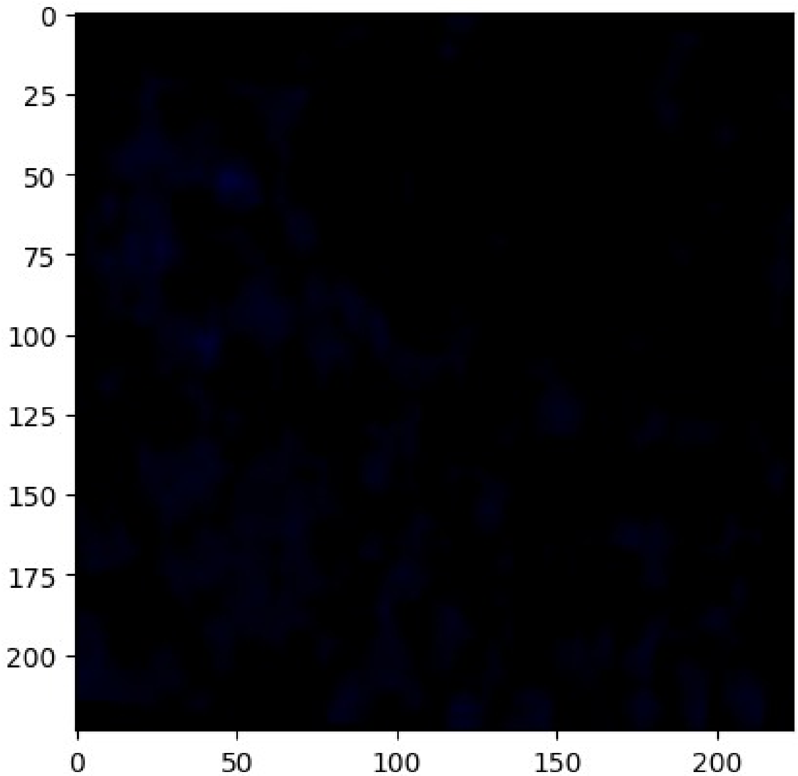
```python
import cv2

image, label = data[random.randint(0, len(data))]
print(f"Image Size: {image.shape[2]} x {image.shape[1]} x
{image.shape[0]}")
print(f"Label: {label}")
print([key  for key, value in data.classes_names.items() if value ==
label][0])
plt.imshow(image.permute(1,2,0).cpu().numpy())
plt.show()

train_dataset = EuroSATDataset('./EuroSAT/2750')
test_dataset = EuroSATDataset('./EuroSAT/2750',train_flag = False)
train_dataset_loader = DataLoader(train_dataset, batch_size=32,
shuffle=True, drop_last=True)
test_dataset_loader = DataLoader(test_dataset, batch_size=1,
shuffle=True, drop_last=True)

# fine turning MODEL
# change last model layer (model.fc) for classification on EuroSAT set
of classes, 10 classes
# (not 1000 like for ImageNet dataset pretrained weights)
# pretrained models https://pytorch.org/vision/stable/models.html

model = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
model.fc = torch.nn.Linear(model.fc.in_features,10)
model = model.to(device)
torchsummary.summary(model, (3, 224, 224))

# Specify number of epochs and learning rate
lr = 1e-3

# Specify criterion and optimizer
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

def train_step(model, loss_function, optimizer, image, label):
    model.train()
    optimizer.zero_grad()
    prediction = model(image)
    #print(prediction)
    #print(label)
    loss = loss_function(prediction.squeeze(), label.long().squeeze())
    loss.backward()
    optimizer.step()
    return loss.item()

@torch.no_grad()
def accuracy(model, loss_function, image, label):
    model.eval()
```

```
    prediction = model(image)
    max_values, argmaxes = prediction.max(-1)
    is_correct = argmaxes == label.int().squeeze()
    return is_correct.cpu().numpy().tolist()

@torch.no_grad()
def validation_loss(model, loss_function, image, label):
    model.eval()
    prediction = model(image)
    loss = loss_function(prediction.squeeze(), label.long().squeeze())
    return loss.item()

train_losses = []
train_accuracies = []
test_losses = []
test_accuracies = []

n_epochs = 10
for epoch in range(n_epochs):
    print(f"Epoch: {epoch+1}")
    train_epoch_losses = []
    train_epoch_accuracies = []
    for idx,(image, label) in enumerate(train_dataset_loader):
        loss = train_step(model, criterion, optimizer, image, label)
        train_epoch_losses.append(loss)
        print(loss)
        #if (idx + 1) %100 == 0: print(loss)
    train_epoch_loss = np.mean(train_epoch_losses)
    print(f"Train Loss: {train_epoch_loss:.4f}")
    train_losses.append(train_epoch_loss)

    for idx,(image, label) in enumerate(train_dataset_loader):
        is_correct = accuracy(model, criterion, image, label)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    print(f"Train Accuracy: {train_epoch_accuracy*100:.2f}%")
    train_accuracies.append(train_epoch_accuracy)

    test_epoch_losses = []
    test_epoch_accuracies = []
    for idx,(image, label) in enumerate(test_dataset_loader):
        #print(label)
        loss = validation_loss(model, criterion, image, label)
        test_epoch_losses.append(loss)
        is_correct = accuracy(model, criterion, image, label)
        test_epoch_accuracies.extend(is_correct)
    test_epoch_loss = np.mean(test_epoch_losses)
    print(f"Test Loss: {test_epoch_loss:.4f}")
    test_losses.append(test_epoch_loss)
    test_epcoh_accuracy = np.mean(test_epoch_accuracies)
```

```
    print(f"Test Accuracy: {test_epcoh_accuracy*100:.2f}%")
    test_accuracies.append(test_epcoh_accuracy)

# example of model saving
model_dir = "./models/"
if not os.path.exists(model_dir):
  os.makedirs(model_dir)

model_file = os.path.join(model_dir, 'best_model.pth')
model_file
torch.save(model.state_dict(), model_file)

# EXAMPLE OF MODEL LOADING
model = models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
model.fc = torch.nn.Linear(model.fc.in_features,10)
model = model.to(device)

# example of loading model from the file
model.load_state_dict(torch.load(model_file, weights_only=True))
```

```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers). Got range [-
2.117904..0.23477131].
```

```
Image Size: 224 x 224 x 3
Label: tensor([2.])
HerbaceousVegetation
```




```
Downloading: "https://download.pytorch.org/models/resnet50-
11ad3fa6.pth" to C:\Users\Даниил/.cache\torch\hub\checkpoints\
resnet50-11ad3fa6.pth
100%|
| 97.8M/97.8M [00:42<00:00, 2.43MB/s]

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 112, 112]           9,408
       BatchNorm2d-2         [-1, 64, 112, 112]             128
              ReLU-3         [-1, 64, 112, 112]               0
         MaxPool2d-4           [-1, 64, 56, 56]               0
            Conv2d-5           [-1, 64, 56, 56]           4,096
       BatchNorm2d-6           [-1, 64, 56, 56]             128
              ReLU-7           [-1, 64, 56, 56]               0
            Conv2d-8           [-1, 64, 56, 56]          36,864
       BatchNorm2d-9           [-1, 64, 56, 56]             128
             ReLU-10           [-1, 64, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]          16,384
      BatchNorm2d-12          [-1, 256, 56, 56]             512
           Conv2d-13          [-1, 256, 56, 56]          16,384
      BatchNorm2d-14          [-1, 256, 56, 56]             512
             ReLU-15          [-1, 256, 56, 56]               0
```

Bottleneck-16	[-1, 256, 56, 56]	0
Conv2d-17	[-1, 64, 56, 56]	16,384
BatchNorm2d-18	[-1, 64, 56, 56]	128
ReLU-19	[-1, 64, 56, 56]	0
Conv2d-20	[-1, 64, 56, 56]	36,864
BatchNorm2d-21	[-1, 64, 56, 56]	128
ReLU-22	[-1, 64, 56, 56]	0
Conv2d-23	[-1, 256, 56, 56]	16,384
BatchNorm2d-24	[-1, 256, 56, 56]	512
ReLU-25	[-1, 256, 56, 56]	0
Bottleneck-26	[-1, 256, 56, 56]	0
Conv2d-27	[-1, 64, 56, 56]	16,384
BatchNorm2d-28	[-1, 64, 56, 56]	128
ReLU-29	[-1, 64, 56, 56]	0
Conv2d-30	[-1, 64, 56, 56]	36,864
BatchNorm2d-31	[-1, 64, 56, 56]	128
ReLU-32	[-1, 64, 56, 56]	0
Conv2d-33	[-1, 256, 56, 56]	16,384
BatchNorm2d-34	[-1, 256, 56, 56]	512
ReLU-35	[-1, 256, 56, 56]	0
Bottleneck-36	[-1, 256, 56, 56]	0
Conv2d-37	[-1, 128, 56, 56]	32,768
BatchNorm2d-38	[-1, 128, 56, 56]	256
ReLU-39	[-1, 128, 56, 56]	0
Conv2d-40	[-1, 128, 28, 28]	147,456
BatchNorm2d-41	[-1, 128, 28, 28]	256
ReLU-42	[-1, 128, 28, 28]	0
Conv2d-43	[-1, 512, 28, 28]	65,536
BatchNorm2d-44	[-1, 512, 28, 28]	1,024
Conv2d-45	[-1, 512, 28, 28]	131,072
BatchNorm2d-46	[-1, 512, 28, 28]	1,024
ReLU-47	[-1, 512, 28, 28]	0
Bottleneck-48	[-1, 512, 28, 28]	0
Conv2d-49	[-1, 128, 28, 28]	65,536
BatchNorm2d-50	[-1, 128, 28, 28]	256
ReLU-51	[-1, 128, 28, 28]	0
Conv2d-52	[-1, 128, 28, 28]	147,456
BatchNorm2d-53	[-1, 128, 28, 28]	256
ReLU-54	[-1, 128, 28, 28]	0
Conv2d-55	[-1, 512, 28, 28]	65,536
BatchNorm2d-56	[-1, 512, 28, 28]	1,024
ReLU-57	[-1, 512, 28, 28]	0
Bottleneck-58	[-1, 512, 28, 28]	0
Conv2d-59	[-1, 128, 28, 28]	65,536
BatchNorm2d-60	[-1, 128, 28, 28]	256
ReLU-61	[-1, 128, 28, 28]	0
Conv2d-62	[-1, 128, 28, 28]	147,456
BatchNorm2d-63	[-1, 128, 28, 28]	256
ReLU-64	[-1, 128, 28, 28]	0

Conv2d-65	[-1, 512, 28, 28]	65,536
BatchNorm2d-66	[-1, 512, 28, 28]	1,024
ReLU-67	[-1, 512, 28, 28]	0
Bottleneck-68	[-1, 512, 28, 28]	0
Conv2d-69	[-1, 128, 28, 28]	65,536
BatchNorm2d-70	[-1, 128, 28, 28]	256
ReLU-71	[-1, 128, 28, 28]	0
Conv2d-72	[-1, 128, 28, 28]	147,456
BatchNorm2d-73	[-1, 128, 28, 28]	256
ReLU-74	[-1, 128, 28, 28]	0
Conv2d-75	[-1, 512, 28, 28]	65,536
BatchNorm2d-76	[-1, 512, 28, 28]	1,024
ReLU-77	[-1, 512, 28, 28]	0
Bottleneck-78	[-1, 512, 28, 28]	0
Conv2d-79	[-1, 256, 28, 28]	131,072
BatchNorm2d-80	[-1, 256, 28, 28]	512
ReLU-81	[-1, 256, 28, 28]	0
Conv2d-82	[-1, 256, 14, 14]	589,824
BatchNorm2d-83	[-1, 256, 14, 14]	512
ReLU-84	[-1, 256, 14, 14]	0
Conv2d-85	[-1, 1024, 14, 14]	262,144
BatchNorm2d-86	[-1, 1024, 14, 14]	2,048
Conv2d-87	[-1, 1024, 14, 14]	524,288
BatchNorm2d-88	[-1, 1024, 14, 14]	2,048
ReLU-89	[-1, 1024, 14, 14]	0
Bottleneck-90	[-1, 1024, 14, 14]	0
Conv2d-91	[-1, 256, 14, 14]	262,144
BatchNorm2d-92	[-1, 256, 14, 14]	512
ReLU-93	[-1, 256, 14, 14]	0
Conv2d-94	[-1, 256, 14, 14]	589,824
BatchNorm2d-95	[-1, 256, 14, 14]	512
ReLU-96	[-1, 256, 14, 14]	0
Conv2d-97	[-1, 1024, 14, 14]	262,144
BatchNorm2d-98	[-1, 1024, 14, 14]	2,048
ReLU-99	[-1, 1024, 14, 14]	0
Bottleneck-100	[-1, 1024, 14, 14]	0
Conv2d-101	[-1, 256, 14, 14]	262,144
BatchNorm2d-102	[-1, 256, 14, 14]	512
ReLU-103	[-1, 256, 14, 14]	0
Conv2d-104	[-1, 256, 14, 14]	589,824
BatchNorm2d-105	[-1, 256, 14, 14]	512
ReLU-106	[-1, 256, 14, 14]	0
Conv2d-107	[-1, 1024, 14, 14]	262,144
BatchNorm2d-108	[-1, 1024, 14, 14]	2,048
ReLU-109	[-1, 1024, 14, 14]	0
Bottleneck-110	[-1, 1024, 14, 14]	0
Conv2d-111	[-1, 256, 14, 14]	262,144
BatchNorm2d-112	[-1, 256, 14, 14]	512
ReLU-113	[-1, 256, 14, 14]	0

```
        Conv2d-114          [-1, 256, 14, 14]         589,824
   BatchNorm2d-115          [-1, 256, 14, 14]             512
          ReLU-116          [-1, 256, 14, 14]               0
        Conv2d-117         [-1, 1024, 14, 14]         262,144
   BatchNorm2d-118         [-1, 1024, 14, 14]           2,048
          ReLU-119         [-1, 1024, 14, 14]               0
    Bottleneck-120         [-1, 1024, 14, 14]               0
        Conv2d-121          [-1, 256, 14, 14]         262,144
   BatchNorm2d-122          [-1, 256, 14, 14]             512
          ReLU-123          [-1, 256, 14, 14]               0
        Conv2d-124          [-1, 256, 14, 14]         589,824
   BatchNorm2d-125          [-1, 256, 14, 14]             512
          ReLU-126          [-1, 256, 14, 14]               0
        Conv2d-127         [-1, 1024, 14, 14]         262,144
   BatchNorm2d-128         [-1, 1024, 14, 14]           2,048
          ReLU-129         [-1, 1024, 14, 14]               0
    Bottleneck-130         [-1, 1024, 14, 14]               0
        Conv2d-131          [-1, 256, 14, 14]         262,144
   BatchNorm2d-132          [-1, 256, 14, 14]             512
          ReLU-133          [-1, 256, 14, 14]               0
        Conv2d-134          [-1, 256, 14, 14]         589,824
   BatchNorm2d-135          [-1, 256, 14, 14]             512
          ReLU-136          [-1, 256, 14, 14]               0
        Conv2d-137         [-1, 1024, 14, 14]         262,144
   BatchNorm2d-138         [-1, 1024, 14, 14]           2,048
          ReLU-139         [-1, 1024, 14, 14]               0
    Bottleneck-140         [-1, 1024, 14, 14]               0
        Conv2d-141          [-1, 512, 14, 14]         524,288
   BatchNorm2d-142          [-1, 512, 14, 14]           1,024
          ReLU-143          [-1, 512, 14, 14]               0
        Conv2d-144            [-1, 512, 7, 7]       2,359,296
   BatchNorm2d-145            [-1, 512, 7, 7]           1,024
          ReLU-146            [-1, 512, 7, 7]               0
        Conv2d-147           [-1, 2048, 7, 7]       1,048,576
   BatchNorm2d-148           [-1, 2048, 7, 7]           4,096
        Conv2d-149           [-1, 2048, 7, 7]       2,097,152
   BatchNorm2d-150           [-1, 2048, 7, 7]           4,096
          ReLU-151           [-1, 2048, 7, 7]               0
    Bottleneck-152           [-1, 2048, 7, 7]               0
        Conv2d-153            [-1, 512, 7, 7]       1,048,576
   BatchNorm2d-154            [-1, 512, 7, 7]           1,024
          ReLU-155            [-1, 512, 7, 7]               0
        Conv2d-156            [-1, 512, 7, 7]       2,359,296
   BatchNorm2d-157            [-1, 512, 7, 7]           1,024
          ReLU-158            [-1, 512, 7, 7]               0
        Conv2d-159           [-1, 2048, 7, 7]       1,048,576
   BatchNorm2d-160           [-1, 2048, 7, 7]           4,096
          ReLU-161           [-1, 2048, 7, 7]               0
    Bottleneck-162           [-1, 2048, 7, 7]               0
```

```
          Conv2d-163            [-1, 512, 7, 7]       1,048,576
     BatchNorm2d-164            [-1, 512, 7, 7]           1,024
            ReLU-165            [-1, 512, 7, 7]               0
          Conv2d-166            [-1, 512, 7, 7]       2,359,296
     BatchNorm2d-167            [-1, 512, 7, 7]           1,024
            ReLU-168            [-1, 512, 7, 7]               0
          Conv2d-169           [-1, 2048, 7, 7]       1,048,576
     BatchNorm2d-170           [-1, 2048, 7, 7]           4,096
            ReLU-171           [-1, 2048, 7, 7]               0
      Bottleneck-172           [-1, 2048, 7, 7]               0
AdaptiveAvgPool2d-173           [-1, 2048, 1, 1]               0
          Linear-174                   [-1, 10]          20,490
================================================================
Total params: 23,528,522
Trainable params: 23,528,522
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 286.55
Params size (MB): 89.75
Estimated Total Size (MB): 376.88
----------------------------------------------------------------
Epoch: 1
2.2946979999542236
2.3574068546295166
2.3407657146453857
Train Loss: 2.3310
Train Accuracy: 4.17%
Test Loss: 2.3344
Test Accuracy: 6.67%
Epoch: 2
2.377584934234619
2.3298561573028564
2.286466360092163
Train Loss: 2.3313
Train Accuracy: 8.33%
Test Loss: 2.3362
Test Accuracy: 6.67%
Epoch: 3
2.317997694015503
2.3509457111358643
2.311251640319824
Train Loss: 2.3267
Train Accuracy: 10.42%
Test Loss: 2.3207
Test Accuracy: 10.00%
Epoch: 4
2.339057683944702
2.3189327716827393
```

```
2.2845723628997803
Train Loss: 2.3142
Train Accuracy: 9.38%
Test Loss: 2.3078
Test Accuracy: 6.67%
Epoch: 5
2.302266836166382
2.3094396591186523
2.3284969329833984
Train Loss: 2.3134
Train Accuracy: 8.33%
Test Loss: 2.3024
Test Accuracy: 0.00%
Epoch: 6
2.2890167236328125
2.3025779724121094
2.3293771743774414
Train Loss: 2.3070
Train Accuracy: 6.25%
Test Loss: 2.3052
Test Accuracy: 0.00%
Epoch: 7
2.3336730003356934
2.313509941101074
2.3204541206359863
Train Loss: 2.3225
Train Accuracy: 12.50%
Test Loss: 2.3135
Test Accuracy: 3.33%
Epoch: 8
2.320241689682007
2.298161745071411
2.3121495246887207
Train Loss: 2.3102
Train Accuracy: 7.29%
Test Loss: 2.3183
Test Accuracy: 3.33%
Epoch: 9
2.305762529373169
2.2849740982055664
2.281937599182129
Train Loss: 2.2909
Train Accuracy: 7.29%
Test Loss: 2.3182
Test Accuracy: 6.67%
Epoch: 10
2.253018379211426
2.3571977615356445
2.275646448135376
Train Loss: 2.2953
```

```
Train Accuracy: 10.42%
Test Loss: 2.3139
Test Accuracy: 6.67%

<All keys matched successfully>
```