

TP4

September 6, 2026

0.0.1 Names : Chloé Habasque et Colin Coërchon

1 TP4 : Learning on a low budget

Théo Rudkiewicz, Cyriaque Rousselot

2 Introduction

Context :

Assume we are in a context where few “gold” labeled data samples are available for training, say

$$\mathcal{X}_{\text{train}} = \{(x_n, y_n)\}_{n \leq N_{\text{train}}}$$

where N_{train} is small.

A large test set $\mathcal{X}_{\text{test}}$ exists but is not accessible. (To make your task easier, we provide you with some data (named `test_dataset` in the code) that you can use to test your model, but you **must not** use it to train your model).

We also assume that we have a limited computational budget.

The goal of this practical session is to guide you through different methods that will help you get better results from few resources (data & compute).

In this practical, we will use the `resnet18` architecture. We will use models from the [pytorch vision hub](#).

3 QUESTIONS

3.1 Grading

You will be graded on 5 questions. You will need to provide 7 files : 1. This Notebook 2. `utils.py` 3. `last_layer_finetune.pth` (the file **must be of size less than 5Mo**) 4. `daug_resnet.pth` (the file **must be of size less than 50Mo**) 5. `final_model.pth` (the file **must be of size less than 50Mo**) 6. `drawing_lora.png` 7. `cutmix.png`

If the code you defined passes all our tests, you will get the full grade. Otherwise we will look at the intermediate questions in the notebook to give you partial credit.

Please provide clear and short answers between `<div class="alert alert-info">` `<your answer>` `</div>` tags (when it's not code).

Example of answer

```
[290]: import os

if not os.path.exists("data"):
    os.mkdir("data")
if not os.path.exists("data/TP4_images"):
    os.mkdir("data/TP4_images")
    !cd data/TP4_images && wget -O north_dataset_train.zip "https://nextcloud.
    ↳lisn.upsaclay.fr/index.php/s/yzQRWE2YjmFn9WA/download/north_dataset_train.zip"
    ↳&& unzip north_dataset_train.zip
    !cd data/TP4_images && wget -O north_dataset_test.zip "https://nextcloud.
    ↳lisn.upsaclay.fr/index.php/s/zntidWrFdYsGMDm/download/north_dataset_test.zip"
    ↳&& unzip north_dataset_test.zip
dir_path = "data/TP4_images/"
```

```
[291]: import torch
import torch.nn as nn
from torchvision import datasets, transforms, models
from torchmetrics.classification import BinaryAccuracy, Accuracy, ConfusionMatrix
# you can install torchmetrics with `pip install torchmetrics`
from torch.utils.data import Dataset, TensorDataset
import torch
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

os.environ["KMP_DUPLICATE_LIB_OK"] = "True"
```

```
[292]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

north_dataset = datasets.ImageFolder(
    dir_path + "north_dataset_sample",
    transform=transforms.Compose([transforms.ToTensor()]),
)
test_dataset = datasets.ImageFolder(
    dir_path + "north_dataset_test",
    transform=transforms.Compose([transforms.ToTensor()]),
)
base_model = models.resnet18()
base_model.to(device) # move the model to the GPU
```

```

[292]: 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): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
      1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)

```

```

    )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)

```

```

        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)

```

3.2 Question 1 :

Change the last layer of the resnet model so that its size fits the problem.

[268]: `# Hint`
`print(base_model)`

```

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): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

```

```

        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
    )
)
(layer2): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)

```

```

)
(layer3): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
(layer4): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(

```

```

        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
)
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=1000, bias=True)
)

```

```

[269]: num_classes = len(north_dataset.classes)

base_model.fc = torch.nn.Linear(in_features=512, out_features=num_classes).
        ↪to(device)

```

```

[270]: assert (
    base_model.fc.out_features == 2
) # we could also change the last layer to have 1 output. Do it with 2 so that
    ↪it matches our tests procedure during grading.

```

3.3 Question 2:

Train the last layer of a randomly initialized resnet model. Provide a function `precompute_features` in `utils.py` that creates a new dataset from the features precomputed by the model.

Intermediate question : Provide the training process in the notebook with training curve. Comment on the accuracy.

```

[272]: from utils import precompute_features

```

```

[273]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device {device}")

train_features_dataset = precompute_features(base_model, north_dataset, device)
train_loader = DataLoader(train_features_dataset, batch_size=32, shuffle=True)

test_features_dataset = precompute_features(base_model, test_dataset, device)
test_loader = DataLoader(test_features_dataset, batch_size=32, shuffle=False)

# Freeze the entire model except the last layer
for param in base_model.parameters():

```

```

    param.requires_grad = False

for param in base_model.fc.parameters():
    param.requires_grad = True

optimizer = optim.Adam(base_model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

def train_test(model, train_loader, test_loader, optimizer, criterion, epochs = 10):
    train_loss_values = []
    val_accuracy_values = []

    train_accuracy_values = []
    test_loss_values = []

    for epoch in range(epochs):
        model.train()
        epoch_loss = 0
        correct_train, total_train = 0, 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model.fc(inputs)
            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()

            # Calcul de la précision sur le train
            _, predicted = torch.max(outputs, 1)
            correct_train += (predicted == labels).sum().item()
            total_train += labels.size(0)

        train_loss = epoch_loss / len(train_loader)
        train_accuracy = 100 * correct_train / total_train
        train_loss_values.append(train_loss)
        train_accuracy_values.append(train_accuracy)

        # Évaluation sur l'ensemble de test
        model.eval()
        correct_test, total_test = 0, 0
        test_loss = 0

```

```

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model.fc(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

    _, predicted = torch.max(outputs, 1)
    correct_test += (predicted == labels).sum().item()
    total_test += labels.size(0)

val_accuracy = 100 * correct_test / total_test
avg_test_loss = test_loss / len(test_loader)
val_accuracy_values.append(val_accuracy)
test_loss_values.append(avg_test_loss)

print(f"Epoch {epoch+1}: "
      f"Train Loss={train_loss:.4f}, Train Acc={train_accuracy:.2f}%, "
      f"Test Loss={avg_test_loss:.4f}, Test Acc={val_accuracy:.2f}%")

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(train_loss_values, label="Train Loss")
plt.plot(test_loss_values, label="Test Loss")
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_accuracy_values, label="Train Accuracy")
plt.plot(val_accuracy_values, label="Test Accuracy", color='green')
plt.title("Accuracy over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.legend()

plt.tight_layout()
plt.show()
return

train_test(base_model, train_loader, test_loader, optimizer, criterion, epochs =
↪100)

```

Using device cuda

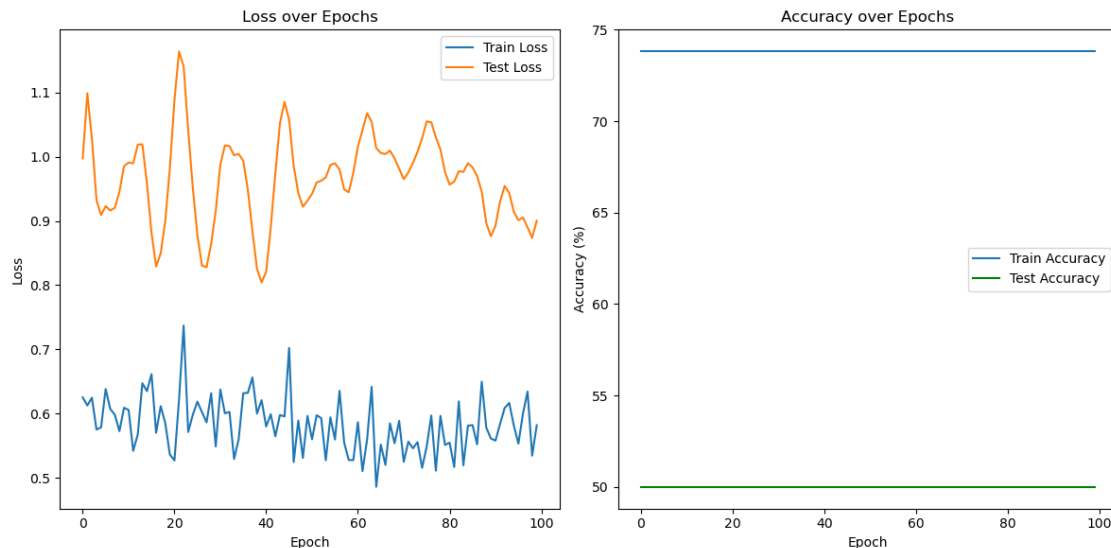
Precomputing features: 100%|| 1/1 [00:00<00:00, 9.42it/s]

Precomputing features: 100%|| 1/1 [00:00<00:00, 3.28it/s]

Epoch 1: Train Loss=0.6251, Train Acc=73.81%, Test Loss=0.9975, Test Acc=50.00%
Epoch 2: Train Loss=0.6125, Train Acc=73.81%, Test Loss=1.0990, Test Acc=50.00%
Epoch 3: Train Loss=0.6246, Train Acc=73.81%, Test Loss=1.0285, Test Acc=50.00%
Epoch 4: Train Loss=0.5750, Train Acc=73.81%, Test Loss=0.9323, Test Acc=50.00%
Epoch 5: Train Loss=0.5783, Train Acc=73.81%, Test Loss=0.9091, Test Acc=50.00%
Epoch 6: Train Loss=0.6382, Train Acc=73.81%, Test Loss=0.9232, Test Acc=50.00%
Epoch 7: Train Loss=0.6072, Train Acc=73.81%, Test Loss=0.9164, Test Acc=50.00%
Epoch 8: Train Loss=0.5981, Train Acc=73.81%, Test Loss=0.9205, Test Acc=50.00%
Epoch 9: Train Loss=0.5724, Train Acc=73.81%, Test Loss=0.9456, Test Acc=50.00%
Epoch 10: Train Loss=0.6090, Train Acc=73.81%, Test Loss=0.9853, Test Acc=50.00%
Epoch 11: Train Loss=0.6055, Train Acc=73.81%, Test Loss=0.9911, Test Acc=50.00%
Epoch 12: Train Loss=0.5419, Train Acc=73.81%, Test Loss=0.9897, Test Acc=50.00%
Epoch 13: Train Loss=0.5673, Train Acc=73.81%, Test Loss=1.0189, Test Acc=50.00%
Epoch 14: Train Loss=0.6472, Train Acc=73.81%, Test Loss=1.0195, Test Acc=50.00%
Epoch 15: Train Loss=0.6349, Train Acc=73.81%, Test Loss=0.9587, Test Acc=50.00%
Epoch 16: Train Loss=0.6614, Train Acc=73.81%, Test Loss=0.8812, Test Acc=50.00%
Epoch 17: Train Loss=0.5700, Train Acc=73.81%, Test Loss=0.8292, Test Acc=50.00%
Epoch 18: Train Loss=0.6116, Train Acc=73.81%, Test Loss=0.8496, Test Acc=50.00%
Epoch 19: Train Loss=0.5847, Train Acc=73.81%, Test Loss=0.9002, Test Acc=50.00%
Epoch 20: Train Loss=0.5361, Train Acc=73.81%, Test Loss=0.9828, Test Acc=50.00%
Epoch 21: Train Loss=0.5269, Train Acc=73.81%, Test Loss=1.0888, Test Acc=50.00%
Epoch 22: Train Loss=0.6221, Train Acc=73.81%, Test Loss=1.1641, Test Acc=50.00%
Epoch 23: Train Loss=0.7371, Train Acc=73.81%, Test Loss=1.1410, Test Acc=50.00%
Epoch 24: Train Loss=0.5712, Train Acc=73.81%, Test Loss=1.0410, Test Acc=50.00%
Epoch 25: Train Loss=0.5981, Train Acc=73.81%, Test Loss=0.9517, Test Acc=50.00%
Epoch 26: Train Loss=0.6185, Train Acc=73.81%, Test Loss=0.8781, Test Acc=50.00%
Epoch 27: Train Loss=0.6022, Train Acc=73.81%, Test Loss=0.8305, Test Acc=50.00%
Epoch 28: Train Loss=0.5863, Train Acc=73.81%, Test Loss=0.8279, Test Acc=50.00%
Epoch 29: Train Loss=0.6316, Train Acc=73.81%, Test Loss=0.8644, Test Acc=50.00%
Epoch 30: Train Loss=0.5486, Train Acc=73.81%, Test Loss=0.9157, Test Acc=50.00%
Epoch 31: Train Loss=0.6374, Train Acc=73.81%, Test Loss=0.9874, Test Acc=50.00%
Epoch 32: Train Loss=0.6003, Train Acc=73.81%, Test Loss=1.0179, Test Acc=50.00%
Epoch 33: Train Loss=0.6024, Train Acc=73.81%, Test Loss=1.0166, Test Acc=50.00%
Epoch 34: Train Loss=0.5292, Train Acc=73.81%, Test Loss=1.0024, Test Acc=50.00%
Epoch 35: Train Loss=0.5600, Train Acc=73.81%, Test Loss=1.0044, Test Acc=50.00%
Epoch 36: Train Loss=0.6317, Train Acc=73.81%, Test Loss=0.9937, Test Acc=50.00%
Epoch 37: Train Loss=0.6322, Train Acc=73.81%, Test Loss=0.9487, Test Acc=50.00%
Epoch 38: Train Loss=0.6562, Train Acc=73.81%, Test Loss=0.8844, Test Acc=50.00%
Epoch 39: Train Loss=0.5998, Train Acc=73.81%, Test Loss=0.8254, Test Acc=50.00%
Epoch 40: Train Loss=0.6208, Train Acc=73.81%, Test Loss=0.8042, Test Acc=50.00%
Epoch 41: Train Loss=0.5797, Train Acc=73.81%, Test Loss=0.8211, Test Acc=50.00%
Epoch 42: Train Loss=0.5989, Train Acc=73.81%, Test Loss=0.8904, Test Acc=50.00%
Epoch 43: Train Loss=0.5645, Train Acc=73.81%, Test Loss=0.9746, Test Acc=50.00%
Epoch 44: Train Loss=0.5977, Train Acc=73.81%, Test Loss=1.0521, Test Acc=50.00%
Epoch 45: Train Loss=0.5954, Train Acc=73.81%, Test Loss=1.0856, Test Acc=50.00%
Epoch 46: Train Loss=0.7023, Train Acc=73.81%, Test Loss=1.0580, Test Acc=50.00%

Epoch 47: Train Loss=0.5243, Train Acc=73.81%, Test Loss=0.9855, Test Acc=50.00%
Epoch 48: Train Loss=0.5890, Train Acc=73.81%, Test Loss=0.9438, Test Acc=50.00%
Epoch 49: Train Loss=0.5308, Train Acc=73.81%, Test Loss=0.9223, Test Acc=50.00%
Epoch 50: Train Loss=0.5963, Train Acc=73.81%, Test Loss=0.9315, Test Acc=50.00%
Epoch 51: Train Loss=0.5598, Train Acc=73.81%, Test Loss=0.9422, Test Acc=50.00%
Epoch 52: Train Loss=0.5973, Train Acc=73.81%, Test Loss=0.9600, Test Acc=50.00%
Epoch 53: Train Loss=0.5928, Train Acc=73.81%, Test Loss=0.9627, Test Acc=50.00%
Epoch 54: Train Loss=0.5272, Train Acc=73.81%, Test Loss=0.9678, Test Acc=50.00%
Epoch 55: Train Loss=0.5939, Train Acc=73.81%, Test Loss=0.9873, Test Acc=50.00%
Epoch 56: Train Loss=0.5595, Train Acc=73.81%, Test Loss=0.9899, Test Acc=50.00%
Epoch 57: Train Loss=0.6354, Train Acc=73.81%, Test Loss=0.9803, Test Acc=50.00%
Epoch 58: Train Loss=0.5549, Train Acc=73.81%, Test Loss=0.9494, Test Acc=50.00%
Epoch 59: Train Loss=0.5275, Train Acc=73.81%, Test Loss=0.9448, Test Acc=50.00%
Epoch 60: Train Loss=0.5274, Train Acc=73.81%, Test Loss=0.9751, Test Acc=50.00%
Epoch 61: Train Loss=0.5866, Train Acc=73.81%, Test Loss=1.0159, Test Acc=50.00%
Epoch 62: Train Loss=0.5103, Train Acc=73.81%, Test Loss=1.0415, Test Acc=50.00%
Epoch 63: Train Loss=0.5590, Train Acc=73.81%, Test Loss=1.0681, Test Acc=50.00%
Epoch 64: Train Loss=0.6416, Train Acc=73.81%, Test Loss=1.0545, Test Acc=50.00%
Epoch 65: Train Loss=0.4858, Train Acc=73.81%, Test Loss=1.0135, Test Acc=50.00%
Epoch 66: Train Loss=0.5517, Train Acc=73.81%, Test Loss=1.0059, Test Acc=50.00%
Epoch 67: Train Loss=0.5200, Train Acc=73.81%, Test Loss=1.0044, Test Acc=50.00%
Epoch 68: Train Loss=0.5846, Train Acc=73.81%, Test Loss=1.0096, Test Acc=50.00%
Epoch 69: Train Loss=0.5537, Train Acc=73.81%, Test Loss=0.9974, Test Acc=50.00%
Epoch 70: Train Loss=0.5885, Train Acc=73.81%, Test Loss=0.9814, Test Acc=50.00%
Epoch 71: Train Loss=0.5248, Train Acc=73.81%, Test Loss=0.9651, Test Acc=50.00%
Epoch 72: Train Loss=0.5561, Train Acc=73.81%, Test Loss=0.9756, Test Acc=50.00%
Epoch 73: Train Loss=0.5459, Train Acc=73.81%, Test Loss=0.9904, Test Acc=50.00%
Epoch 74: Train Loss=0.5554, Train Acc=73.81%, Test Loss=1.0077, Test Acc=50.00%
Epoch 75: Train Loss=0.5154, Train Acc=73.81%, Test Loss=1.0287, Test Acc=50.00%
Epoch 76: Train Loss=0.5485, Train Acc=73.81%, Test Loss=1.0552, Test Acc=50.00%
Epoch 77: Train Loss=0.5968, Train Acc=73.81%, Test Loss=1.0538, Test Acc=50.00%
Epoch 78: Train Loss=0.5109, Train Acc=73.81%, Test Loss=1.0304, Test Acc=50.00%
Epoch 79: Train Loss=0.5963, Train Acc=73.81%, Test Loss=1.0113, Test Acc=50.00%
Epoch 80: Train Loss=0.5511, Train Acc=73.81%, Test Loss=0.9763, Test Acc=50.00%
Epoch 81: Train Loss=0.5546, Train Acc=73.81%, Test Loss=0.9566, Test Acc=50.00%
Epoch 82: Train Loss=0.5167, Train Acc=73.81%, Test Loss=0.9612, Test Acc=50.00%
Epoch 83: Train Loss=0.6189, Train Acc=73.81%, Test Loss=0.9774, Test Acc=50.00%
Epoch 84: Train Loss=0.5193, Train Acc=73.81%, Test Loss=0.9767, Test Acc=50.00%
Epoch 85: Train Loss=0.5811, Train Acc=73.81%, Test Loss=0.9899, Test Acc=50.00%
Epoch 86: Train Loss=0.5820, Train Acc=73.81%, Test Loss=0.9835, Test Acc=50.00%
Epoch 87: Train Loss=0.5520, Train Acc=73.81%, Test Loss=0.9698, Test Acc=50.00%
Epoch 88: Train Loss=0.6495, Train Acc=73.81%, Test Loss=0.9458, Test Acc=50.00%
Epoch 89: Train Loss=0.5780, Train Acc=73.81%, Test Loss=0.8965, Test Acc=50.00%
Epoch 90: Train Loss=0.5611, Train Acc=73.81%, Test Loss=0.8762, Test Acc=50.00%
Epoch 91: Train Loss=0.5577, Train Acc=73.81%, Test Loss=0.8928, Test Acc=50.00%
Epoch 92: Train Loss=0.5836, Train Acc=73.81%, Test Loss=0.9303, Test Acc=50.00%
Epoch 93: Train Loss=0.6086, Train Acc=73.81%, Test Loss=0.9546, Test Acc=50.00%
Epoch 94: Train Loss=0.6165, Train Acc=73.81%, Test Loss=0.9436, Test Acc=50.00%

Epoch 95: Train Loss=0.5813, Train Acc=73.81%, Test Loss=0.9145, Test Acc=50.00%
Epoch 96: Train Loss=0.5531, Train Acc=73.81%, Test Loss=0.9012, Test Acc=50.00%
Epoch 97: Train Loss=0.6001, Train Acc=73.81%, Test Loss=0.9055, Test Acc=50.00%
Epoch 98: Train Loss=0.6342, Train Acc=73.81%, Test Loss=0.8900, Test Acc=50.00%
Epoch 99: Train Loss=0.5344, Train Acc=73.81%, Test Loss=0.8736, Test Acc=50.00%
Epoch 100: Train Loss=0.5818, Train Acc=73.81%, Test Loss=0.9003, Test Acc=50.00%



Replacing the model's final layer with `torch.nn.Linear(in_features=512, out_features=num_classes).to(device)` does not yield great results. Accuracy remains almost constant over the epochs, with:

- $\approx 97\%$ on the training set
- $\approx 60\%$ on the test set

There is a clear overfitting issue, which can likely be explained by the fact that the model uses a randomly initialized ResNet18. Its convolutional layers have no meaningful structure to extract useful features. In short, we need to find a better approach!

3.4 Question 3 :

Now start from a pretrained model on Imagenet (<https://pytorch.org/vision/stable/models.html#>) and only train the last layer. Provide the training process in the notebook with training curve.

Provide two files : (https://pytorch.org/tutorials/beginner/saving_loading_models.html) - a file `utils.py` containing only the last layer class `LastLayer` inheriting from `torch.nn.Module` architecture of your final model to load - a `last_layer_finetune.pth` file containing **only the last layer weights** (we will check the size)

We will test your model on final accuracy on a test set.

```
[274]: class LastLayer(nn.Module):
        def __init__(self):
            super().__init__()
            self.linear = nn.Linear(512, 2)  # 512 from ResNet18 output

        def forward(self, x):
            return self.linear(x)

[275]: # Model
resnet = models.resnet18(weights="DEFAULT")
for param in resnet.parameters():
    param.requires_grad = False  # Freeze entire model

fc = LastLayer()
resnet.fc = fc
resnet.to(device)

#Data
train_features_dataset = precompute_features(resnet, north_dataset, device)
train_loader = DataLoader(train_features_dataset, batch_size=32, shuffle=True)

test_features_dataset = precompute_features(resnet, test_dataset, device)
test_loader = DataLoader(test_features_dataset, batch_size=32, shuffle=False)

# Only train last layer
optimizer = optim.Adam(resnet.fc.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

train_test(resnet, train_loader, test_loader, optimizer, criterion, epochs = 100)
```

Precomputing features: 100%|| 1/1 [00:00<00:00, 15.01it/s]

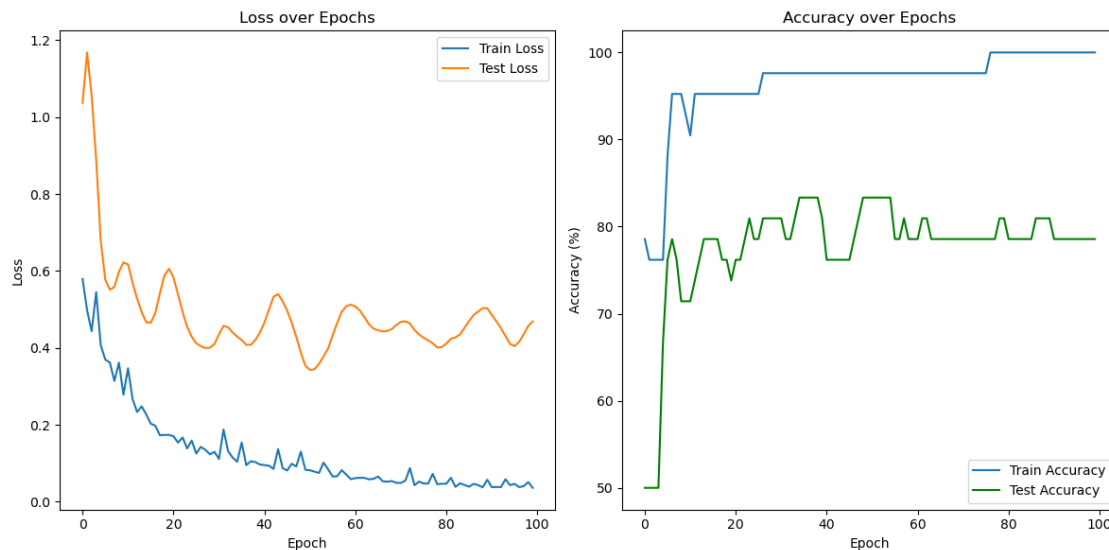
Precomputing features: 100%|| 1/1 [00:00<00:00, 13.65it/s]

```
Epoch 1: Train Loss=0.5794, Train Acc=78.57%, Test Loss=1.0371, Test Acc=50.00%
Epoch 2: Train Loss=0.4975, Train Acc=76.19%, Test Loss=1.1681, Test Acc=50.00%
Epoch 3: Train Loss=0.4433, Train Acc=76.19%, Test Loss=1.0563, Test Acc=50.00%
Epoch 4: Train Loss=0.5447, Train Acc=76.19%, Test Loss=0.8886, Test Acc=50.00%
Epoch 5: Train Loss=0.4070, Train Acc=76.19%, Test Loss=0.6781, Test Acc=66.67%
Epoch 6: Train Loss=0.3694, Train Acc=88.10%, Test Loss=0.5774, Test Acc=76.19%
Epoch 7: Train Loss=0.3619, Train Acc=95.24%, Test Loss=0.5512, Test Acc=78.57%
Epoch 8: Train Loss=0.3137, Train Acc=95.24%, Test Loss=0.5586, Test Acc=76.19%
Epoch 9: Train Loss=0.3616, Train Acc=95.24%, Test Loss=0.5976, Test Acc=71.43%
Epoch 10: Train Loss=0.2782, Train Acc=92.86%, Test Loss=0.6222, Test Acc=71.43%
Epoch 11: Train Loss=0.3470, Train Acc=90.48%, Test Loss=0.6168, Test Acc=71.43%
Epoch 12: Train Loss=0.2671, Train Acc=95.24%, Test Loss=0.5692, Test Acc=73.81%
Epoch 13: Train Loss=0.2331, Train Acc=95.24%, Test Loss=0.5279, Test Acc=76.19%
Epoch 14: Train Loss=0.2480, Train Acc=95.24%, Test Loss=0.4932, Test Acc=78.57%
```

Epoch 15: Train Loss=0.2268, Train Acc=95.24%, Test Loss=0.4666, Test Acc=78.57%
 Epoch 16: Train Loss=0.2029, Train Acc=95.24%, Test Loss=0.4650, Test Acc=78.57%
 Epoch 17: Train Loss=0.1974, Train Acc=95.24%, Test Loss=0.4906, Test Acc=78.57%
 Epoch 18: Train Loss=0.1728, Train Acc=95.24%, Test Loss=0.5415, Test Acc=76.19%
 Epoch 19: Train Loss=0.1735, Train Acc=95.24%, Test Loss=0.5882, Test Acc=76.19%
 Epoch 20: Train Loss=0.1736, Train Acc=95.24%, Test Loss=0.6057, Test Acc=73.81%
 Epoch 21: Train Loss=0.1702, Train Acc=95.24%, Test Loss=0.5828, Test Acc=76.19%
 Epoch 22: Train Loss=0.1537, Train Acc=95.24%, Test Loss=0.5389, Test Acc=76.19%
 Epoch 23: Train Loss=0.1670, Train Acc=95.24%, Test Loss=0.4934, Test Acc=78.57%
 Epoch 24: Train Loss=0.1383, Train Acc=95.24%, Test Loss=0.4546, Test Acc=80.95%
 Epoch 25: Train Loss=0.1589, Train Acc=95.24%, Test Loss=0.4290, Test Acc=78.57%
 Epoch 26: Train Loss=0.1253, Train Acc=95.24%, Test Loss=0.4118, Test Acc=78.57%
 Epoch 27: Train Loss=0.1425, Train Acc=97.62%, Test Loss=0.4044, Test Acc=80.95%
 Epoch 28: Train Loss=0.1349, Train Acc=97.62%, Test Loss=0.3994, Test Acc=80.95%
 Epoch 29: Train Loss=0.1233, Train Acc=97.62%, Test Loss=0.4004, Test Acc=80.95%
 Epoch 30: Train Loss=0.1295, Train Acc=97.62%, Test Loss=0.4098, Test Acc=80.95%
 Epoch 31: Train Loss=0.1110, Train Acc=97.62%, Test Loss=0.4350, Test Acc=80.95%
 Epoch 32: Train Loss=0.1878, Train Acc=97.62%, Test Loss=0.4573, Test Acc=78.57%
 Epoch 33: Train Loss=0.1315, Train Acc=97.62%, Test Loss=0.4534, Test Acc=78.57%
 Epoch 34: Train Loss=0.1147, Train Acc=97.62%, Test Loss=0.4399, Test Acc=80.95%
 Epoch 35: Train Loss=0.1039, Train Acc=97.62%, Test Loss=0.4293, Test Acc=83.33%
 Epoch 36: Train Loss=0.1538, Train Acc=97.62%, Test Loss=0.4201, Test Acc=83.33%
 Epoch 37: Train Loss=0.0949, Train Acc=97.62%, Test Loss=0.4079, Test Acc=83.33%
 Epoch 38: Train Loss=0.1052, Train Acc=97.62%, Test Loss=0.4079, Test Acc=83.33%
 Epoch 39: Train Loss=0.1029, Train Acc=97.62%, Test Loss=0.4207, Test Acc=83.33%
 Epoch 40: Train Loss=0.0968, Train Acc=97.62%, Test Loss=0.4398, Test Acc=80.95%
 Epoch 41: Train Loss=0.0950, Train Acc=97.62%, Test Loss=0.4660, Test Acc=76.19%
 Epoch 42: Train Loss=0.0936, Train Acc=97.62%, Test Loss=0.4997, Test Acc=76.19%
 Epoch 43: Train Loss=0.0851, Train Acc=97.62%, Test Loss=0.5328, Test Acc=76.19%
 Epoch 44: Train Loss=0.1367, Train Acc=97.62%, Test Loss=0.5398, Test Acc=76.19%
 Epoch 45: Train Loss=0.0874, Train Acc=97.62%, Test Loss=0.5210, Test Acc=76.19%
 Epoch 46: Train Loss=0.0811, Train Acc=97.62%, Test Loss=0.4967, Test Acc=76.19%
 Epoch 47: Train Loss=0.0987, Train Acc=97.62%, Test Loss=0.4653, Test Acc=78.57%
 Epoch 48: Train Loss=0.0914, Train Acc=97.62%, Test Loss=0.4288, Test Acc=80.95%
 Epoch 49: Train Loss=0.1300, Train Acc=97.62%, Test Loss=0.3878, Test Acc=83.33%
 Epoch 50: Train Loss=0.0830, Train Acc=97.62%, Test Loss=0.3535, Test Acc=83.33%
 Epoch 51: Train Loss=0.0816, Train Acc=97.62%, Test Loss=0.3425, Test Acc=83.33%
 Epoch 52: Train Loss=0.0782, Train Acc=97.62%, Test Loss=0.3445, Test Acc=83.33%
 Epoch 53: Train Loss=0.0747, Train Acc=97.62%, Test Loss=0.3585, Test Acc=83.33%
 Epoch 54: Train Loss=0.1016, Train Acc=97.62%, Test Loss=0.3780, Test Acc=83.33%
 Epoch 55: Train Loss=0.0843, Train Acc=97.62%, Test Loss=0.3986, Test Acc=83.33%
 Epoch 56: Train Loss=0.0653, Train Acc=97.62%, Test Loss=0.4326, Test Acc=78.57%
 Epoch 57: Train Loss=0.0664, Train Acc=97.62%, Test Loss=0.4650, Test Acc=78.57%
 Epoch 58: Train Loss=0.0820, Train Acc=97.62%, Test Loss=0.4939, Test Acc=80.95%
 Epoch 59: Train Loss=0.0702, Train Acc=97.62%, Test Loss=0.5084, Test Acc=78.57%
 Epoch 60: Train Loss=0.0585, Train Acc=97.62%, Test Loss=0.5125, Test Acc=78.57%
 Epoch 61: Train Loss=0.0614, Train Acc=97.62%, Test Loss=0.5073, Test Acc=78.57%
 Epoch 62: Train Loss=0.0621, Train Acc=97.62%, Test Loss=0.4970, Test Acc=80.95%

Epoch 63: Train Loss=0.0620, Train Acc=97.62%, Test Loss=0.4793, Test Acc=80.95%
Epoch 64: Train Loss=0.0581, Train Acc=97.62%, Test Loss=0.4612, Test Acc=78.57%
Epoch 65: Train Loss=0.0596, Train Acc=97.62%, Test Loss=0.4499, Test Acc=78.57%
Epoch 66: Train Loss=0.0654, Train Acc=97.62%, Test Loss=0.4457, Test Acc=78.57%
Epoch 67: Train Loss=0.0533, Train Acc=97.62%, Test Loss=0.4426, Test Acc=78.57%
Epoch 68: Train Loss=0.0519, Train Acc=97.62%, Test Loss=0.4439, Test Acc=78.57%
Epoch 69: Train Loss=0.0537, Train Acc=97.62%, Test Loss=0.4489, Test Acc=78.57%
Epoch 70: Train Loss=0.0491, Train Acc=97.62%, Test Loss=0.4593, Test Acc=78.57%
Epoch 71: Train Loss=0.0487, Train Acc=97.62%, Test Loss=0.4677, Test Acc=78.57%
Epoch 72: Train Loss=0.0551, Train Acc=97.62%, Test Loss=0.4689, Test Acc=78.57%
Epoch 73: Train Loss=0.0874, Train Acc=97.62%, Test Loss=0.4632, Test Acc=78.57%
Epoch 74: Train Loss=0.0429, Train Acc=97.62%, Test Loss=0.4463, Test Acc=78.57%
Epoch 75: Train Loss=0.0522, Train Acc=97.62%, Test Loss=0.4346, Test Acc=78.57%
Epoch 76: Train Loss=0.0473, Train Acc=97.62%, Test Loss=0.4264, Test Acc=78.57%
Epoch 77: Train Loss=0.0471, Train Acc=100.00%, Test Loss=0.4194, Test Acc=78.57%
Epoch 78: Train Loss=0.0722, Train Acc=100.00%, Test Loss=0.4119, Test Acc=78.57%
Epoch 79: Train Loss=0.0454, Train Acc=100.00%, Test Loss=0.4009, Test Acc=80.95%
Epoch 80: Train Loss=0.0466, Train Acc=100.00%, Test Loss=0.4022, Test Acc=80.95%
Epoch 81: Train Loss=0.0466, Train Acc=100.00%, Test Loss=0.4111, Test Acc=78.57%
Epoch 82: Train Loss=0.0622, Train Acc=100.00%, Test Loss=0.4235, Test Acc=78.57%
Epoch 83: Train Loss=0.0384, Train Acc=100.00%, Test Loss=0.4271, Test Acc=78.57%
Epoch 84: Train Loss=0.0479, Train Acc=100.00%, Test Loss=0.4349, Test Acc=78.57%
Epoch 85: Train Loss=0.0434, Train Acc=100.00%, Test Loss=0.4522, Test Acc=78.57%
Epoch 86: Train Loss=0.0390, Train Acc=100.00%, Test Loss=0.4699, Test Acc=78.57%
Epoch 87: Train Loss=0.0463, Train Acc=100.00%, Test Loss=0.4851, Test Acc=80.95%
Epoch 88: Train Loss=0.0430, Train Acc=100.00%, Test Loss=0.4943, Test Acc=80.95%
Epoch 89: Train Loss=0.0373, Train Acc=100.00%, Test Loss=0.5035, Test Acc=80.95%
Epoch 90: Train Loss=0.0572, Train Acc=100.00%, Test Loss=0.5029, Test Acc=80.95%
Epoch 91: Train Loss=0.0377, Train Acc=100.00%, Test Loss=0.4856, Test Acc=78.57%
Epoch 92: Train Loss=0.0383, Train Acc=100.00%, Test Loss=0.4690, Test Acc=78.57%
Epoch 93: Train Loss=0.0378, Train Acc=100.00%, Test Loss=0.4513, Test Acc=78.57%

Epoch 94: Train Loss=0.0583, Train Acc=100.00%, Test Loss=0.4310, Test Acc=78.57%
 Epoch 95: Train Loss=0.0431, Train Acc=100.00%, Test Loss=0.4101, Test Acc=78.57%
 Epoch 96: Train Loss=0.0464, Train Acc=100.00%, Test Loss=0.4046, Test Acc=78.57%
 Epoch 97: Train Loss=0.0379, Train Acc=100.00%, Test Loss=0.4155, Test Acc=78.57%
 Epoch 98: Train Loss=0.0404, Train Acc=100.00%, Test Loss=0.4352, Test Acc=78.57%
 Epoch 99: Train Loss=0.0509, Train Acc=100.00%, Test Loss=0.4568, Test Acc=78.57%
 Epoch 100: Train Loss=0.0360, Train Acc=100.00%, Test Loss=0.4686, Test Acc=78.57%



```
[276]: # torch.save(base_model.fc.state_dict(), "lastlayer.pth")
torch.save(resnet.fc.state_dict(), "lastlayer_finetune.pth")
```

```
[277]: # from utils import LastLayer
resnet = models.resnet18(weights="DEFAULT")
fc = LastLayer() # ! Important : No argument
fc.load_state_dict(torch.load("lastlayer_finetune.pth", weights_only=True))
resnet.fc = fc
resnet.eval()
```

```
[277]: 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): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=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)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,

```

```

1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    )
    (layer3): Sequential(
    (0): BasicBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (downsample): Sequential(
    (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    )
    (1): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    )
    (layer4): Sequential(
    (0): BasicBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,

```

```

1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): LastLayer(
  (linear): Linear(in_features=512, out_features=2, bias=True)
)
)

```

3.5 Question 4 :

Perform LoRA <https://arxiv.org/pdf/2106.09685> on the model (We are perfectly fine if you use an external library **for this question only**, and of course use it in the next questions). (Warning : without data augmentation it may not improve the accuracy.)

Intermediate question : Describe LoRA. There are different ways of implementing LoRa for convolutions. You can choose your preferred one. Explain the version of LoRa you used, provide a drawing of the process in the `drawing_lora.png` file. (Hint: you can obtain a small rank convolution by combining a convolution and a 1x1 convolution. One of the two goes from a higher number of channels to a lower number of channels and the other one restores the number of channels.)

LoRA (Low-Rank Adaptation) is a technique to fine-tune large models without updating all of their parameters. Instead of training the full weight matrices, LoRA learns low-rank decompositions (often denoted $A \times B$) that are added to the frozen original weights. This substantially reduces the number of trainable parameters while still allowing the model to adapt to new tasks.

In our case, we applied LoRA to the **last two convolutional layers** (layer3 and layer4 of ResNet) and to the **final fully connected (fc) layer**. We froze the rest of the network to preserve its pre-trained knowledge and only updated the newly introduced low-rank modules, thus minimizing both computational cost and the risk of overfitting on a small dataset.

```
[278]: import loralib as lora
```

```
[279]: def train_test2(model, train_loader, test_loader, optimizer, criterion,
    ↪ epochs=10):
    train_loss_values = []
    val_accuracy_values = []

    train_accuracy_values = []
    test_loss_values = []

    for epoch in range(epochs):
        model.train()
        epoch_loss = 0
        correct_train, total_train = 0, 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            # >>> FIX : si on a un label [B, 2], on le convertit en [B]
            if labels.dim() == 2:
                labels = labels.argmax(dim=1)

            # Passe l'image complète dans le modèle
            outputs = model(inputs)

            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()

            # Calcul de la précision sur le train
            _, predicted = torch.max(outputs, 1)
            correct_train += (predicted == labels).sum().item()
            total_train += labels.size(0)

        train_loss = epoch_loss / len(train_loader)
        train_accuracy = 100 * correct_train / total_train
        train_loss_values.append(train_loss)
        train_accuracy_values.append(train_accuracy)

        # Évaluation sur l'ensemble de test
        model.eval()
        correct_test, total_test = 0, 0
        test_loss = 0
```

```

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

    _, predicted = torch.max(outputs, 1)
    correct_test += (predicted == labels).sum().item()
    total_test += labels.size(0)

val_accuracy = 100 * correct_test / total_test
avg_test_loss = test_loss / len(test_loader)
val_accuracy_values.append(val_accuracy)
test_loss_values.append(avg_test_loss)

print(f"Epoch {epoch+1}: "
      f"Train Loss={train_loss:.4f}, Train Acc={train_accuracy:.2f}%, "
      f"Test Loss={avg_test_loss:.4f}, Test Acc={val_accuracy:.2f}%")

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(train_loss_values, label="Train Loss")
plt.plot(test_loss_values, label="Test Loss")
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_accuracy_values, label="Train Accuracy")
plt.plot(val_accuracy_values, label="Test Accuracy")
plt.title("Accuracy over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.legend()

plt.tight_layout()
plt.show()

```

```

[280]: class LoRALinear(lora.Linear):
        def __init__(self, in_features, out_features, r=8):
            super().__init__(in_features, out_features, r=r)

lora_model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)

```

```

lora_model = lora_model.to(device)
print(lora_model)

num_classes = 2
lora_model.fc = LoRALinear(512, num_classes, r=8)
lora_model.fc.to(device)

```

```

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): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)

```

```

        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )

```

```

    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=1000, bias=True)
  )

```

[280]: LoRALinear(in_features=512, out_features=2, bias=True)

```

[ ]: import torch
import torch.nn.functional as F
import loralib as lora

class MyLoRAConv2d(lora.Conv2d):
    # On redéfinit _conv_forward comme avant
    def _conv_forward(self, input, weight, bias):
        return F.conv2d(
            input, weight, bias,
            stride=self.conv.stride,
            padding=self.conv.padding,

```

```

        dilation=self.conv.dilation,
        groups=self.conv.groups
    )

    def forward(self, x):
        return self._conv_forward(x, self.conv.weight, self.conv.bias)

    @property
    def weight(self):
        return self.conv.weight

    @weight.setter
    def weight(self, value):
        self.conv.weight = value

    @property
    def bias(self):
        return self.conv.bias

    @bias.setter
    def bias(self, value):
        self.conv.bias = value

def replace_conv_with_lora(module, r=8):
    for name, child in list(module.named_children()):
        if isinstance(child, nn.Conv2d):
            new_conv = MyLoRAConv2d(
                in_channels=child.in_channels,
                out_channels=child.out_channels,
                kernel_size=child.kernel_size[0],
                stride=child.stride[0],
                padding=child.padding[0],
                dilation=child.dilation[0],
                groups=child.groups,
                bias=(child.bias is not None),
                r=r
            )

            with torch.no_grad():
                new_conv.conv.weight.copy_(child.weight)
                if child.bias is not None:
                    new_conv.conv.bias.copy_(child.bias)

            setattr(module, name, new_conv)
        else:
            replace_conv_with_lora(child, r=r)

```

```
[282]: replace_conv_with_lora(lora_model.layer3, r=8)
replace_conv_with_lora(lora_model.layer4, r=8)

lora_model.to(device)
```

```
[282]: 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): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=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)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
```

```

track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
)
(layer3): Sequential(
  (0): BasicBlock(
    (conv1): MyLoRAConv2d(
      (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    )
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): MyLoRAConv2d(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    )
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (downsample): Sequential(
      (0): MyLoRAConv2d(
        (conv): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
      )
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): MyLoRAConv2d(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)

```

```

    )
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): MyLoRAConv2d(
        (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    )
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    )
    (layer4): Sequential(
    (0): BasicBlock(
        (conv1): MyLoRAConv2d(
            (conv): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        )
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): MyLoRAConv2d(
            (conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        )
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
            (0): MyLoRAConv2d(
                (conv): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2),
bias=False)
            )
            (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
    )
    )
    (1): BasicBlock(
        (conv1): MyLoRAConv2d(
            (conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        )
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): MyLoRAConv2d(
            (conv): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)

```

```

    )
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): LoRALinear(in_features=512, out_features=2, bias=True)
)

```

```

[283]: for param in lora_model.layer1.parameters():
        param.requires_grad = False

for param in lora_model.layer2.parameters():
        param.requires_grad = False

train_loader = DataLoader(north_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# On optimise seulement les params requires_grad=True
params_to_train = [p for p in lora_model.parameters() if p.requires_grad]
optimizer = optim.Adam(params_to_train, lr=0.001)
criterion = nn.CrossEntropyLoss()

train_test2(lora_model, train_loader, test_loader, optimizer, criterion,
↪ epochs=100)

```

```

Epoch 1: Train Loss=1.0522, Train Acc=26.19%, Test Loss=0.7086, Test Acc=47.62%
Epoch 2: Train Loss=0.8923, Train Acc=28.57%, Test Loss=0.6995, Test Acc=42.86%
Epoch 3: Train Loss=0.8495, Train Acc=30.95%, Test Loss=0.6686, Test Acc=52.38%
Epoch 4: Train Loss=0.8337, Train Acc=30.95%, Test Loss=0.6319, Test Acc=54.76%
Epoch 5: Train Loss=0.8846, Train Acc=40.48%, Test Loss=0.6005, Test Acc=54.76%
Epoch 6: Train Loss=0.8420, Train Acc=50.00%, Test Loss=0.5742, Test Acc=57.14%
Epoch 7: Train Loss=0.5716, Train Acc=54.76%, Test Loss=0.5517, Test Acc=59.52%
Epoch 8: Train Loss=0.5712, Train Acc=76.19%, Test Loss=0.5282, Test Acc=64.29%
Epoch 9: Train Loss=0.4589, Train Acc=85.71%, Test Loss=0.5174, Test Acc=64.29%
Epoch 10: Train Loss=0.4319, Train Acc=92.86%, Test Loss=0.5012, Test Acc=73.81%
Epoch 11: Train Loss=0.3292, Train Acc=95.24%, Test Loss=0.5006, Test Acc=78.57%
Epoch 12: Train Loss=0.3760, Train Acc=95.24%, Test Loss=0.5037, Test Acc=78.57%
Epoch 13: Train Loss=0.2590, Train Acc=100.00%, Test Loss=0.5395, Test
Acc=76.19%
Epoch 14: Train Loss=0.2940, Train Acc=97.62%, Test Loss=0.5717, Test Acc=76.19%
Epoch 15: Train Loss=0.2364, Train Acc=100.00%, Test Loss=0.5693, Test
Acc=78.57%
Epoch 16: Train Loss=0.2605, Train Acc=100.00%, Test Loss=0.5467, Test
Acc=80.95%
Epoch 17: Train Loss=0.1681, Train Acc=100.00%, Test Loss=0.5496, Test
Acc=78.57%

```

Epoch 18: Train Loss=0.1329, Train Acc=100.00%, Test Loss=0.5661, Test Acc=76.19%

Epoch 19: Train Loss=0.1165, Train Acc=100.00%, Test Loss=0.5731, Test Acc=78.57%

Epoch 20: Train Loss=0.1627, Train Acc=100.00%, Test Loss=0.6130, Test Acc=76.19%

Epoch 21: Train Loss=0.1180, Train Acc=100.00%, Test Loss=0.6292, Test Acc=76.19%

Epoch 22: Train Loss=0.0969, Train Acc=100.00%, Test Loss=0.6655, Test Acc=73.81%

Epoch 23: Train Loss=0.0888, Train Acc=100.00%, Test Loss=0.6926, Test Acc=71.43%

Epoch 24: Train Loss=0.0797, Train Acc=100.00%, Test Loss=0.7015, Test Acc=71.43%

Epoch 25: Train Loss=0.1018, Train Acc=100.00%, Test Loss=0.6790, Test Acc=71.43%

Epoch 26: Train Loss=0.0734, Train Acc=100.00%, Test Loss=0.6764, Test Acc=73.81%

Epoch 27: Train Loss=0.0955, Train Acc=100.00%, Test Loss=0.6403, Test Acc=76.19%

Epoch 28: Train Loss=0.0863, Train Acc=100.00%, Test Loss=0.6629, Test Acc=76.19%

Epoch 29: Train Loss=0.0589, Train Acc=100.00%, Test Loss=0.6749, Test Acc=76.19%

Epoch 30: Train Loss=0.0516, Train Acc=100.00%, Test Loss=0.6648, Test Acc=76.19%

Epoch 31: Train Loss=0.0553, Train Acc=100.00%, Test Loss=0.6603, Test Acc=76.19%

Epoch 32: Train Loss=0.0483, Train Acc=100.00%, Test Loss=0.6743, Test Acc=76.19%

Epoch 33: Train Loss=0.0456, Train Acc=100.00%, Test Loss=0.6819, Test Acc=76.19%

Epoch 34: Train Loss=0.0496, Train Acc=100.00%, Test Loss=0.6922, Test Acc=76.19%

Epoch 35: Train Loss=0.0582, Train Acc=100.00%, Test Loss=0.7432, Test Acc=76.19%

Epoch 36: Train Loss=0.0542, Train Acc=100.00%, Test Loss=0.7563, Test Acc=76.19%

Epoch 37: Train Loss=0.0371, Train Acc=100.00%, Test Loss=0.7589, Test Acc=76.19%

Epoch 38: Train Loss=0.0325, Train Acc=100.00%, Test Loss=0.7324, Test Acc=76.19%

Epoch 39: Train Loss=0.0374, Train Acc=100.00%, Test Loss=0.7114, Test Acc=76.19%

Epoch 40: Train Loss=0.0322, Train Acc=100.00%, Test Loss=0.6707, Test Acc=76.19%

Epoch 41: Train Loss=0.0320, Train Acc=100.00%, Test Loss=0.6601, Test Acc=76.19%

Epoch 42: Train Loss=0.1843, Train Acc=97.62%, Test Loss=0.7109, Test Acc=76.19%
Epoch 43: Train Loss=0.0497, Train Acc=100.00%, Test Loss=0.6090, Test Acc=76.19%
Epoch 44: Train Loss=0.0301, Train Acc=100.00%, Test Loss=0.5573, Test Acc=78.57%
Epoch 45: Train Loss=0.0295, Train Acc=100.00%, Test Loss=0.5493, Test Acc=78.57%
Epoch 46: Train Loss=0.0491, Train Acc=100.00%, Test Loss=0.5399, Test Acc=78.57%
Epoch 47: Train Loss=0.0544, Train Acc=100.00%, Test Loss=0.5514, Test Acc=76.19%
Epoch 48: Train Loss=0.0667, Train Acc=100.00%, Test Loss=0.6183, Test Acc=76.19%
Epoch 49: Train Loss=0.0254, Train Acc=100.00%, Test Loss=0.5769, Test Acc=76.19%
Epoch 50: Train Loss=0.0213, Train Acc=100.00%, Test Loss=0.5352, Test Acc=76.19%
Epoch 51: Train Loss=0.0240, Train Acc=100.00%, Test Loss=0.5120, Test Acc=76.19%
Epoch 52: Train Loss=0.0347, Train Acc=100.00%, Test Loss=0.5341, Test Acc=76.19%
Epoch 53: Train Loss=0.0274, Train Acc=100.00%, Test Loss=0.5355, Test Acc=76.19%
Epoch 54: Train Loss=0.0325, Train Acc=100.00%, Test Loss=0.5795, Test Acc=76.19%
Epoch 55: Train Loss=0.0616, Train Acc=100.00%, Test Loss=0.6154, Test Acc=78.57%
Epoch 56: Train Loss=0.0269, Train Acc=100.00%, Test Loss=0.6151, Test Acc=76.19%
Epoch 57: Train Loss=0.0197, Train Acc=100.00%, Test Loss=0.6053, Test Acc=78.57%
Epoch 58: Train Loss=0.0194, Train Acc=100.00%, Test Loss=0.5926, Test Acc=78.57%
Epoch 59: Train Loss=0.0179, Train Acc=100.00%, Test Loss=0.5765, Test Acc=78.57%
Epoch 60: Train Loss=0.0167, Train Acc=100.00%, Test Loss=0.5602, Test Acc=78.57%
Epoch 61: Train Loss=0.0155, Train Acc=100.00%, Test Loss=0.5555, Test Acc=80.95%
Epoch 62: Train Loss=0.0189, Train Acc=100.00%, Test Loss=0.5274, Test Acc=80.95%
Epoch 63: Train Loss=0.0239, Train Acc=100.00%, Test Loss=0.5424, Test Acc=78.57%
Epoch 64: Train Loss=0.0178, Train Acc=100.00%, Test Loss=0.5369, Test Acc=78.57%
Epoch 65: Train Loss=0.0336, Train Acc=100.00%, Test Loss=0.5152, Test Acc=83.33%
Epoch 66: Train Loss=0.0211, Train Acc=100.00%, Test Loss=0.5550, Test

Acc=80.95%

Epoch 67: Train Loss=0.0388, Train Acc=100.00%, Test Loss=0.5611, Test Acc=78.57%

Epoch 68: Train Loss=0.0216, Train Acc=100.00%, Test Loss=0.6054, Test Acc=80.95%

Epoch 69: Train Loss=0.0240, Train Acc=100.00%, Test Loss=0.6000, Test Acc=80.95%

Epoch 70: Train Loss=0.0424, Train Acc=100.00%, Test Loss=0.6222, Test Acc=78.57%

Epoch 71: Train Loss=0.0147, Train Acc=100.00%, Test Loss=0.5894, Test Acc=78.57%

Epoch 72: Train Loss=0.0116, Train Acc=100.00%, Test Loss=0.5934, Test Acc=78.57%

Epoch 73: Train Loss=0.0302, Train Acc=100.00%, Test Loss=0.5506, Test Acc=78.57%

Epoch 74: Train Loss=0.0308, Train Acc=100.00%, Test Loss=0.5177, Test Acc=80.95%

Epoch 75: Train Loss=0.0116, Train Acc=100.00%, Test Loss=0.5304, Test Acc=80.95%

Epoch 76: Train Loss=0.0106, Train Acc=100.00%, Test Loss=0.5387, Test Acc=80.95%

Epoch 77: Train Loss=0.0505, Train Acc=100.00%, Test Loss=0.5806, Test Acc=78.57%

Epoch 78: Train Loss=0.0116, Train Acc=100.00%, Test Loss=0.5929, Test Acc=80.95%

Epoch 79: Train Loss=0.0123, Train Acc=100.00%, Test Loss=0.6070, Test Acc=78.57%

Epoch 80: Train Loss=0.0095, Train Acc=100.00%, Test Loss=0.6349, Test Acc=78.57%

Epoch 81: Train Loss=0.0122, Train Acc=100.00%, Test Loss=0.6433, Test Acc=78.57%

Epoch 82: Train Loss=0.0172, Train Acc=100.00%, Test Loss=0.6699, Test Acc=78.57%

Epoch 83: Train Loss=0.0239, Train Acc=100.00%, Test Loss=0.6231, Test Acc=80.95%

Epoch 84: Train Loss=0.0111, Train Acc=100.00%, Test Loss=0.6017, Test Acc=80.95%

Epoch 85: Train Loss=0.0107, Train Acc=100.00%, Test Loss=0.6090, Test Acc=80.95%

Epoch 86: Train Loss=0.0078, Train Acc=100.00%, Test Loss=0.6158, Test Acc=80.95%

Epoch 87: Train Loss=0.0077, Train Acc=100.00%, Test Loss=0.6223, Test Acc=80.95%

Epoch 88: Train Loss=0.0727, Train Acc=100.00%, Test Loss=0.5668, Test Acc=80.95%

Epoch 89: Train Loss=0.0077, Train Acc=100.00%, Test Loss=0.5973, Test Acc=78.57%

Epoch 90: Train Loss=0.0196, Train Acc=100.00%, Test Loss=0.5829, Test

Acc=78.57%

Epoch 91: Train Loss=0.0072, Train Acc=100.00%, Test Loss=0.6229, Test Acc=78.57%

Epoch 92: Train Loss=0.0176, Train Acc=100.00%, Test Loss=0.6724, Test Acc=76.19%

Epoch 93: Train Loss=0.0176, Train Acc=100.00%, Test Loss=0.7040, Test Acc=76.19%

Epoch 94: Train Loss=0.0076, Train Acc=100.00%, Test Loss=0.7055, Test Acc=78.57%

Epoch 95: Train Loss=0.0158, Train Acc=100.00%, Test Loss=0.7338, Test Acc=78.57%

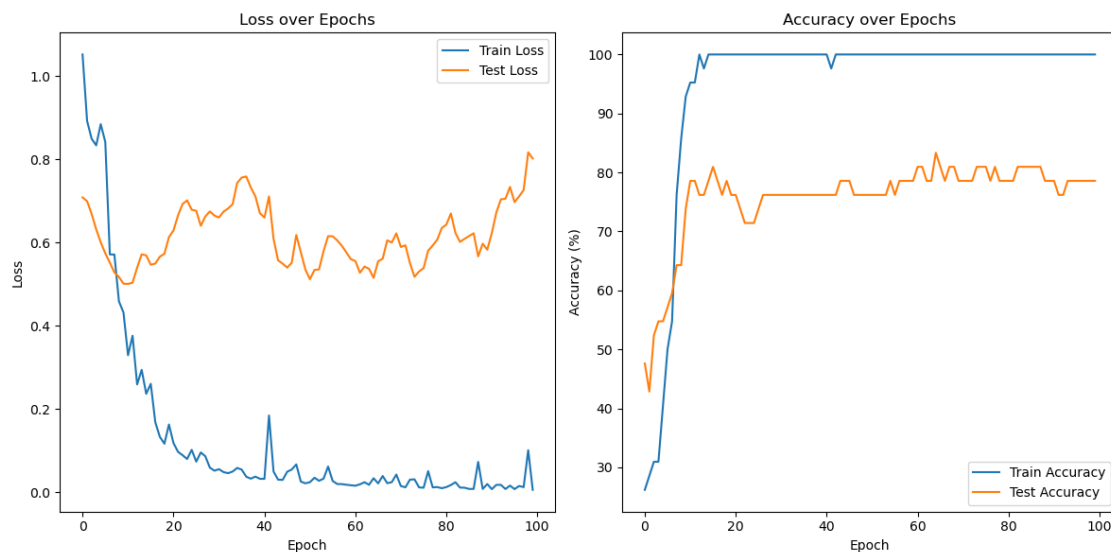
Epoch 96: Train Loss=0.0073, Train Acc=100.00%, Test Loss=0.6973, Test Acc=78.57%

Epoch 97: Train Loss=0.0149, Train Acc=100.00%, Test Loss=0.7111, Test Acc=78.57%

Epoch 98: Train Loss=0.0121, Train Acc=100.00%, Test Loss=0.7263, Test Acc=78.57%

Epoch 99: Train Loss=0.1009, Train Acc=100.00%, Test Loss=0.8169, Test Acc=78.57%

Epoch 100: Train Loss=0.0056, Train Acc=100.00%, Test Loss=0.8021, Test Acc=78.57%



```
[284]: # torch.save(lora_model.state_dict(), "lora_resnet.pth")
```

3.6 Question 5 :

In order to better train our LORA weights, let's do some Data Augmentation https://en.wikipedia.org/wiki/Data_augmentation . Load some alteration of the data from the `torchvision.transforms` module and incorporate them in your training pipeline.

Intermediate question : Check CutMix (https://pytorch.org/vision/stable/auto_examples/transforms/plot_cutmix.html) and explain it with a small drawing cutmix.png.

Provide one file : (https://pytorch.org/tutorials/beginner/saving_loading_models.html) - a daug_resnet.pth file containing the weight of the ResNet18 after DAUG (! It has to be of the class ResNet so you have to merge LoRA weights with the ResNet18 weights)

```
[293]: ## Data Augmentation
from torchvision.transforms.v2 import CutMix
from torch.utils.data.dataloader import default_collate
```

```
[294]: train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3),
    transforms.ToTensor(),
])

test_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
])
```

```
[295]: train_dataset = datasets.ImageFolder(
    root=dir_path + "north_dataset_sample",
    transform=train_transform
)

test_dataset = datasets.ImageFolder(
    root=dir_path + "north_dataset_test",
    transform=test_transform
)

# Instancier CutMix, puis l'appliquer
# On suppose 2 classes, alpha=1.0
cutmix_transform = CutMix(num_classes=2, alpha=1.0)

def cutmix_collate_fn(batch):
    """
    Le DataLoader fournit un batch (liste) de (image, label).
    On fait d'abord un collate standard (stack en batch),
    puis on applique CutMix sur ce batch complet.
    """
    images, labels = default_collate(batch) # => (B, C, H, W), (B,...)
    images, labels = cutmix_transform(images, labels)
```

```

    return images, labels

train_loader = DataLoader(
    train_dataset,
    batch_size=32,
    shuffle=True,
    collate_fn=cutmix_collate_fn # injection de CutMix
)

test_loader = DataLoader(
    test_dataset,
    batch_size=32,
    shuffle=False
)

```

```

[296]: params_to_train = [p for p in lora_model.parameters() if p.requires_grad]
optimizer = optim.Adam(params_to_train, lr=1e-3)
criterion = nn.CrossEntropyLoss()

train_test2(lora_model, train_loader, test_loader, optimizer, criterion,
    ↪ epochs=100)

```

```

Epoch 1: Train Loss=0.2863, Train Acc=92.86%, Test Loss=0.3204, Test Acc=85.71%
Epoch 2: Train Loss=0.2625, Train Acc=83.33%, Test Loss=0.4479, Test Acc=80.95%
Epoch 3: Train Loss=0.3046, Train Acc=83.33%, Test Loss=0.5120, Test Acc=78.57%
Epoch 4: Train Loss=0.1818, Train Acc=88.10%, Test Loss=0.4358, Test Acc=80.95%
Epoch 5: Train Loss=0.3283, Train Acc=85.71%, Test Loss=0.3851, Test Acc=85.71%
Epoch 6: Train Loss=0.2704, Train Acc=85.71%, Test Loss=0.3473, Test Acc=85.71%
Epoch 7: Train Loss=0.6469, Train Acc=85.71%, Test Loss=0.3210, Test Acc=88.10%
Epoch 8: Train Loss=0.1521, Train Acc=97.62%, Test Loss=0.2866, Test Acc=90.48%
Epoch 9: Train Loss=0.1790, Train Acc=88.10%, Test Loss=0.2907, Test Acc=88.10%
Epoch 10: Train Loss=0.3619, Train Acc=83.33%, Test Loss=0.2581, Test Acc=90.48%
Epoch 11: Train Loss=0.2308, Train Acc=88.10%, Test Loss=0.2579, Test Acc=90.48%
Epoch 12: Train Loss=0.3414, Train Acc=92.86%, Test Loss=0.3004, Test Acc=90.48%
Epoch 13: Train Loss=0.2502, Train Acc=92.86%, Test Loss=0.3144, Test Acc=88.10%
Epoch 14: Train Loss=0.1207, Train Acc=92.86%, Test Loss=0.2970, Test Acc=88.10%
Epoch 15: Train Loss=0.2100, Train Acc=90.48%, Test Loss=0.3052, Test Acc=88.10%
Epoch 16: Train Loss=0.7887, Train Acc=88.10%, Test Loss=0.3335, Test Acc=88.10%
Epoch 17: Train Loss=0.2134, Train Acc=92.86%, Test Loss=0.3936, Test Acc=83.33%
Epoch 18: Train Loss=0.1862, Train Acc=90.48%, Test Loss=0.4883, Test Acc=80.95%
Epoch 19: Train Loss=0.1300, Train Acc=95.24%, Test Loss=0.5494, Test Acc=76.19%
Epoch 20: Train Loss=0.4669, Train Acc=85.71%, Test Loss=0.5003, Test Acc=76.19%
Epoch 21: Train Loss=0.5539, Train Acc=73.81%, Test Loss=0.4315, Test Acc=78.57%
Epoch 22: Train Loss=0.1593, Train Acc=97.62%, Test Loss=0.4002, Test Acc=80.95%
Epoch 23: Train Loss=0.4577, Train Acc=80.95%, Test Loss=0.4062, Test Acc=80.95%
Epoch 24: Train Loss=0.1076, Train Acc=97.62%, Test Loss=0.4069, Test Acc=80.95%
Epoch 25: Train Loss=0.1671, Train Acc=95.24%, Test Loss=0.4106, Test Acc=80.95%
Epoch 26: Train Loss=0.1797, Train Acc=100.00%, Test Loss=0.3599, Test

```

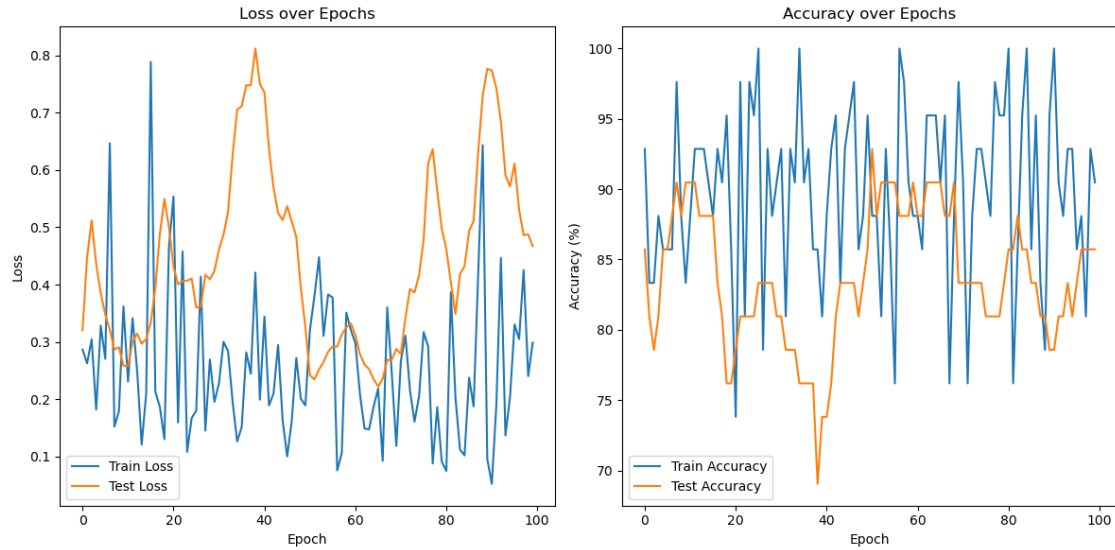
Acc=83.33%

Epoch 27: Train Loss=0.4138, Train Acc=78.57%, Test Loss=0.3614, Test Acc=83.33%
Epoch 28: Train Loss=0.1450, Train Acc=92.86%, Test Loss=0.4171, Test Acc=83.33%
Epoch 29: Train Loss=0.2692, Train Acc=88.10%, Test Loss=0.4094, Test Acc=83.33%
Epoch 30: Train Loss=0.1952, Train Acc=90.48%, Test Loss=0.4239, Test Acc=80.95%
Epoch 31: Train Loss=0.2276, Train Acc=92.86%, Test Loss=0.4610, Test Acc=80.95%
Epoch 32: Train Loss=0.3002, Train Acc=80.95%, Test Loss=0.4873, Test Acc=78.57%
Epoch 33: Train Loss=0.2837, Train Acc=92.86%, Test Loss=0.5275, Test Acc=78.57%
Epoch 34: Train Loss=0.1954, Train Acc=90.48%, Test Loss=0.6259, Test Acc=78.57%
Epoch 35: Train Loss=0.1260, Train Acc=100.00%, Test Loss=0.7056, Test
Acc=76.19%

Epoch 36: Train Loss=0.1513, Train Acc=90.48%, Test Loss=0.7113, Test Acc=76.19%
Epoch 37: Train Loss=0.2818, Train Acc=92.86%, Test Loss=0.7480, Test Acc=76.19%
Epoch 38: Train Loss=0.2443, Train Acc=85.71%, Test Loss=0.7481, Test Acc=76.19%
Epoch 39: Train Loss=0.4211, Train Acc=85.71%, Test Loss=0.8123, Test Acc=69.05%
Epoch 40: Train Loss=0.1990, Train Acc=80.95%, Test Loss=0.7504, Test Acc=73.81%
Epoch 41: Train Loss=0.3441, Train Acc=88.10%, Test Loss=0.7352, Test Acc=73.81%
Epoch 42: Train Loss=0.1890, Train Acc=92.86%, Test Loss=0.6366, Test Acc=76.19%
Epoch 43: Train Loss=0.2099, Train Acc=95.24%, Test Loss=0.5691, Test Acc=80.95%
Epoch 44: Train Loss=0.2946, Train Acc=83.33%, Test Loss=0.5249, Test Acc=83.33%
Epoch 45: Train Loss=0.1627, Train Acc=92.86%, Test Loss=0.5126, Test Acc=83.33%
Epoch 46: Train Loss=0.1000, Train Acc=95.24%, Test Loss=0.5370, Test Acc=83.33%
Epoch 47: Train Loss=0.1614, Train Acc=97.62%, Test Loss=0.5103, Test Acc=83.33%
Epoch 48: Train Loss=0.2719, Train Acc=85.71%, Test Loss=0.4834, Test Acc=80.95%
Epoch 49: Train Loss=0.2007, Train Acc=88.10%, Test Loss=0.3940, Test Acc=83.33%
Epoch 50: Train Loss=0.1891, Train Acc=95.24%, Test Loss=0.3259, Test Acc=85.71%
Epoch 51: Train Loss=0.3210, Train Acc=88.10%, Test Loss=0.2418, Test Acc=92.86%
Epoch 52: Train Loss=0.3803, Train Acc=88.10%, Test Loss=0.2345, Test Acc=88.10%
Epoch 53: Train Loss=0.4479, Train Acc=80.95%, Test Loss=0.2517, Test Acc=90.48%
Epoch 54: Train Loss=0.3106, Train Acc=92.86%, Test Loss=0.2648, Test Acc=90.48%
Epoch 55: Train Loss=0.3827, Train Acc=85.71%, Test Loss=0.2829, Test Acc=90.48%
Epoch 56: Train Loss=0.3772, Train Acc=76.19%, Test Loss=0.2919, Test Acc=90.48%
Epoch 57: Train Loss=0.0755, Train Acc=100.00%, Test Loss=0.2917, Test
Acc=88.10%

Epoch 58: Train Loss=0.1054, Train Acc=97.62%, Test Loss=0.3126, Test Acc=88.10%
Epoch 59: Train Loss=0.3510, Train Acc=90.48%, Test Loss=0.3254, Test Acc=88.10%
Epoch 60: Train Loss=0.3207, Train Acc=88.10%, Test Loss=0.3320, Test Acc=90.48%
Epoch 61: Train Loss=0.2976, Train Acc=88.10%, Test Loss=0.3117, Test Acc=88.10%
Epoch 62: Train Loss=0.2091, Train Acc=85.71%, Test Loss=0.2786, Test Acc=88.10%
Epoch 63: Train Loss=0.1489, Train Acc=95.24%, Test Loss=0.2603, Test Acc=90.48%
Epoch 64: Train Loss=0.1470, Train Acc=95.24%, Test Loss=0.2525, Test Acc=90.48%
Epoch 65: Train Loss=0.1866, Train Acc=95.24%, Test Loss=0.2350, Test Acc=90.48%
Epoch 66: Train Loss=0.2184, Train Acc=90.48%, Test Loss=0.2220, Test Acc=90.48%
Epoch 67: Train Loss=0.0922, Train Acc=95.24%, Test Loss=0.2353, Test Acc=88.10%
Epoch 68: Train Loss=0.3603, Train Acc=76.19%, Test Loss=0.2684, Test Acc=88.10%
Epoch 69: Train Loss=0.2450, Train Acc=88.10%, Test Loss=0.2689, Test Acc=90.48%
Epoch 70: Train Loss=0.1183, Train Acc=97.62%, Test Loss=0.2875, Test Acc=83.33%
Epoch 71: Train Loss=0.2665, Train Acc=90.48%, Test Loss=0.2788, Test Acc=83.33%

Epoch 72: Train Loss=0.3108, Train Acc=76.19%, Test Loss=0.3414, Test Acc=83.33%
 Epoch 73: Train Loss=0.2155, Train Acc=88.10%, Test Loss=0.3923, Test Acc=83.33%
 Epoch 74: Train Loss=0.1607, Train Acc=92.86%, Test Loss=0.3863, Test Acc=83.33%
 Epoch 75: Train Loss=0.2048, Train Acc=92.86%, Test Loss=0.4164, Test Acc=83.33%
 Epoch 76: Train Loss=0.3170, Train Acc=90.48%, Test Loss=0.4772, Test Acc=80.95%
 Epoch 77: Train Loss=0.2921, Train Acc=88.10%, Test Loss=0.6116, Test Acc=80.95%
 Epoch 78: Train Loss=0.0878, Train Acc=97.62%, Test Loss=0.6362, Test Acc=80.95%
 Epoch 79: Train Loss=0.1861, Train Acc=95.24%, Test Loss=0.5633, Test Acc=80.95%
 Epoch 80: Train Loss=0.0905, Train Acc=95.24%, Test Loss=0.4973, Test Acc=83.33%
 Epoch 81: Train Loss=0.0746, Train Acc=100.00%, Test Loss=0.4616, Test
 Acc=85.71%
 Epoch 82: Train Loss=0.3869, Train Acc=76.19%, Test Loss=0.4032, Test Acc=85.71%
 Epoch 83: Train Loss=0.2033, Train Acc=85.71%, Test Loss=0.3486, Test Acc=88.10%
 Epoch 84: Train Loss=0.1120, Train Acc=95.24%, Test Loss=0.4182, Test Acc=85.71%
 Epoch 85: Train Loss=0.1018, Train Acc=100.00%, Test Loss=0.4322, Test
 Acc=85.71%
 Epoch 86: Train Loss=0.2375, Train Acc=85.71%, Test Loss=0.4933, Test Acc=83.33%
 Epoch 87: Train Loss=0.1872, Train Acc=95.24%, Test Loss=0.5110, Test Acc=83.33%
 Epoch 88: Train Loss=0.4182, Train Acc=83.33%, Test Loss=0.6304, Test Acc=80.95%
 Epoch 89: Train Loss=0.6437, Train Acc=78.57%, Test Loss=0.7299, Test Acc=80.95%
 Epoch 90: Train Loss=0.0956, Train Acc=95.24%, Test Loss=0.7769, Test Acc=78.57%
 Epoch 91: Train Loss=0.0521, Train Acc=100.00%, Test Loss=0.7739, Test
 Acc=78.57%
 Epoch 92: Train Loss=0.1899, Train Acc=90.48%, Test Loss=0.7429, Test Acc=80.95%
 Epoch 93: Train Loss=0.4465, Train Acc=88.10%, Test Loss=0.6850, Test Acc=80.95%
 Epoch 94: Train Loss=0.1367, Train Acc=92.86%, Test Loss=0.5911, Test Acc=83.33%
 Epoch 95: Train Loss=0.2033, Train Acc=92.86%, Test Loss=0.5713, Test Acc=80.95%
 Epoch 96: Train Loss=0.3305, Train Acc=85.71%, Test Loss=0.6112, Test Acc=83.33%
 Epoch 97: Train Loss=0.3049, Train Acc=88.10%, Test Loss=0.5306, Test Acc=85.71%
 Epoch 98: Train Loss=0.4253, Train Acc=80.95%, Test Loss=0.4863, Test Acc=85.71%
 Epoch 99: Train Loss=0.2402, Train Acc=92.86%, Test Loss=0.4877, Test Acc=85.71%
 Epoch 100: Train Loss=0.2988, Train Acc=90.48%, Test Loss=0.4671, Test
 Acc=85.71%



```
[289]: # Fusion complète (toutes les couches LoRA -> merges)
for module in lora_model.modules():
    if hasattr(module, "merge_weights") and callable(module.merge_weights):
        module.merge_weights()

# torch.save(lora_model.state_dict(), "daug_resnet.pth")
```

3.7 Question 6 : (BONUS)

Do the best you can : improve performance on test set while keeping ResNet 18 architecture, or decrease the size of the model

Provide a file `final_model.pth` containing the weights of the final model and provide the class `FinalModel()` in the `utils.py` file.