Small data and deep learning

This pratical session proposes to study several techniques to improve training performance in the challenging context where few data and resources are available.

In [1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from tqdm import tqdm
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

Introduction

Assume we are in a context where few "gold" labeled data are available for training, say $\mathcal{X}_{\text{train}} \triangleq \{(x_n, y_n)\}_{n \leq N_{\text{train}}}$, where N_{train} is small. A large test set $\mathcal{X}_{\text{test}}$ is available. A large amount of unlabeled data, $\mathcal{X}_{\text{nolabel}}$, is available. We also assume that we have a limited computational budget (e.g., no GPUs).

For each question, write a commented *Code* or a complete answer as a *Markdown*. When the objective of a question is to report a CNN accuracy, please use the following format to report it, at the end of the question :

Model	Number of epochs	Train accuracy	Test accuracy
XXX	XXX	XXX	XXX

If applicable, please add the field corresponding to the **Accuracy on Full Data** as well as a link to the **Reference paper** you used to report those numbers. (You do not need to train a CNN on the full CIFAR10 dataset.)

In your final report, please keep the logs of each training procedure you used. We will only run this jupyter if we have some doubts on your implementation.

The total file sizes should be reasonable (feasible with 2MB only!). You will be asked to hand in the notebook, together with any necessary files required to run it if any.

To run your experiments, you can use the same local installation as for previous TPs, or otherwise https://colab.research.google.com/.

Training set creation

Question 1 (2 points) : Propose a dataloader that will only use the first 100 samples of the CIFAR-10 training set.

Hint : You can modify the file located at https://github.com/pytorch/vision/blob/master/torchvision/datasets/cifar.py or use the information from https://pytorch.org/vision/stable/datasets.html

This is our dataset $\mathcal{X}_{\text{train}}$, it will be used until the end of this project. The remaining samples correspond to $\mathcal{X}_{\text{nolabel}}$. The testing set $\mathcal{X}_{\text{test}}$ corresponds to the whole testing set of CIFAR-10.

In [3]:
```
transform_train = transforms.Compose([
    transforms.ToTensor(),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
])

batch_size = 10

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, tr
nolabelset =  torch.utils.data.Subset(trainset, list(range(100, len(trainset))))
trainset = torch.utils.data.Subset(trainset, list(range(100)))
nolabel_dataloader = torch.utils.data.DataLoader(nolabelset, batch_size=batch_size,
train_dataloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuf

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, tr
test_dataloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffl

len(train_dataloader), len(nolabel_dataloader), len(test_dataloader)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-
10-python.tar.gz

Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Out[3]:
```
(10, 4990, 1000)
```

Testing procedure

Question 2 (1.5 points) : Explain why the evaluation of the training procedure is difficult. Propose several solutions.

Answer:

The evaluation of the training procedure is difficult since there is a risk of overfitting the training data after a few epochs of the training process. In particular, in this case we only have 100 training samples which make the risks for overfitting quite high.

A common method to evaluate the training procedure is to include a validation set in the training process on which we can evaluate the loss and/or accuracy of the model on samples that are out of the training set at each epoch. This way we can detect when the model starts to overfit. However, in this case, we only have 100 training samples so we might not be able to afford cutting a validation set in the training set. We could still use the testing set to evaluate the model at each (few) epoch(s) but considering the size of our testing set, this might be too costly to use the whole testing set.

Another solution could be to use part of the unlabeled data. We could first apply an unsupervised learning method (such as clustering) on the training data mixed with part of the unlabeled data to infer some labels on the unlabeled data. Then, during the training on the training set, we could compare the model's predictions with the infered labels to see if they match. The problem with this method is that we the unsupervised learning could be very inaccurate and mess up the evaluation process.

Raw approach: the baseline

In this section, the goal is to train a CNN on $\mathcal{X}_{\text{train}}$ and compare its performance with reported numbers from the litterature. You will have to re-use and/or design a standard classification pipeline. You should optimize your pipeline to obtain the best performances (image size, data augmentation by flip, ...).

The key ingredients for training a CNN are the batch size, as well as the learning rate schedule, i.e. how to decrease the learning rate as a function of the number of epochs. A possible schedule is to start the learning rate at 0.1 and decreasing it every 30 epochs by 10. In case of divergence, reduce the learning rate. A potential batch size could be 10, yet this can be cross-validated.

You can get some baselines accuracies in this paper : http://openaccess.thecvf.com/content_cvpr_2018/papers/Keshari_Learning_Structure_and_CVPR_201
Obviously, it is a different context for those researchers who had access to GPUs.

ResNet architectures

Question 3 (4 points) : Write a classification pipeline for $\mathcal{X}_{\text{train}}$, train from scratch and evaluate a *ResNet-18* architecture specific to CIFAR10 (details about the ResNet-18 model, originally designed for the ImageNet dataset, can be found here: https://arxiv.org/abs/1512.03385). Please report the accuracy obtained on the whole dataset as well as the reference paper/GitHub link.

Hint: You can re-use the following code: https://github.com/kuangliu/pytorch-cifar. During a training of 10 epochs, a batch size of 10 and a learning rate of 0.01, one obtains 40% accuracy on $\mathcal{X}_{\text{train}}$ (~2 minutes) and 20% accuracy on $\mathcal{X}_{\text{test}}$ (~5 minutes).

ResNet model

In [4]:
```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
```

```
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out


class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out

def ResNet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])
```

Training

In [5]:

```
model = ResNet18()
model.to(device)

epochs = 100
criterion = nn.CrossEntropyLoss()
lr = 0.1
optimizer = optim.Adam(model.parameters(), lr=lr)
lr_update_period = 30
milestones = list(range(0, epochs, lr_update_period))[1:]
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones, gamma=0.1)
```

In [6]:

```
def train(model, train_dataloader, criterion, optimizer, scheduler):
    total_train_losses = []
    total_train_accs = []

    for epoch in range(1,epochs+1):
        model.train()
        train_losses = []

        correct = 0
        for i, batch, in enumerate(tqdm(train_dataloader)):
            img_batch, lbl_batch = batch
            img_batch, lbl_batch = img_batch.to(device), lbl_batch.to(device)

            optimizer.zero_grad()
            outputs = model(img_batch)
            loss = criterion(torch.flatten(outputs, 1), lbl_batch)
            loss.backward()
            optimizer.step()

            train_losses.append(loss.item())
            correct += (torch.flatten(outputs, 1).argmax(1) == lbl_batch).float().su

        print('Train (epoch {}/{})\tLoss: {:.6f}\tAccuracy: {:.2f}%'.format(
            epoch, epochs, np.mean(train_losses), 100 * correct / len(trainset)))

        train_loss_mean = np.mean(train_losses)
        total_train_losses.append(train_loss_mean)
        train_acc_mean = correct / len(trainset)
        total_train_accs.append(train_acc_mean)

        scheduler.step()

    train_loss = total_train_losses[-1]
    train_acc = total_train_accs[-1]
    print("\nTrain loss: {:.6f}\tTrain accuracy: {:.2f}%".format(train_loss, train_a

    plt.plot(list(range(epochs+1))[1:], total_train_losses)
    plt.title('Train loss evolution')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')

    plt.show()

    plt.plot(list(range(epochs+1))[1:], total_train_accs)
    plt.title('Train accuracy evolution')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')

    plt.show()

train(model, train_dataloader, criterion, optimizer, scheduler)
```

```
100%|██████████| 10/10 [00:00<00:00, 20.57it/s]
Train (epoch 1/100)     Loss: 14.173867 Accuracy: 10.00%
100%|██████████| 10/10 [00:00<00:00, 56.91it/s]
Train (epoch 2/100)     Loss: 2.647141  Accuracy: 13.00%
100%|██████████| 10/10 [00:00<00:00, 55.65it/s]
Train (epoch 3/100)     Loss: 2.481507  Accuracy: 14.00%
100%|██████████| 10/10 [00:00<00:00, 57.98it/s]
Train (epoch 4/100)     Loss: 2.271270  Accuracy: 11.00%
100%|██████████| 10/10 [00:00<00:00, 55.14it/s]
Train (epoch 5/100)     Loss: 2.282340  Accuracy: 15.00%
100%|██████████| 10/10 [00:00<00:00, 57.99it/s]
```

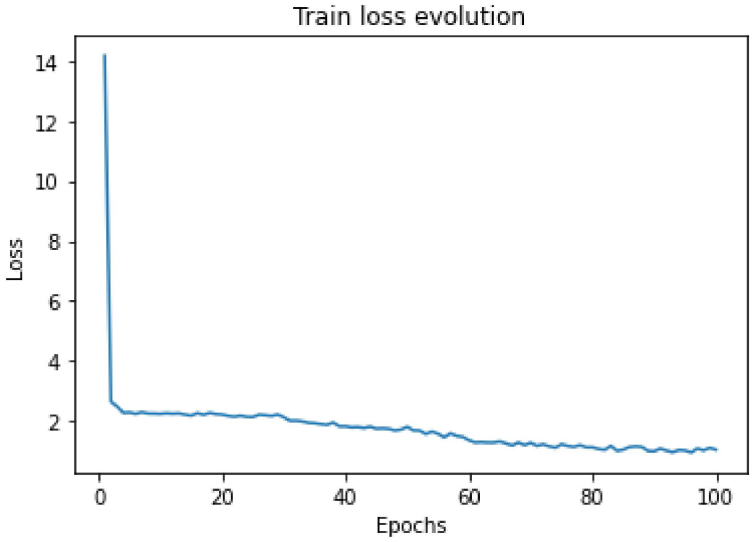
```
Train (epoch 6/100)     Loss: 2.240300  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 58.67it/s]
Train (epoch 7/100)     Loss: 2.291780  Accuracy: 15.00%
100%|██████████| 10/10 [00:00<00:00, 56.64it/s]
Train (epoch 8/100)     Loss: 2.250262  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 54.21it/s]
Train (epoch 9/100)     Loss: 2.247396  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 56.91it/s]
Train (epoch 10/100)    Loss: 2.238914  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 56.68it/s]
Train (epoch 11/100)    Loss: 2.258385  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 56.70it/s]
Train (epoch 12/100)    Loss: 2.242633  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 57.51it/s]
Train (epoch 13/100)    Loss: 2.253733  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 57.62it/s]
Train (epoch 14/100)    Loss: 2.212862  Accuracy: 14.00%
100%|██████████| 10/10 [00:00<00:00, 59.24it/s]
Train (epoch 15/100)    Loss: 2.176782  Accuracy: 20.00%
100%|██████████| 10/10 [00:00<00:00, 52.55it/s]
Train (epoch 16/100)    Loss: 2.263088  Accuracy: 15.00%
100%|██████████| 10/10 [00:00<00:00, 58.39it/s]
Train (epoch 17/100)    Loss: 2.207663  Accuracy: 13.00%
100%|██████████| 10/10 [00:00<00:00, 55.69it/s]
Train (epoch 18/100)    Loss: 2.268725  Accuracy: 15.00%
100%|██████████| 10/10 [00:00<00:00, 57.45it/s]
Train (epoch 19/100)    Loss: 2.230460  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 57.53it/s]
Train (epoch 20/100)    Loss: 2.214703  Accuracy: 17.00%
100%|██████████| 10/10 [00:00<00:00, 56.94it/s]
Train (epoch 21/100)    Loss: 2.169014  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 56.97it/s]
Train (epoch 22/100)    Loss: 2.143315  Accuracy: 23.00%
100%|██████████| 10/10 [00:00<00:00, 58.39it/s]
Train (epoch 23/100)    Loss: 2.176824  Accuracy: 19.00%
100%|██████████| 10/10 [00:00<00:00, 58.89it/s]
Train (epoch 24/100)    Loss: 2.136893  Accuracy: 22.00%
100%|██████████| 10/10 [00:00<00:00, 59.27it/s]
Train (epoch 25/100)    Loss: 2.132151  Accuracy: 24.00%
100%|██████████| 10/10 [00:00<00:00, 58.28it/s]
Train (epoch 26/100)    Loss: 2.211286  Accuracy: 23.00%
100%|██████████| 10/10 [00:00<00:00, 58.21it/s]
Train (epoch 27/100)    Loss: 2.191934  Accuracy: 19.00%
100%|██████████| 10/10 [00:00<00:00, 58.00it/s]
Train (epoch 28/100)    Loss: 2.167705  Accuracy: 25.00%
100%|██████████| 10/10 [00:00<00:00, 57.61it/s]
Train (epoch 29/100)    Loss: 2.215671  Accuracy: 18.00%
100%|██████████| 10/10 [00:00<00:00, 56.21it/s]
Train (epoch 30/100)    Loss: 2.124735  Accuracy: 22.00%
100%|██████████| 10/10 [00:00<00:00, 56.54it/s]
Train (epoch 31/100)    Loss: 2.007487  Accuracy: 23.00%
100%|██████████| 10/10 [00:00<00:00, 56.90it/s]
Train (epoch 32/100)    Loss: 2.009177  Accuracy: 26.00%
100%|██████████| 10/10 [00:00<00:00, 54.17it/s]
Train (epoch 33/100)    Loss: 1.986536  Accuracy: 25.00%
100%|██████████| 10/10 [00:00<00:00, 57.57it/s]
Train (epoch 34/100)    Loss: 1.939341  Accuracy: 31.00%
100%|██████████| 10/10 [00:00<00:00, 56.43it/s]
Train (epoch 35/100)    Loss: 1.925496  Accuracy: 29.00%
```

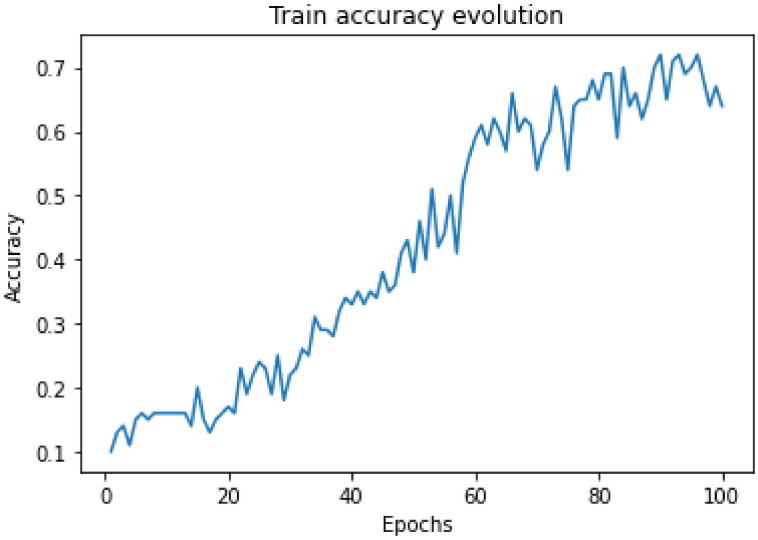
```
100%|██████████| 10/10 [00:00<00:00, 58.01it/s]
Train (epoch 36/100)    Loss: 1.894661  Accuracy: 29.00%
100%|██████████| 10/10 [00:00<00:00, 56.98it/s]
Train (epoch 37/100)    Loss: 1.871072  Accuracy: 28.00%
100%|██████████| 10/10 [00:00<00:00, 57.67it/s]
Train (epoch 38/100)    Loss: 1.949564  Accuracy: 32.00%
100%|██████████| 10/10 [00:00<00:00, 57.42it/s]
Train (epoch 39/100)    Loss: 1.819986  Accuracy: 34.00%
100%|██████████| 10/10 [00:00<00:00, 56.32it/s]
Train (epoch 40/100)    Loss: 1.820623  Accuracy: 33.00%
100%|██████████| 10/10 [00:00<00:00, 55.84it/s]
Train (epoch 41/100)    Loss: 1.787356  Accuracy: 35.00%
100%|██████████| 10/10 [00:00<00:00, 56.54it/s]
Train (epoch 42/100)    Loss: 1.797617  Accuracy: 33.00%
100%|██████████| 10/10 [00:00<00:00, 59.12it/s]
Train (epoch 43/100)    Loss: 1.765094  Accuracy: 35.00%
100%|██████████| 10/10 [00:00<00:00, 57.57it/s]
Train (epoch 44/100)    Loss: 1.810713  Accuracy: 34.00%
100%|██████████| 10/10 [00:00<00:00, 58.35it/s]
Train (epoch 45/100)    Loss: 1.743226  Accuracy: 38.00%
100%|██████████| 10/10 [00:00<00:00, 57.13it/s]
Train (epoch 46/100)    Loss: 1.752311  Accuracy: 35.00%
100%|██████████| 10/10 [00:00<00:00, 58.75it/s]
Train (epoch 47/100)    Loss: 1.739355  Accuracy: 36.00%
100%|██████████| 10/10 [00:00<00:00, 59.06it/s]
Train (epoch 48/100)    Loss: 1.680105  Accuracy: 41.00%
100%|██████████| 10/10 [00:00<00:00, 56.95it/s]
Train (epoch 49/100)    Loss: 1.715570  Accuracy: 43.00%
100%|██████████| 10/10 [00:00<00:00, 57.78it/s]
Train (epoch 50/100)    Loss: 1.801788  Accuracy: 38.00%
100%|██████████| 10/10 [00:00<00:00, 58.14it/s]
Train (epoch 51/100)    Loss: 1.684847  Accuracy: 46.00%
100%|██████████| 10/10 [00:00<00:00, 58.67it/s]
Train (epoch 52/100)    Loss: 1.680698  Accuracy: 40.00%
100%|██████████| 10/10 [00:00<00:00, 58.96it/s]
Train (epoch 53/100)    Loss: 1.563620  Accuracy: 51.00%
100%|██████████| 10/10 [00:00<00:00, 58.79it/s]
Train (epoch 54/100)    Loss: 1.645716  Accuracy: 42.00%
100%|██████████| 10/10 [00:00<00:00, 58.06it/s]
Train (epoch 55/100)    Loss: 1.573211  Accuracy: 44.00%
100%|██████████| 10/10 [00:00<00:00, 57.92it/s]
Train (epoch 56/100)    Loss: 1.453446  Accuracy: 50.00%
100%|██████████| 10/10 [00:00<00:00, 57.25it/s]
Train (epoch 57/100)    Loss: 1.588128  Accuracy: 41.00%
100%|██████████| 10/10 [00:00<00:00, 55.07it/s]
Train (epoch 58/100)    Loss: 1.500733  Accuracy: 52.00%
100%|██████████| 10/10 [00:00<00:00, 56.63it/s]
Train (epoch 59/100)    Loss: 1.466129  Accuracy: 56.00%
100%|██████████| 10/10 [00:00<00:00, 55.61it/s]
Train (epoch 60/100)    Loss: 1.356649  Accuracy: 59.00%
100%|██████████| 10/10 [00:00<00:00, 56.79it/s]
Train (epoch 61/100)    Loss: 1.281722  Accuracy: 61.00%
100%|██████████| 10/10 [00:00<00:00, 55.28it/s]
Train (epoch 62/100)    Loss: 1.295134  Accuracy: 58.00%
100%|██████████| 10/10 [00:00<00:00, 56.95it/s]
Train (epoch 63/100)    Loss: 1.281352  Accuracy: 62.00%
100%|██████████| 10/10 [00:00<00:00, 58.88it/s]
Train (epoch 64/100)    Loss: 1.280813  Accuracy: 60.00%
100%|██████████| 10/10 [00:00<00:00, 58.05it/s]
Train (epoch 65/100)    Loss: 1.316476  Accuracy: 57.00%
```

```
100%|██████████| 10/10 [00:00<00:00, 57.91it/s]
Train (epoch 66/100)    Loss: 1.251877  Accuracy: 66.00%
100%|██████████| 10/10 [00:00<00:00, 58.61it/s]
Train (epoch 67/100)    Loss: 1.182145  Accuracy: 60.00%
100%|██████████| 10/10 [00:00<00:00, 58.48it/s]
Train (epoch 68/100)    Loss: 1.279623  Accuracy: 62.00%
100%|██████████| 10/10 [00:00<00:00, 58.17it/s]
Train (epoch 69/100)    Loss: 1.197192  Accuracy: 61.00%
100%|██████████| 10/10 [00:00<00:00, 57.42it/s]
Train (epoch 70/100)    Loss: 1.270977  Accuracy: 54.00%
100%|██████████| 10/10 [00:00<00:00, 56.06it/s]
Train (epoch 71/100)    Loss: 1.171638  Accuracy: 58.00%
100%|██████████| 10/10 [00:00<00:00, 56.35it/s]
Train (epoch 72/100)    Loss: 1.220926  Accuracy: 60.00%
100%|██████████| 10/10 [00:00<00:00, 55.60it/s]
Train (epoch 73/100)    Loss: 1.154077  Accuracy: 67.00%
100%|██████████| 10/10 [00:00<00:00, 57.84it/s]
Train (epoch 74/100)    Loss: 1.107821  Accuracy: 62.00%
100%|██████████| 10/10 [00:00<00:00, 56.74it/s]
Train (epoch 75/100)    Loss: 1.224511  Accuracy: 54.00%
100%|██████████| 10/10 [00:00<00:00, 58.37it/s]
Train (epoch 76/100)    Loss: 1.164805  Accuracy: 64.00%
100%|██████████| 10/10 [00:00<00:00, 54.15it/s]
Train (epoch 77/100)    Loss: 1.136278  Accuracy: 65.00%
100%|██████████| 10/10 [00:00<00:00, 58.47it/s]
Train (epoch 78/100)    Loss: 1.193771  Accuracy: 65.00%
100%|██████████| 10/10 [00:00<00:00, 58.21it/s]
Train (epoch 79/100)    Loss: 1.120062  Accuracy: 68.00%
100%|██████████| 10/10 [00:00<00:00, 58.44it/s]
Train (epoch 80/100)    Loss: 1.122673  Accuracy: 65.00%
100%|██████████| 10/10 [00:00<00:00, 58.26it/s]
Train (epoch 81/100)    Loss: 1.063897  Accuracy: 69.00%
100%|██████████| 10/10 [00:00<00:00, 56.51it/s]
Train (epoch 82/100)    Loss: 1.035164  Accuracy: 69.00%
100%|██████████| 10/10 [00:00<00:00, 57.05it/s]
Train (epoch 83/100)    Loss: 1.170233  Accuracy: 59.00%
100%|██████████| 10/10 [00:00<00:00, 56.25it/s]
Train (epoch 84/100)    Loss: 1.001470  Accuracy: 70.00%
100%|██████████| 10/10 [00:00<00:00, 57.82it/s]
Train (epoch 85/100)    Loss: 1.043422  Accuracy: 64.00%
100%|██████████| 10/10 [00:00<00:00, 57.84it/s]
Train (epoch 86/100)    Loss: 1.130207  Accuracy: 66.00%
100%|██████████| 10/10 [00:00<00:00, 58.33it/s]
Train (epoch 87/100)    Loss: 1.151870  Accuracy: 62.00%
100%|██████████| 10/10 [00:00<00:00, 58.16it/s]
Train (epoch 88/100)    Loss: 1.134730  Accuracy: 65.00%
100%|██████████| 10/10 [00:00<00:00, 58.68it/s]
Train (epoch 89/100)    Loss: 1.000151  Accuracy: 70.00%
100%|██████████| 10/10 [00:00<00:00, 57.71it/s]
Train (epoch 90/100)    Loss: 0.987956  Accuracy: 72.00%
100%|██████████| 10/10 [00:00<00:00, 58.98it/s]
Train (epoch 91/100)    Loss: 1.078234  Accuracy: 65.00%
100%|██████████| 10/10 [00:00<00:00, 58.54it/s]
Train (epoch 92/100)    Loss: 1.012300  Accuracy: 71.00%
100%|██████████| 10/10 [00:00<00:00, 56.67it/s]
Train (epoch 93/100)    Loss: 0.955418  Accuracy: 72.00%
100%|██████████| 10/10 [00:00<00:00, 57.36it/s]
Train (epoch 94/100)    Loss: 1.030773  Accuracy: 69.00%
100%|██████████| 10/10 [00:00<00:00, 56.74it/s]
```

```
Train (epoch 95/100)    Loss: 1.014880  Accuracy: 70.00%
100%|██████████| 10/10 [00:00<00:00, 59.01it/s]
Train (epoch 96/100)    Loss: 0.949116  Accuracy: 72.00%
100%|██████████| 10/10 [00:00<00:00, 58.97it/s]
Train (epoch 97/100)    Loss: 1.080860  Accuracy: 68.00%
100%|██████████| 10/10 [00:00<00:00, 58.08it/s]
Train (epoch 98/100)    Loss: 1.017342  Accuracy: 64.00%
100%|██████████| 10/10 [00:00<00:00, 57.22it/s]
Train (epoch 99/100)    Loss: 1.100903  Accuracy: 67.00%
100%|██████████| 10/10 [00:00<00:00, 58.89it/s]
Train (epoch 100/100)   Loss: 1.043173  Accuracy: 64.00%

Train loss: 1.043173    Train accuracy: 64.00%
```





Testing

In [7]:

```python
def test(model, test_dataloader):
    model.eval()
    test_losses = []
    correct = 0

    for i, batch, in enumerate(tqdm(test_dataloader)):
        img_batch, lbl_batch = batch
        img_batch, lbl_batch = img_batch.to(device), lbl_batch.to(device)

        outputs = model(img_batch)
        loss = criterion(torch.flatten(outputs, 1), lbl_batch)
```

```
        test_losses.append(loss.item())
        correct += (torch.flatten(outputs, 1).argmax(1) == lbl_batch).float().sum().

    test_loss = np.mean(test_losses)
    test_acc = correct / len(testset)

    print("\nTest loss: {:.6f}\tTest accuracy: {:.2f}%".format(test_loss, 100 * test

test(model, test_dataloader)
```

```
100%|██████████| 1000/1000 [00:05<00:00, 187.07it/s]
Test loss: 2.650611     Test accuracy: 19.40%
```

Results

Model	Number of epochs	Train accuracy	Test accuracy
ResNet18	100	0.64	0.1940

Transfer learning

We propose to use pre-trained models on a classification task, in order to improve the results of our setting.

ImageNet features

Now, we will use a model pre-trained on ImageNet and see how well it performs on CIFAR. A list of ImageNet pre-trained models is available on : https://pytorch.org/vision/stable/models.html

Question 4 (3 points) : Pick a model from the list above, adapt it for CIFAR and retrain its final layer (or a block of layers, depending on the resources to which you have access to). Report its accuracy.

Training

In [18]:
```
model = torchvision.models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 10)
model.to(device)

epochs = 100
criterion = nn.CrossEntropyLoss()
lr = 0.001
optimizer = optim.Adam(model.parameters(), lr=lr)
lr_update_period = 30
milestones = list(range(0, epochs, lr_update_period))[1:]
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones, gamma=0.1)
```

In [19]:
```
train(model, train_dataloader, criterion, optimizer, scheduler)
```

```
100%|██████████| 10/10 [00:00<00:00, 71.15it/s]
Train (epoch 1/100)     Loss: 2.654333  Accuracy: 24.00%
100%|██████████| 10/10 [00:00<00:00, 70.61it/s]
Train (epoch 2/100)     Loss: 2.105064  Accuracy: 42.00%
100%|██████████| 10/10 [00:00<00:00, 70.90it/s]
Train (epoch 3/100)     Loss: 1.268868  Accuracy: 59.00%
```

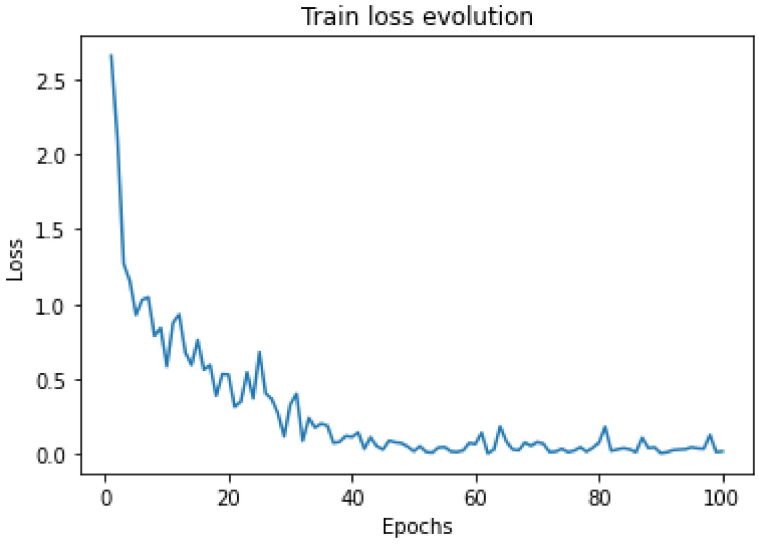
```
100%|██████████| 10/10 [00:00<00:00, 70.66it/s]
Train (epoch 4/100)     Loss: 1.155815  Accuracy: 62.00%
100%|██████████| 10/10 [00:00<00:00, 65.27it/s]
Train (epoch 5/100)     Loss: 0.929193  Accuracy: 70.00%
100%|██████████| 10/10 [00:00<00:00, 68.87it/s]
Train (epoch 6/100)     Loss: 1.029377  Accuracy: 71.00%
100%|██████████| 10/10 [00:00<00:00, 64.37it/s]
Train (epoch 7/100)     Loss: 1.046078  Accuracy: 71.00%
100%|██████████| 10/10 [00:00<00:00, 68.90it/s]
Train (epoch 8/100)     Loss: 0.789808  Accuracy: 80.00%
100%|██████████| 10/10 [00:00<00:00, 71.93it/s]
Train (epoch 9/100)     Loss: 0.842119  Accuracy: 78.00%
100%|██████████| 10/10 [00:00<00:00, 68.96it/s]
Train (epoch 10/100)    Loss: 0.585474  Accuracy: 81.00%
100%|██████████| 10/10 [00:00<00:00, 68.36it/s]
Train (epoch 11/100)    Loss: 0.875376  Accuracy: 74.00%
100%|██████████| 10/10 [00:00<00:00, 69.34it/s]
Train (epoch 12/100)    Loss: 0.931330  Accuracy: 79.00%
100%|██████████| 10/10 [00:00<00:00, 68.46it/s]
Train (epoch 13/100)    Loss: 0.679561  Accuracy: 81.00%
100%|██████████| 10/10 [00:00<00:00, 61.16it/s]
Train (epoch 14/100)    Loss: 0.594977  Accuracy: 79.00%
100%|██████████| 10/10 [00:00<00:00, 56.58it/s]
Train (epoch 15/100)    Loss: 0.762102  Accuracy: 72.00%
100%|██████████| 10/10 [00:00<00:00, 33.24it/s]
Train (epoch 16/100)    Loss: 0.564322  Accuracy: 81.00%
100%|██████████| 10/10 [00:00<00:00, 36.96it/s]
Train (epoch 17/100)    Loss: 0.594385  Accuracy: 85.00%
100%|██████████| 10/10 [00:00<00:00, 35.89it/s]
Train (epoch 18/100)    Loss: 0.390600  Accuracy: 91.00%
100%|██████████| 10/10 [00:00<00:00, 31.95it/s]
Train (epoch 19/100)    Loss: 0.535006  Accuracy: 87.00%
100%|██████████| 10/10 [00:00<00:00, 47.54it/s]
Train (epoch 20/100)    Loss: 0.530476  Accuracy: 82.00%
100%|██████████| 10/10 [00:00<00:00, 35.02it/s]
Train (epoch 21/100)    Loss: 0.317350  Accuracy: 92.00%
100%|██████████| 10/10 [00:00<00:00, 37.81it/s]
Train (epoch 22/100)    Loss: 0.351545  Accuracy: 85.00%
100%|██████████| 10/10 [00:00<00:00, 25.52it/s]
Train (epoch 23/100)    Loss: 0.546726  Accuracy: 85.00%
100%|██████████| 10/10 [00:00<00:00, 34.64it/s]
Train (epoch 24/100)    Loss: 0.373854  Accuracy: 91.00%
100%|██████████| 10/10 [00:00<00:00, 34.76it/s]
Train (epoch 25/100)    Loss: 0.681156  Accuracy: 84.00%
100%|██████████| 10/10 [00:00<00:00, 28.85it/s]
Train (epoch 26/100)    Loss: 0.406504  Accuracy: 84.00%
100%|██████████| 10/10 [00:00<00:00, 31.24it/s]
Train (epoch 27/100)    Loss: 0.368460  Accuracy: 89.00%
100%|██████████| 10/10 [00:00<00:00, 37.73it/s]
Train (epoch 28/100)    Loss: 0.271760  Accuracy: 90.00%
100%|██████████| 10/10 [00:00<00:00, 36.20it/s]
Train (epoch 29/100)    Loss: 0.119364  Accuracy: 96.00%
100%|██████████| 10/10 [00:00<00:00, 35.77it/s]
Train (epoch 30/100)    Loss: 0.333305  Accuracy: 89.00%
100%|██████████| 10/10 [00:00<00:00, 30.51it/s]
Train (epoch 31/100)    Loss: 0.403329  Accuracy: 86.00%
100%|██████████| 10/10 [00:00<00:00, 28.54it/s]
Train (epoch 32/100)    Loss: 0.090575  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 25.50it/s]
```

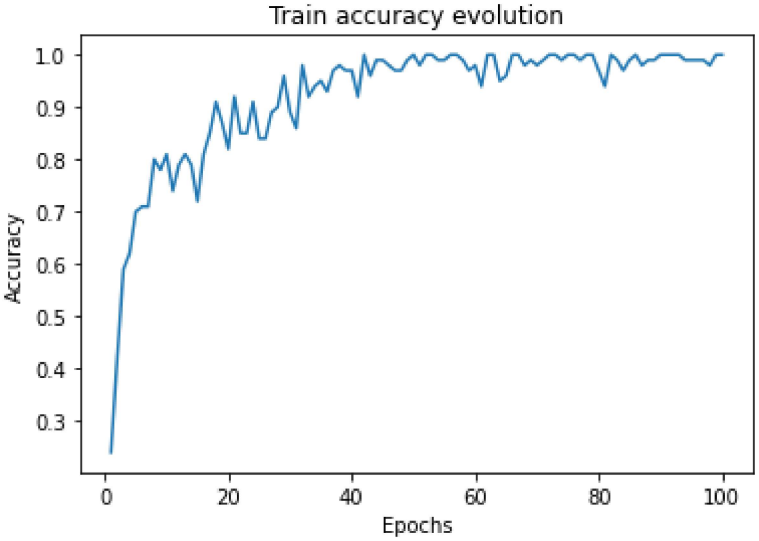
```
Train (epoch 33/100)    Loss: 0.243140  Accuracy: 92.00%
100%|██████████| 10/10 [00:00<00:00, 27.63it/s]
Train (epoch 34/100)    Loss: 0.178384  Accuracy: 94.00%
100%|██████████| 10/10 [00:00<00:00, 29.38it/s]
Train (epoch 35/100)    Loss: 0.204884  Accuracy: 95.00%
100%|██████████| 10/10 [00:00<00:00, 35.48it/s]
Train (epoch 36/100)    Loss: 0.192377  Accuracy: 93.00%
100%|██████████| 10/10 [00:00<00:00, 34.94it/s]
Train (epoch 37/100)    Loss: 0.076209  Accuracy: 97.00%
100%|██████████| 10/10 [00:00<00:00, 38.69it/s]
Train (epoch 38/100)    Loss: 0.084062  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 42.86it/s]
Train (epoch 39/100)    Loss: 0.122668  Accuracy: 97.00%
100%|██████████| 10/10 [00:00<00:00, 35.74it/s]
Train (epoch 40/100)    Loss: 0.115955  Accuracy: 97.00%
100%|██████████| 10/10 [00:00<00:00, 22.27it/s]
Train (epoch 41/100)    Loss: 0.146371  Accuracy: 92.00%
100%|██████████| 10/10 [00:00<00:00, 31.08it/s]
Train (epoch 42/100)    Loss: 0.037030  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 60.58it/s]
Train (epoch 43/100)    Loss: 0.113854  Accuracy: 96.00%
100%|██████████| 10/10 [00:00<00:00, 72.87it/s]
Train (epoch 44/100)    Loss: 0.055109  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 68.13it/s]
Train (epoch 45/100)    Loss: 0.033256  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 70.06it/s]
Train (epoch 46/100)    Loss: 0.091546  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 67.78it/s]
Train (epoch 47/100)    Loss: 0.080521  Accuracy: 97.00%
100%|██████████| 10/10 [00:00<00:00, 65.70it/s]
Train (epoch 48/100)    Loss: 0.075182  Accuracy: 97.00%
100%|██████████| 10/10 [00:00<00:00, 70.12it/s]
Train (epoch 49/100)    Loss: 0.052392  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 70.41it/s]
Train (epoch 50/100)    Loss: 0.021225  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 68.09it/s]
Train (epoch 51/100)    Loss: 0.054994  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 69.05it/s]
Train (epoch 52/100)    Loss: 0.016674  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 68.70it/s]
Train (epoch 53/100)    Loss: 0.011970  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 70.36it/s]
Train (epoch 54/100)    Loss: 0.045598  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 72.97it/s]
Train (epoch 55/100)    Loss: 0.049664  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 68.30it/s]
Train (epoch 56/100)    Loss: 0.019863  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 74.21it/s]
Train (epoch 57/100)    Loss: 0.016603  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 68.24it/s]
Train (epoch 58/100)    Loss: 0.026326  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 69.70it/s]
Train (epoch 59/100)    Loss: 0.076123  Accuracy: 97.00%
100%|██████████| 10/10 [00:00<00:00, 69.46it/s]
Train (epoch 60/100)    Loss: 0.069347  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 67.01it/s]
Train (epoch 61/100)    Loss: 0.144144  Accuracy: 94.00%
100%|██████████| 10/10 [00:00<00:00, 67.35it/s]
Train (epoch 62/100)    Loss: 0.007285  Accuracy: 100.00%
```

```
100%|██████████| 10/10 [00:00<00:00, 68.55it/s]
Train (epoch 63/100)    Loss: 0.032019  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 64.51it/s]
Train (epoch 64/100)    Loss: 0.186206  Accuracy: 95.00%
100%|██████████| 10/10 [00:00<00:00, 71.59it/s]
Train (epoch 65/100)    Loss: 0.086446  Accuracy: 96.00%
100%|██████████| 10/10 [00:00<00:00, 68.35it/s]
Train (epoch 66/100)    Loss: 0.033519  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 70.66it/s]
Train (epoch 67/100)    Loss: 0.027742  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 71.32it/s]
Train (epoch 68/100)    Loss: 0.078759  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 73.50it/s]
Train (epoch 69/100)    Loss: 0.057207  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 75.39it/s]
Train (epoch 70/100)    Loss: 0.082496  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 68.94it/s]
Train (epoch 71/100)    Loss: 0.070863  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 68.83it/s]
Train (epoch 72/100)    Loss: 0.016160  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 66.12it/s]
Train (epoch 73/100)    Loss: 0.018946  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 70.07it/s]
Train (epoch 74/100)    Loss: 0.037918  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 69.77it/s]
Train (epoch 75/100)    Loss: 0.014793  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 67.54it/s]
Train (epoch 76/100)    Loss: 0.025603  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 61.79it/s]
Train (epoch 77/100)    Loss: 0.047919  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 71.72it/s]
Train (epoch 78/100)    Loss: 0.018200  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 75.68it/s]
Train (epoch 79/100)    Loss: 0.044535  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 72.38it/s]
Train (epoch 80/100)    Loss: 0.077692  Accuracy: 97.00%
100%|██████████| 10/10 [00:00<00:00, 74.23it/s]
Train (epoch 81/100)    Loss: 0.183003  Accuracy: 94.00%
100%|██████████| 10/10 [00:00<00:00, 72.80it/s]
Train (epoch 82/100)    Loss: 0.024014  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 74.65it/s]
Train (epoch 83/100)    Loss: 0.033373  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 68.38it/s]
Train (epoch 84/100)    Loss: 0.042100  Accuracy: 97.00%
100%|██████████| 10/10 [00:00<00:00, 67.83it/s]
Train (epoch 85/100)    Loss: 0.033795  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 72.17it/s]
Train (epoch 86/100)    Loss: 0.014657  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 71.42it/s]
Train (epoch 87/100)    Loss: 0.110096  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 69.25it/s]
Train (epoch 88/100)    Loss: 0.043062  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 69.99it/s]
Train (epoch 89/100)    Loss: 0.047587  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 68.98it/s]
Train (epoch 90/100)    Loss: 0.008716  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 72.40it/s]
Train (epoch 91/100)    Loss: 0.014606  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 72.53it/s]
Train (epoch 92/100)    Loss: 0.028750  Accuracy: 100.00%
```

```
100%|██████████| 10/10 [00:00<00:00, 67.11it/s]
Train (epoch 93/100)    Loss: 0.032452  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 68.52it/s]
Train (epoch 94/100)    Loss: 0.034519  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 65.85it/s]
Train (epoch 95/100)    Loss: 0.047753  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 70.55it/s]
Train (epoch 96/100)    Loss: 0.040617  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 65.90it/s]
Train (epoch 97/100)    Loss: 0.037906  Accuracy: 99.00%
100%|██████████| 10/10 [00:00<00:00, 65.87it/s]
Train (epoch 98/100)    Loss: 0.129487  Accuracy: 98.00%
100%|██████████| 10/10 [00:00<00:00, 68.00it/s]
Train (epoch 99/100)    Loss: 0.016416  Accuracy: 100.00%
100%|██████████| 10/10 [00:00<00:00, 67.75it/s]
Train (epoch 100/100)   Loss: 0.020519  Accuracy: 100.00%

Train loss: 0.020519    Train accuracy: 100.00%
```





Testing

In [20]:
```
test(model, test_dataloader)
```

```
100%|██████████| 1000/1000 [00:05<00:00, 189.96it/s]
Test loss: 2.795136     Test accuracy: 29.82%
```

Results

Model	Number of epochs	Train accuracy	Test accuracy
ResNet18	100	0.64	0.1940
ResNet18 (transfer learning)	100	1.00	0.2982

Incorporating priors

Geometrical priors are appealing for image classification tasks. A color image x can be seen as a function: $\mathbb{S} \to \mathbb{R}^3$, where $\mathbb{S} \subset \mathbb{R}^2$ is the image support. Let us consider transformations $\mathcal{T}$ of possible inputs x. For instance, if an image had infinite support, a translation $\mathcal{T}_a$ of an image x by a shift a would lead to a new infinite-support image $\mathcal{T}_a(x)$, described at each pixel u by :

$$\forall u, \mathcal{T}_a(x)(u) = x(u-a)\,.$$

Question 5 (1.5 points) : Explain the issues when dealing with translations, rotations, scaling effects, color changes on 32×32 images. Propose several ideas to tackle them.

Answer:

The problems with transformations such as translations, rotations, scaling effects or color changes on 32×32 images is that these transformations introduce some information loss while the original image does not carry much information to begin with. If we use these transformations, the image might be too distorted and the characteristics or the features in the image might not be usable for the model to learn.

To fix this, we could limit ourselves to small translations (or rotations, scaling effects, color changes) to make sure most of the charcteristics of the image are preserved. Otherwise, we could increase the image size using some interpolation method.

Data augmentations

Question 6 (3 points) : Propose a set of geometric transformations beyond translation, and incorporate them in your training pipeline. Train the model of the **Question 3** with them and report the accuracies. You can use tools from https://pytorch.org/vision/stable/transforms.html

In [11]:
```
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

batch_size = 10

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, tr
trainset = torch.utils.data.Subset(trainset, list(range(100)))
train_dataloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuf
```

```
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, tr
test_dataloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffl

len(train_dataloader), len(test_dataloader)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

Out[11]: (10, 1000)

Training

In [12]:
```
model = ResNet18()
model.to(device)

epochs = 100
criterion = nn.CrossEntropyLoss()
lr = 0.1
optimizer = optim.Adam(model.parameters(), lr=lr)
lr_update_period = 30
milestones = list(range(0, epochs, lr_update_period))[1:]
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones, gamma=0.1)
```

In [13]:
```
train(model, train_dataloader, criterion, optimizer, scheduler)
```

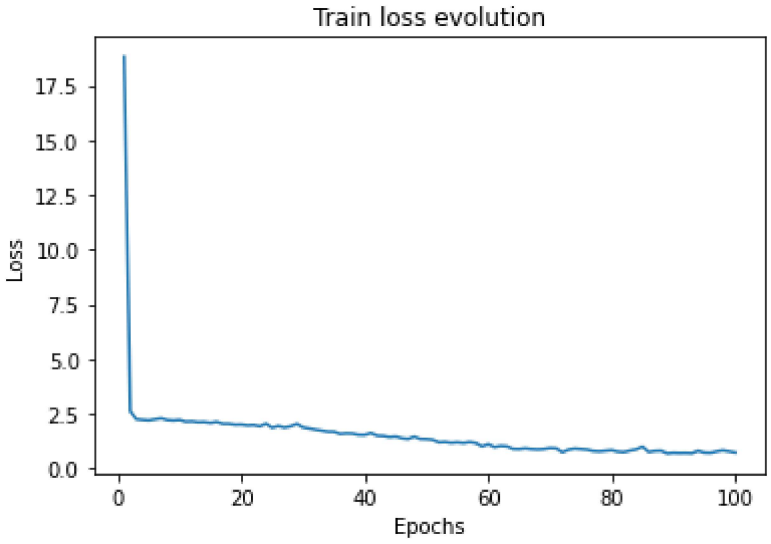
```
100%|██████████| 10/10 [00:00<00:00, 59.71it/s]
Train (epoch 1/100)     Loss: 18.809826 Accuracy: 10.00%
100%|██████████| 10/10 [00:00<00:00, 59.04it/s]
Train (epoch 2/100)     Loss: 2.603609  Accuracy: 10.00%
100%|██████████| 10/10 [00:00<00:00, 58.40it/s]
Train (epoch 3/100)     Loss: 2.242220  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 57.67it/s]
Train (epoch 4/100)     Loss: 2.215702  Accuracy: 10.00%
100%|██████████| 10/10 [00:00<00:00, 57.46it/s]
Train (epoch 5/100)     Loss: 2.190791  Accuracy: 13.00%
100%|██████████| 10/10 [00:00<00:00, 57.59it/s]
Train (epoch 6/100)     Loss: 2.240300  Accuracy: 14.00%
100%|██████████| 10/10 [00:00<00:00, 59.23it/s]
Train (epoch 7/100)     Loss: 2.286387  Accuracy: 11.00%
100%|██████████| 10/10 [00:00<00:00, 58.89it/s]
Train (epoch 8/100)     Loss: 2.207250  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 57.69it/s]
Train (epoch 9/100)     Loss: 2.186592  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 57.58it/s]
Train (epoch 10/100)    Loss: 2.211465  Accuracy: 11.00%
100%|██████████| 10/10 [00:00<00:00, 58.85it/s]
Train (epoch 11/100)    Loss: 2.129247  Accuracy: 16.00%
100%|██████████| 10/10 [00:00<00:00, 59.93it/s]
Train (epoch 12/100)    Loss: 2.140280  Accuracy: 23.00%
100%|██████████| 10/10 [00:00<00:00, 57.62it/s]
Train (epoch 13/100)    Loss: 2.104954  Accuracy: 24.00%
100%|██████████| 10/10 [00:00<00:00, 58.30it/s]
Train (epoch 14/100)    Loss: 2.112455  Accuracy: 20.00%
100%|██████████| 10/10 [00:00<00:00, 59.29it/s]
Train (epoch 15/100)    Loss: 2.072767  Accuracy: 23.00%
100%|██████████| 10/10 [00:00<00:00, 58.22it/s]
Train (epoch 16/100)    Loss: 2.119914  Accuracy: 22.00%
100%|██████████| 10/10 [00:00<00:00, 59.58it/s]
```

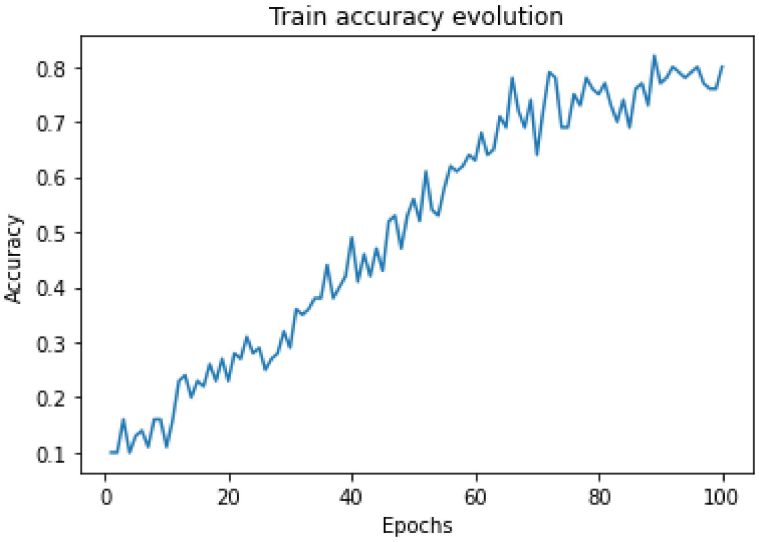
```
Train (epoch 17/100)    Loss: 2.032615  Accuracy: 26.00%
100%|██████████| 10/10 [00:00<00:00, 58.34it/s]
Train (epoch 18/100)    Loss: 2.033696  Accuracy: 23.00%
100%|██████████| 10/10 [00:00<00:00, 58.02it/s]
Train (epoch 19/100)    Loss: 1.990460  Accuracy: 27.00%
100%|██████████| 10/10 [00:00<00:00, 59.65it/s]
Train (epoch 20/100)    Loss: 1.999297  Accuracy: 23.00%
100%|██████████| 10/10 [00:00<00:00, 57.27it/s]
Train (epoch 21/100)    Loss: 1.960430  Accuracy: 28.00%
100%|██████████| 10/10 [00:00<00:00, 59.66it/s]
Train (epoch 22/100)    Loss: 1.976060  Accuracy: 27.00%
100%|██████████| 10/10 [00:00<00:00, 58.40it/s]
Train (epoch 23/100)    Loss: 1.919089  Accuracy: 31.00%
100%|██████████| 10/10 [00:00<00:00, 59.18it/s]
Train (epoch 24/100)    Loss: 2.038855  Accuracy: 28.00%
100%|██████████| 10/10 [00:00<00:00, 58.85it/s]
Train (epoch 25/100)    Loss: 1.855110  Accuracy: 29.00%
100%|██████████| 10/10 [00:00<00:00, 58.60it/s]
Train (epoch 26/100)    Loss: 1.934399  Accuracy: 25.00%
100%|██████████| 10/10 [00:00<00:00, 58.76it/s]
Train (epoch 27/100)    Loss: 1.861685  Accuracy: 27.00%
100%|██████████| 10/10 [00:00<00:00, 59.62it/s]
Train (epoch 28/100)    Loss: 1.921638  Accuracy: 28.00%
100%|██████████| 10/10 [00:00<00:00, 59.34it/s]
Train (epoch 29/100)    Loss: 2.029971  Accuracy: 32.00%
100%|██████████| 10/10 [00:00<00:00, 57.50it/s]
Train (epoch 30/100)    Loss: 1.867102  Accuracy: 29.00%
100%|██████████| 10/10 [00:00<00:00, 58.41it/s]
Train (epoch 31/100)    Loss: 1.817966  Accuracy: 36.00%
100%|██████████| 10/10 [00:00<00:00, 59.21it/s]
Train (epoch 32/100)    Loss: 1.764226  Accuracy: 35.00%
100%|██████████| 10/10 [00:00<00:00, 58.35it/s]
Train (epoch 33/100)    Loss: 1.719914  Accuracy: 36.00%
100%|██████████| 10/10 [00:00<00:00, 57.06it/s]
Train (epoch 34/100)    Loss: 1.670803  Accuracy: 38.00%
100%|██████████| 10/10 [00:00<00:00, 58.45it/s]
Train (epoch 35/100)    Loss: 1.668811  Accuracy: 38.00%
100%|██████████| 10/10 [00:00<00:00, 57.97it/s]
Train (epoch 36/100)    Loss: 1.570626  Accuracy: 44.00%
100%|██████████| 10/10 [00:00<00:00, 58.85it/s]
Train (epoch 37/100)    Loss: 1.594658  Accuracy: 38.00%
100%|██████████| 10/10 [00:00<00:00, 56.23it/s]
Train (epoch 38/100)    Loss: 1.585293  Accuracy: 40.00%
100%|██████████| 10/10 [00:00<00:00, 58.73it/s]
Train (epoch 39/100)    Loss: 1.527768  Accuracy: 42.00%
100%|██████████| 10/10 [00:00<00:00, 58.28it/s]
Train (epoch 40/100)    Loss: 1.523466  Accuracy: 49.00%
100%|██████████| 10/10 [00:00<00:00, 57.53it/s]
Train (epoch 41/100)    Loss: 1.611788  Accuracy: 41.00%
100%|██████████| 10/10 [00:00<00:00, 58.35it/s]
Train (epoch 42/100)    Loss: 1.490047  Accuracy: 46.00%
100%|██████████| 10/10 [00:00<00:00, 59.19it/s]
Train (epoch 43/100)    Loss: 1.478188  Accuracy: 42.00%
100%|██████████| 10/10 [00:00<00:00, 57.67it/s]
Train (epoch 44/100)    Loss: 1.427063  Accuracy: 47.00%
100%|██████████| 10/10 [00:00<00:00, 58.41it/s]
Train (epoch 45/100)    Loss: 1.449704  Accuracy: 43.00%
100%|██████████| 10/10 [00:00<00:00, 58.24it/s]
Train (epoch 46/100)    Loss: 1.373501  Accuracy: 52.00%
```

```
100%|██████████| 10/10 [00:00<00:00, 58.53it/s]
Train (epoch 47/100)    Loss: 1.336140  Accuracy: 53.00%
100%|██████████| 10/10 [00:00<00:00, 59.37it/s]
Train (epoch 48/100)    Loss: 1.444744  Accuracy: 47.00%
100%|██████████| 10/10 [00:00<00:00, 56.69it/s]
Train (epoch 49/100)    Loss: 1.336955  Accuracy: 53.00%
100%|██████████| 10/10 [00:00<00:00, 58.82it/s]
Train (epoch 50/100)    Loss: 1.330273  Accuracy: 56.00%
100%|██████████| 10/10 [00:00<00:00, 58.61it/s]
Train (epoch 51/100)    Loss: 1.301129  Accuracy: 52.00%
100%|██████████| 10/10 [00:00<00:00, 58.83it/s]
Train (epoch 52/100)    Loss: 1.183607  Accuracy: 61.00%
100%|██████████| 10/10 [00:00<00:00, 58.27it/s]
Train (epoch 53/100)    Loss: 1.201599  Accuracy: 54.00%
100%|██████████| 10/10 [00:00<00:00, 59.23it/s]
Train (epoch 54/100)    Loss: 1.161085  Accuracy: 53.00%
100%|██████████| 10/10 [00:00<00:00, 57.27it/s]
Train (epoch 55/100)    Loss: 1.185804  Accuracy: 58.00%
100%|██████████| 10/10 [00:00<00:00, 58.73it/s]
Train (epoch 56/100)    Loss: 1.153504  Accuracy: 62.00%
100%|██████████| 10/10 [00:00<00:00, 59.23it/s]
Train (epoch 57/100)    Loss: 1.198178  Accuracy: 61.00%
100%|██████████| 10/10 [00:00<00:00, 59.57it/s]
Train (epoch 58/100)    Loss: 1.155060  Accuracy: 62.00%
100%|██████████| 10/10 [00:00<00:00, 58.55it/s]
Train (epoch 59/100)    Loss: 0.997102  Accuracy: 64.00%
100%|██████████| 10/10 [00:00<00:00, 57.66it/s]
Train (epoch 60/100)    Loss: 1.099027  Accuracy: 63.00%
100%|██████████| 10/10 [00:00<00:00, 59.04it/s]
Train (epoch 61/100)    Loss: 0.966277  Accuracy: 68.00%
100%|██████████| 10/10 [00:00<00:00, 58.23it/s]
Train (epoch 62/100)    Loss: 1.020356  Accuracy: 64.00%
100%|██████████| 10/10 [00:00<00:00, 57.71it/s]
Train (epoch 63/100)    Loss: 1.003638  Accuracy: 65.00%
100%|██████████| 10/10 [00:00<00:00, 58.12it/s]
Train (epoch 64/100)    Loss: 0.883082  Accuracy: 71.00%
100%|██████████| 10/10 [00:00<00:00, 58.26it/s]
Train (epoch 65/100)    Loss: 0.872688  Accuracy: 69.00%
100%|██████████| 10/10 [00:00<00:00, 58.46it/s]
Train (epoch 66/100)    Loss: 0.909623  Accuracy: 78.00%
100%|██████████| 10/10 [00:00<00:00, 58.88it/s]
Train (epoch 67/100)    Loss: 0.873946  Accuracy: 72.00%
100%|██████████| 10/10 [00:00<00:00, 58.51it/s]
Train (epoch 68/100)    Loss: 0.858894  Accuracy: 69.00%
100%|██████████| 10/10 [00:00<00:00, 59.25it/s]
Train (epoch 69/100)    Loss: 0.874970  Accuracy: 74.00%
100%|██████████| 10/10 [00:00<00:00, 58.99it/s]
Train (epoch 70/100)    Loss: 0.923939  Accuracy: 64.00%
100%|██████████| 10/10 [00:00<00:00, 57.24it/s]
Train (epoch 71/100)    Loss: 0.910861  Accuracy: 72.00%
100%|██████████| 10/10 [00:00<00:00, 56.16it/s]
Train (epoch 72/100)    Loss: 0.732920  Accuracy: 79.00%
100%|██████████| 10/10 [00:00<00:00, 57.30it/s]
Train (epoch 73/100)    Loss: 0.846929  Accuracy: 78.00%
100%|██████████| 10/10 [00:00<00:00, 55.47it/s]
Train (epoch 74/100)    Loss: 0.899868  Accuracy: 69.00%
100%|██████████| 10/10 [00:00<00:00, 57.36it/s]
Train (epoch 75/100)    Loss: 0.874829  Accuracy: 69.00%
100%|██████████| 10/10 [00:00<00:00, 58.41it/s]
Train (epoch 76/100)    Loss: 0.852087  Accuracy: 75.00%
```

```
100%|██████████| 10/10 [00:00<00:00, 58.51it/s]
Train (epoch 77/100)    Loss: 0.789549  Accuracy: 73.00%
100%|██████████| 10/10 [00:00<00:00, 59.51it/s]
Train (epoch 78/100)    Loss: 0.771053  Accuracy: 78.00%
100%|██████████| 10/10 [00:00<00:00, 58.77it/s]
Train (epoch 79/100)    Loss: 0.800555  Accuracy: 76.00%
100%|██████████| 10/10 [00:00<00:00, 58.97it/s]
Train (epoch 80/100)    Loss: 0.826740  Accuracy: 75.00%
100%|██████████| 10/10 [00:00<00:00, 59.17it/s]
Train (epoch 81/100)    Loss: 0.757706  Accuracy: 77.00%
100%|██████████| 10/10 [00:00<00:00, 58.70it/s]
Train (epoch 82/100)    Loss: 0.740442  Accuracy: 73.00%
100%|██████████| 10/10 [00:00<00:00, 56.43it/s]
Train (epoch 83/100)    Loss: 0.809234  Accuracy: 70.00%
100%|██████████| 10/10 [00:00<00:00, 56.66it/s]
Train (epoch 84/100)    Loss: 0.863745  Accuracy: 74.00%
100%|██████████| 10/10 [00:00<00:00, 57.15it/s]
Train (epoch 85/100)    Loss: 0.985828  Accuracy: 69.00%
100%|██████████| 10/10 [00:00<00:00, 58.43it/s]
Train (epoch 86/100)    Loss: 0.745245  Accuracy: 76.00%
100%|██████████| 10/10 [00:00<00:00, 58.31it/s]
Train (epoch 87/100)    Loss: 0.790299  Accuracy: 77.00%
100%|██████████| 10/10 [00:00<00:00, 57.82it/s]
Train (epoch 88/100)    Loss: 0.803151  Accuracy: 73.00%
100%|██████████| 10/10 [00:00<00:00, 57.55it/s]
Train (epoch 89/100)    Loss: 0.675784  Accuracy: 82.00%
100%|██████████| 10/10 [00:00<00:00, 57.34it/s]
Train (epoch 90/100)    Loss: 0.702526  Accuracy: 77.00%
100%|██████████| 10/10 [00:00<00:00, 55.90it/s]
Train (epoch 91/100)    Loss: 0.691293  Accuracy: 78.00%
100%|██████████| 10/10 [00:00<00:00, 58.52it/s]
Train (epoch 92/100)    Loss: 0.697133  Accuracy: 80.00%
100%|██████████| 10/10 [00:00<00:00, 56.09it/s]
Train (epoch 93/100)    Loss: 0.682923  Accuracy: 79.00%
100%|██████████| 10/10 [00:00<00:00, 55.33it/s]
Train (epoch 94/100)    Loss: 0.799321  Accuracy: 78.00%
100%|██████████| 10/10 [00:00<00:00, 57.04it/s]
Train (epoch 95/100)    Loss: 0.713160  Accuracy: 79.00%
100%|██████████| 10/10 [00:00<00:00, 57.86it/s]
Train (epoch 96/100)    Loss: 0.701933  Accuracy: 80.00%
100%|██████████| 10/10 [00:00<00:00, 58.71it/s]
Train (epoch 97/100)    Loss: 0.768586  Accuracy: 77.00%
100%|██████████| 10/10 [00:00<00:00, 55.35it/s]
Train (epoch 98/100)    Loss: 0.819670  Accuracy: 76.00%
100%|██████████| 10/10 [00:00<00:00, 56.23it/s]
Train (epoch 99/100)    Loss: 0.769987  Accuracy: 76.00%
100%|██████████| 10/10 [00:00<00:00, 58.62it/s]
Train (epoch 100/100)   Loss: 0.719964  Accuracy: 80.00%

Train loss: 0.719964    Train accuracy: 80.00%
```





Testing

In [14]:
```
test(model, test_dataloader)
```

```
100%|██████████| 1000/1000 [00:05<00:00, 189.96it/s]
Test loss: 3.474962     Test accuracy: 19.84%
```

Results

Model	Number of epochs	Train accuracy	Test accuracy
ResNet18	100	0.64	0.1940
ResNet18 (transfer learning)	100	1.00	0.2982
ResNet18 (data augmentation)	100	0.80	0.1984

Conclusions

Question 7 (5 points) : Write a short report explaining the pros and the cons of each method you implemented. 25% of the grade of this project will correspond to this question, thus, it should be done carefully. In particular, please add a plot that will summarize all your numerical results.

Answer:

In this notebook, we implemented three training methods on the CIFAR-10 dataset with very few (only 100) training samples while the testing set amounted for 10,000 samples. The accrucacies reached by each method are displayed in the following table:

Model	Number of epochs	Train accuracy	Test accuracy
ResNet18	100	0.64	0.1940
ResNet18 (transfer learning)	100	1.00	0.2982
ResNet18 (data augmentation)	100	0.80	0.1984

ResNet18 model trained from scratch

The first training method we implemented was training the ResNet18 model from scratch (i.e. from randomly initialized weights). This model reached a training accuracy of 64% after 100 epoch but its accuracy on the testing set was only 19.40%.

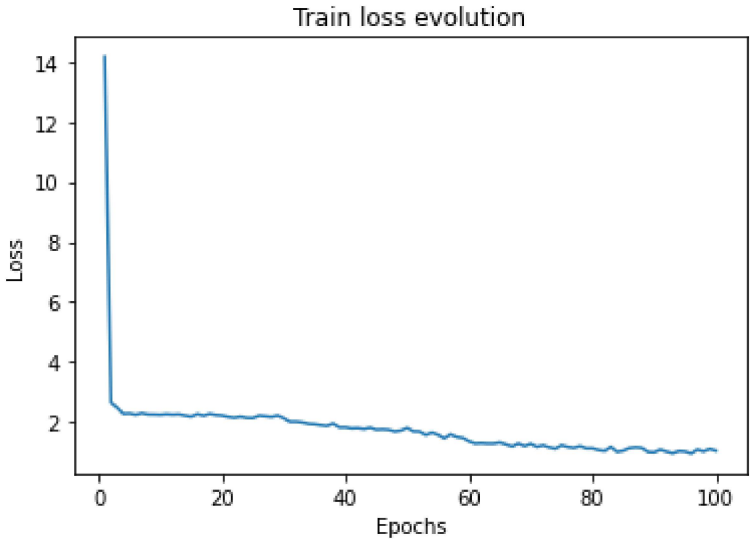
The advantages of this method are the following:

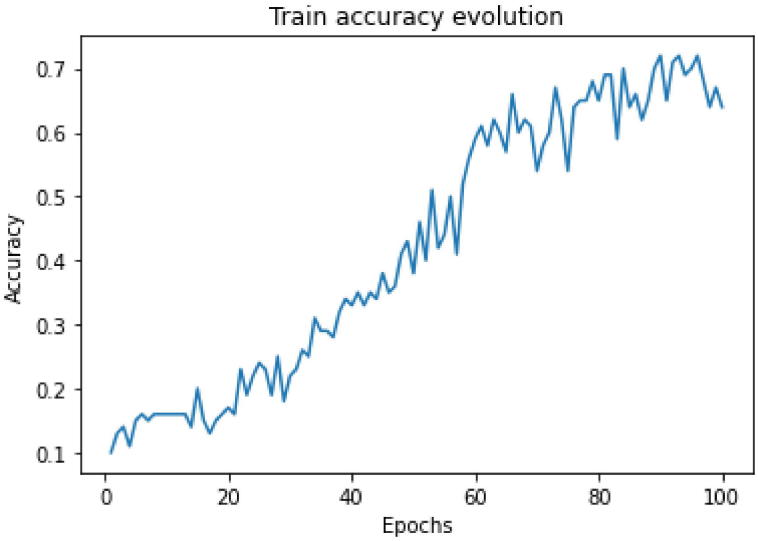
- Does not use pre-trained weights so the model architecture is free to be outside of the typically trained models.

The drawbacks of this method are the following:

- Easily overfits the training set, especially if there are few training samples. Here the training loss was 1.043173 in the end while the testing loss was 2.650611 (i.e. around 2.5 times higher) so we can guess that the overfitting started way before the 100th epoch;
- As shown in the accuracy graph bellow, this method takes a longer time (high number of epochs) to reach a good accuracy. After 60 epochs, the loss is already quite lower than at the start but the accuracy is only around 50%.

Training loss and accuracy graphs:





ResNet18 model trained by transfer learning

The second training method we implemented was training the same model (ResNet18) by transfer learning using weights pre-trained on the ImageNet dataset. We choose to use this same model to be able to reliably compare the results of our experiments. With this method, the training accuracy after 100 epochs was 100% while the testing accuracy was 29.82%.

The advantages of this method are the following:

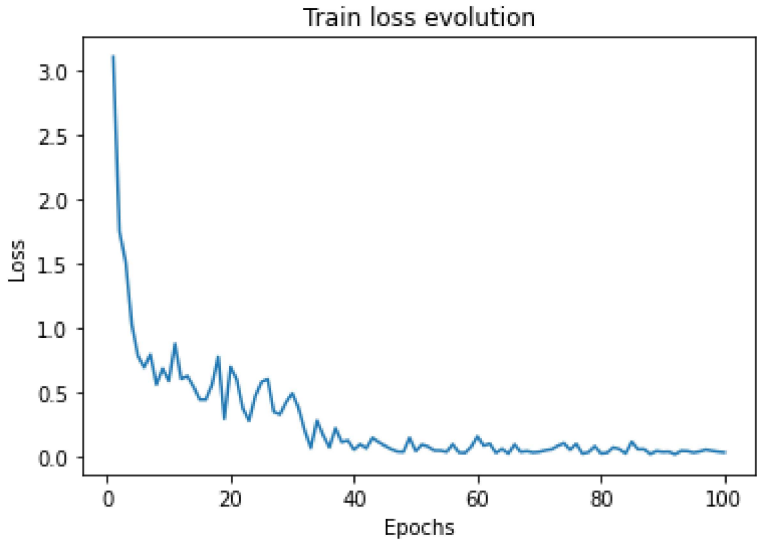
- It reaches a good training accuracy after only a few epoch. As shown in the accuracy graph bellow, the model reaches a training accuracy over 80% after only 20 epochs and at the 40th epoch it reached 100% accuracy already;
- It allows the model to reach lower loss since the basic image features are already knwon by the pre-trained model. The training loss ended up at 0.020519 which is 5 times lower than with the previous method.

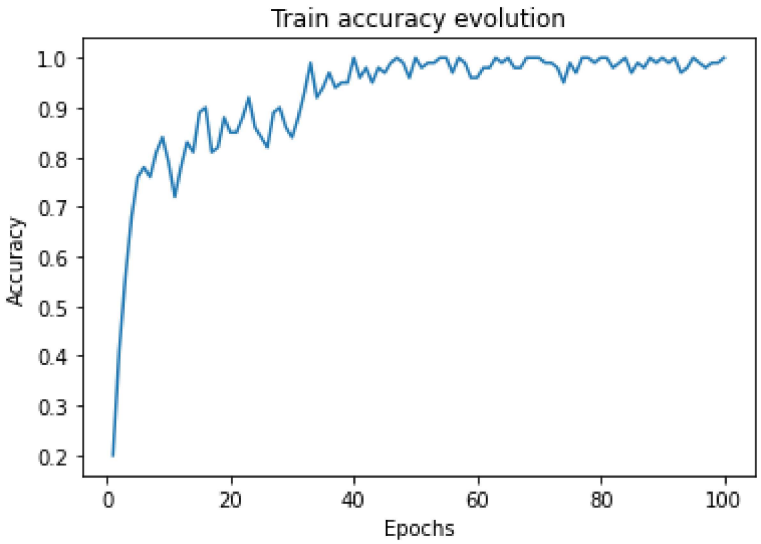
The drawbacks of this method are the following:

- This method is still subject to overfitting. We can see it in the clear gap in training and testing accuracies but also in the fact that the model reaches 100% training accuracy after 40 epochs. We need to carefully fine tune the learning rate and number of epochs in order to reduce this risk;

- The architecture of the model in use has to be part of the typical models trained on ImageNet (or any other dataset). Many pre-trained models are available but it does not allow for unlimited freedom in the architecture of the model.

Training loss and accuracy graphs:





ResNet18 model trained from scratch using data augmentation

The third and last training method we implemented was adding a data augmentation step in the first training method. We used an horizontal flip (no vertical flip to avoid upside down images), a random crop and a normalization of the pixel colors in a preprocessing pipeline. This method reached similar results to the first method with a training accuracy of 80% and a testing accuracy of 19.84%.

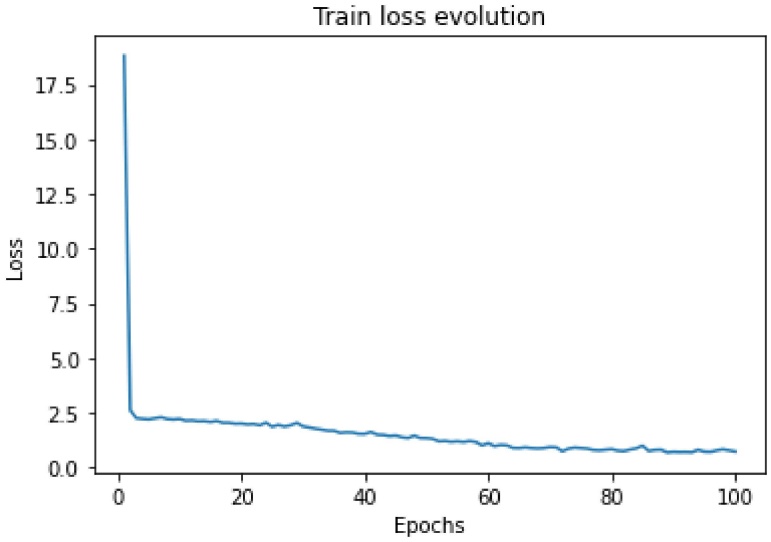
The advantages of this method are the following:

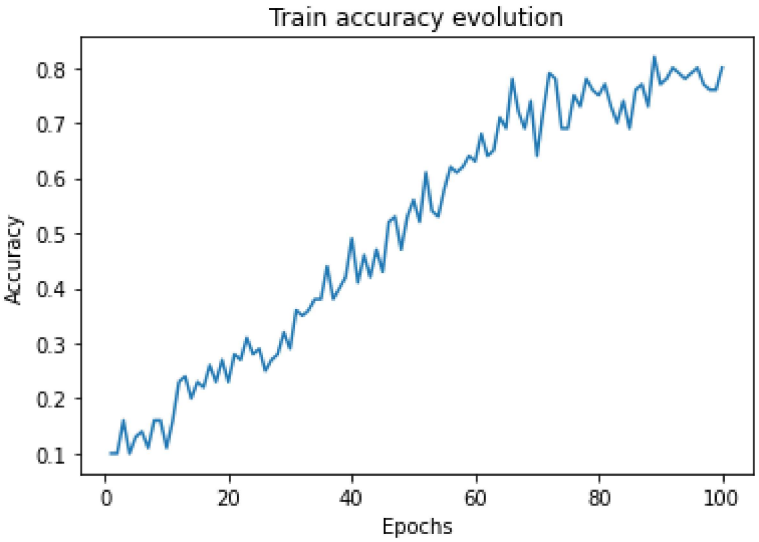
- It can improve the performance of learning processes on small datasets by generating new labeled samples;
- Does not use pre-trained weights so the model architecture is free to be outside of the typically trained models.

The drawbacks of this method are the following:

- With small images such as the ones used here, many transformations are dangerous to use such as translations, rotations, scaling effects or color changes. Thus, the benfits of this method are limited here;
- This method is also subject to overfitting. Here the model reached better training accuracy and loss (0.719964) than the first method but similar testing accuracy and poorer loss (3.474962). This shows that the data augmentation was not successful with only 100 samples of 32×32 images.

Training loss and accuracy graphs:





Weak supervision

Bonus [open] question (up to 4 points) : Pick a weakly supervised method that will potentially use $\mathcal{X}_{\text{nolabel}} \cup \mathcal{X}_{\text{train}}$ to train a model (a subset of $\mathcal{X}_{\text{nolabel}}$ is also fine). Evaluate it and report the accuracies. You should be careful in the choice of your method, in order to avoid heavy computational effort.