Обучение классификатора картинок на примере CIFAR-100 (датасет можно изменить) сверточной сетью (самописной)
Обучение классификатора картинок на примере CIFAR-100 (датасет можно изменить) через дообучение ImageNet Resnet-50
Обучение классификатора картинок на примере CIFAR-100 (датасет можно изменить) через дообучение ImageNet Resnet-50 с аугментацией (самописной, с использованием Pytorch встроенных методов)

In [1]:

```
import numpy as np
import torch

from torch import nn
from torch.nn import functional as F
from PIL import Image
from torchvision import transforms, datasets, models
from tqdm import tqdm

from sklearn.model_selection import train_test_split
```

In [2]:

```
# !jupyter nbextension enable --py widgetsnbextension
```

In [3]:

```
dataset = datasets.CIFAR100(root='data/', train=True, download=True)

def train_valid_split(Xt):
    X_train, X_test = train_test_split(Xt, test_size=0.2, random_state=43)
    return X_train, X_test

class MyOwnCifar(torch.utils.data.Dataset):

    def __init__(self, init_dataset, transform=None):
        self._base_dataset = init_dataset
        self.transform = transform

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

    def __getitem__(self, idx):
        img = self._base_dataset[idx][0]
        if self.transform is not None:
            img = self.transform(img)
        return img, self._base_dataset[idx][1]

trans_actions = transforms.Compose([transforms.Scale(44),
                                    transforms.RandomCrop(32, padding=0),
                                    transforms.ToTensor()])

train_dataset, valid_dataset = train_valid_split(dataset)

train_dataset = MyOwnCifar(train_dataset, trans_actions)
valid_dataset = MyOwnCifar(valid_dataset, transforms.ToTensor())

train_loader = torch.utils.data.DataLoader(train_dataset,
                          batch_size=128,
                          shuffle=True,
                          num_workers=0)
valid_loader = torch.utils.data.DataLoader(valid_dataset,
                          batch_size=128,
                          shuffle=False,
                          num_workers=0)
```

```
Files already downloaded and verified

C:\Users\voron\AppData\Roaming\Python\Python37\site-packages\torchvision\tra
nsforms\transforms.py:317: UserWarning: The use of the transforms.Scale tran
sform is deprecated, please use transforms.Resize instead.
  warnings.warn("The use of the transforms.Scale transform is deprecated, "
+
```

In [4]:

```
dataset
```

Out[4]:

```
Dataset CIFAR100
    Number of datapoints: 50000
    Root location: data/
    Split: Train
```

In [5]:

```
class Net(nn.Module):

    def __init__(self):
        super().__init__()
        self.dp_three = nn.Dropout(0.2)
        self.dp_four = nn.Dropout(0.2)

        self.bn_one = torch.nn.BatchNorm2d(3)
        self.conv_one = torch.nn.Conv2d(3, 30, 3)
        self.bn_two = torch.nn.BatchNorm2d(30)
        self.conv_two = torch.nn.Conv2d(30, 60, 3)
        self.bn_three = torch.nn.BatchNorm2d(60)
        self.conv_three = torch.nn.Conv2d(60, 120, 3)
        self.bn_four = torch.nn.BatchNorm2d(120)
        self.fc1 = torch.nn.Linear(480, 240)
        self.fc2 = torch.nn.Linear(240, 120)
        self.out = torch.nn.Linear(120, 100)


    def forward(self, x):
#         print(x.shape)
        x = self.bn_one(x)
#         print(x.shape)
        x = self.conv_one(x)
#         print(x.shape)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.bn_two(x)
        x = self.conv_two(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.bn_three(x)
        x = self.conv_three(x)
        x = F.leaky_relu(x, 0.1)
        x = F.max_pool2d(x, 2)

        x = self.bn_four(x)
#         print(x.shape)
        x = x.view(x.size(0), -1)
        x = self.dp_three(x)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dp_four(x)
        x = self.fc2(x)
        x = F.relu(x)
        return self.out(x)
        # return x

net = Net()
print(net)
```

```
Net(
  (dp_three): Dropout(p=0.2, inplace=False)
  (dp_four): Dropout(p=0.2, inplace=False)
  (bn_one): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
```

```
  (conv_one): Conv2d(3, 30, kernel_size=(3, 3), stride=(1, 1))
  (bn_two): BatchNorm2d(30, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
  (conv_two): Conv2d(30, 60, kernel_size=(3, 3), stride=(1, 1))
  (bn_three): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
  (conv_three): Conv2d(60, 120, kernel_size=(3, 3), stride=(1, 1))
  (bn_four): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
  (fc1): Linear(in_features=480, out_features=240, bias=True)
  (fc2): Linear(in_features=240, out_features=120, bias=True)
  (out): Linear(in_features=120, out_features=100, bias=True)
)
```

In [6]:

```
optimizer = torch.optim.Adam(net.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()
```

In [7]:

```
epochs = 10
for epoch in tqdm(range(epochs)):
    net.train()
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data[0], data[1]
#         print(data[0], data[1])
        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    net.eval()
    loss_accumed = 0
    for X, y in valid_loader:
        output = net(X)
        loss = criterion(output, y)
        loss_accumed += loss
    print("Epoch {} valid_loss {}".format(epoch, loss_accumed))

print('Training is finished!')
```

```
  0%|
| 0/10 [00:00<?, ?it/s]C:\Users\voron\AppData\Roaming\Python\Python37\site-p
ackages\torch\autograd\__init__.py:156: UserWarning: CUDA initialization: Th
e NVIDIA driver on your system is too old (found version 10010). Please upda
te your GPU driver by downloading and installing a new version from the URL:
http://www.nvidia.com/Download/index.aspx (http://www.nvidia.com/Download/in
dex.aspx) Alternatively, go to: https://pytorch.org (https://pytorch.org) to
install a PyTorch version that has been compiled with your version of the CU
DA driver. (Triggered internally at  ..\c10\cuda\CUDAFunctions.cpp:112.)
  allow_unreachable=True, accumulate_grad=True)  # allow_unreachable flag
 10%|████
| 1/10 [02:05<18:46, 125.16s/it]

Epoch 0 valid_loss 311.4656066894531

 20%|████████
| 2/10 [04:01<16:00, 120.06s/it]

Epoch 1 valid_loss 298.8647766113281

 30%|████████████
| 3/10 [06:10<14:27, 123.96s/it]

Epoch 2 valid_loss 290.56103515625

 40%|████████████████
| 4/10 [08:26<12:52, 128.81s/it]

Epoch 3 valid_loss 285.84112548828125

 50%|████████████████████
| 5/10 [10:45<11:02, 132.44s/it]

Epoch 4 valid_loss 281.1136169433594

 60%|████████████████████████
| 6/10 [13:00<08:53, 133.47s/it]
```

```
Epoch 5 valid_loss 285.297607421875

 70%|███████████████████████████████████████████████████████████████▋
| 7/10 [15:18<06:44, 134.95s/it]

Epoch 6 valid_loss 277.0774230957031

 80%|█████████████████████████████████████████████████████████████████████████
| 8/10 [17:28<04:26, 133.39s/it]

Epoch 7 valid_loss 279.8672180175781

 90%|███████████████████████████████████████████████████████████████████████████████
███▌         | 9/10 [19:44<02:14, 134.13s/it]

Epoch 8 valid_loss 259.97991943359375

100%|███████████████████████████████████████████████████████████████████████████████
██████████| 10/10 [21:52<00:00, 131.26s/it]

Epoch 9 valid_loss 277.7841796875
Training is finished!
```

Image_Net

Создаю класс с собственной структурой, Net

In [32]:

```
class Net(nn.Module):

    def __init__(self):
        super().__init__()
        self.dp_three = nn.Dropout(0.2)
        self.dp_four = nn.Dropout(0.2)

        self.bn_one = torch.nn.BatchNorm2d(256)
        self.conv_one = torch.nn.Conv2d(256, 30, 1)
        self.bn_two = torch.nn.BatchNorm2d(30)
        self.conv_two = torch.nn.Conv2d(30, 60, 3)
        self.bn_three = torch.nn.BatchNorm2d(60)
        self.conv_three = torch.nn.Conv2d(60, 120, 1)
        self.bn_four = torch.nn.BatchNorm2d(120)
        # self.fc1 = torch.nn.Linear(480, 240)
        # self.fc2 = torch.nn.Linear(240, 120)
        # self.out = torch.nn.Linear(120, 100)


    def forward(self, x):
#         print(x.shape)
        x = self.bn_one(x)
#         print(x.shape)
        x = self.conv_one(x)
#         print(x.shape)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.bn_two(x)
        x = self.conv_two(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.bn_three(x)
        x = self.conv_three(x)
        x = F.leaky_relu(x, 0.1)
#         x = F.max_pool2d(x, 2)

#         x = self.bn_four(x)
# #         print(x.shape)
#         x = x.view(x.size(0), -1)
#         x = self.dp_three(x)
#         x = self.fc1(x)
#         x = F.relu(x)
#         x = self.dp_four(x)
#         x = self.fc2(x)
#         x = F.relu(x)
        # return self.out(x)
        return x

net = Net()
print(net)
```

```
Net(
  (dp_three): Dropout(p=0.2, inplace=False)
  (dp_four): Dropout(p=0.2, inplace=False)
  (bn_one): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
```

```
  (conv_one): Conv2d(256, 30, kernel_size=(1, 1), stride=(1, 1))
  (bn_two): BatchNorm2d(30, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
  (conv_two): Conv2d(30, 60, kernel_size=(3, 3), stride=(1, 1))
  (bn_three): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
  (conv_three): Conv2d(60, 120, kernel_size=(1, 1), stride=(1, 1))
  (bn_four): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
)
```

Создаю класс с собственной структурой, Net1

In [33]:

```
class Net1(nn.Module):

    def __init__(self):
        super().__init__()
        self.dp_three = nn.Dropout(0.2)
        self.dp_four = nn.Dropout(0.2)

        self.bn_one = torch.nn.BatchNorm2d(120)
        self.conv_one = torch.nn.Conv2d(120, 30, 1)
        self.bn_two = torch.nn.BatchNorm2d(30)
        self.conv_two = torch.nn.Conv2d(30, 512, 2)
        self.bn_three = torch.nn.BatchNorm2d(512)
        self.conv_three = torch.nn.Conv2d(512, 1024, 1)
        self.bn_four = torch.nn.BatchNorm2d(1024)
        # self.fc1 = torch.nn.Linear(480, 240)
        # self.fc2 = torch.nn.Linear(240, 120)
        # self.out = torch.nn.Linear(120, 100)


    def forward(self, x):
#         print(x.shape)
        x = self.bn_one(x)
#         print(x.shape)
        x = self.conv_one(x)
#         print(x.shape)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.bn_two(x)
        x = self.conv_two(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)

        x = self.bn_three(x)
        x = self.conv_three(x)
        x = F.leaky_relu(x, 0.1)
        # x = F.max_pool2d(x, 2)

#         x = self.bn_four(x)
# #         print(x.shape)
#         x = x.view(x.size(0), -1)
#         x = self.dp_three(x)
#         x = self.fc1(x)
#         x = F.relu(x)
#         x = self.dp_four(x)
#         x = self.fc2(x)
#         x = F.relu(x)
        # return self.out(x)
        return x

net1 = Net1()
print(net1)
```

```
Net1(
  (dp_three): Dropout(p=0.2, inplace=False)
  (dp_four): Dropout(p=0.2, inplace=False)
  (bn_one): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
```

```
  (conv_one): Conv2d(120, 30, kernel_size=(1, 1), stride=(1, 1))
  (bn_two): BatchNorm2d(30, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
  (conv_two): Conv2d(30, 512, kernel_size=(2, 2), stride=(1, 1))
  (bn_three): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
  (conv_three): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1))
  (bn_four): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
)
```

Загружаю готовую сеть resnet50, pretrained= True - это значит сеть подгрузит свои веса, уже ранее обученные

In [34]:

```
resnet = models.resnet50(pretrained= True)
# resnet
```

Функция для установки флажка requires_grad в False, чтоб веса загруженной и уже обученной до меня сети не менялись

In [35]:

```
def set_parameter_requires_grad(model):
      for param in model.parameters():
          param.requires_grad = False
```

Загружаю датасет cifar100

In [36]:

```
dataset = datasets.CIFAR100(root='data/', train=True, download=True)

def train_valid_split(Xt):
    X_train, X_test = train_test_split(Xt, test_size=0.2, random_state=43)
    return X_train, X_test

class MyOwnCifar(torch.utils.data.Dataset):

    def __init__(self, init_dataset, transform=None):
        self._base_dataset = init_dataset
        self.transform = transform

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

    def __getitem__(self, idx):
        img = self._base_dataset[idx][0]
        if self.transform is not None:
            img = self.transform(img)
        return img, self._base_dataset[idx][1]

trans_actions = transforms.Compose([transforms.Scale(44),
                                    transforms.RandomCrop(32, padding=4),
                                    transforms.ToTensor()])

train_dataset, valid_dataset = train_valid_split(dataset)

train_dataset = MyOwnCifar(train_dataset, trans_actions)
valid_dataset = MyOwnCifar(valid_dataset, transforms.ToTensor())

train_loader = torch.utils.data.DataLoader(train_dataset,
                          batch_size=128,
                          shuffle=True,
                          num_workers=0)
valid_loader = torch.utils.data.DataLoader(valid_dataset,
                          batch_size=128,
                          shuffle=False,
                          num_workers=0)
```

```
Files already downloaded and verified
```

In [37]:

```
dataset
```

Out[37]:

```
Dataset CIFAR100
    Number of datapoints: 50000
    Root location: data/
    Split: Train
```

Выходные слои меняю на свои, линейные

У них Градиент стоит в True

Ставлю requires_grad в False, чтоб веса загруженной и уже обученной до меня сети не менялись

In [38]:

```
set_parameter_requires_grad(resnet)
resnet.fc = nn.Linear(2048, 1024)
resnet.fc1 = nn.Linear(1024, 512)
resnet.fc2 = nn.Linear(512, 256)
resnet.fc3 = nn.Linear(256, 100)
# resnet
```

Меняю слой 2 и 3 у сети resnet50 на свои, у них градиент стоит в True

In [27]:

```
resnet.layer2 = Net()
resnet.layer3 = Net1()
```

In [28]:

```
resnet
```

Out[28]:

```
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_runn
ing_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ce
il_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Net(
    (dp_three): Dropout(p=0.2, inplace=False)
    (dp_four): Dropout(p=0.2, inplace=False)
    (bn_one): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
    (conv_one): Conv2d(256, 30, kernel_size=(1, 1), stride=(1, 1))
    (bn_two): BatchNorm2d(30, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
    (conv_two): Conv2d(30, 60, kernel_size=(3, 3), stride=(1, 1))
    (bn_three): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
    (conv_three): Conv2d(60, 120, kernel_size=(1, 1), stride=(1, 1))
    (bn_four): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
  )
  (layer3): Net1(
    (dp_three): Dropout(p=0.2, inplace=False)
    (dp_four): Dropout(p=0.2, inplace=False)
    (bn_one): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
    (conv_one): Conv2d(120, 30, kernel_size=(1, 1), stride=(1, 1))
    (bn_two): BatchNorm2d(30, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
    (conv_two): Conv2d(30, 512, kernel_size=(2, 2), stride=(1, 1))
    (bn_three): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
    (conv_three): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1))
    (bn_four): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, tr
ack_running_stats=True)
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), paddin
g=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=F
alse)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=
False)
```

```
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=
False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1024, bias=True)
  (fc1): Linear(in_features=1024, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=100, bias=True)
)
```

Обязательная трансформация датасета, для сети resnet50

https://pytorch.org/vision/stable/models.html (https://pytorch.org/vision/stable/models.html)

In [39]:

```
trans_actions = transforms.Compose([transforms.Scale(256),
                                    transforms.RandomCrop(224, padding=0),
                                    transforms.ToTensor(),
                                    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                         std=[0.229, 0.224, 0.225])])
valid_transforms = transforms.Compose([transforms.ToTensor(),
                                       transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                         std=[0.229, 0.224, 0.225])])

train_dataset, valid_dataset = train_valid_split(dataset)

train_dataset = MyOwnCifar(train_dataset, trans_actions)
valid_dataset = MyOwnCifar(valid_dataset, valid_transforms)

train_loader = torch.utils.data.DataLoader(train_dataset,
                          batch_size=128,
                          shuffle=True,
                          num_workers=0)
valid_loader = torch.utils.data.DataLoader(valid_dataset,
                          batch_size=128,
                          shuffle=False,
                          num_workers=0)
```

Обучаются все разделы где requires_grad стоит в True

In [40]:

```
params_to_update = []
for name,param in resnet.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)

optimizer = torch.optim.Adam(params_to_update, lr=0.001)
criterion = nn.CrossEntropyLoss()
```

In [41]:

```
epochs = 1
for epoch in tqdm(range(epochs)):
    resnet.train()
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data[0], data[1]
        optimizer.zero_grad()

        outputs = resnet(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    resnet.eval()
    loss_accumed = 0
    for X, y in valid_loader:
        output = resnet(X)
        loss = criterion(output, y)
        loss_accumed += loss
    print("Epoch {} valid_loss {}".format(epoch, loss_accumed))

print('Training is finished!')
```

```
100%|███████████████████████████████████████████████████████████████████
█████████| 1/1 [1:58:33<00:00, 7113.18s/it]

Epoch 0 valid_loss 998.78369140625
Training is finished!
```

ResNet50 с аугментацией

In [42]:

```
resnet50 = models.resnet50(pretrained=True)
```

In [43]:

```
dataset = datasets.CIFAR100(root='data/', train=True, download=True)

def train_valid_split(Xt):
    X_train, X_test = train_test_split(Xt, test_size=0.2, random_state=43)
    return X_train, X_test

class MyOwnCifar(torch.utils.data.Dataset):

    def __init__(self, init_dataset, transform=None):
        self._base_dataset = init_dataset
        self.transform = transform

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

    def __getitem__(self, idx):
        img = self._base_dataset[idx][0]
        if self.transform is not None:
            img = self.transform(img)
        return img, self._base_dataset[idx][1]

# trans_actions = transforms.Compose([transforms.Scale(44),
#                                     transforms.RandomCrop(32, padding=4),
#                                     transforms.ToTensor()])

# train_dataset, valid_dataset = train_valid_split(dataset)

# train_dataset = MyOwnCifar(train_dataset, trans_actions)
# valid_dataset = MyOwnCifar(valid_dataset, transforms.ToTensor())

# train_loader = torch.utils.data.DataLoader(train_dataset,
#                            batch_size=128,
#                            shuffle=True,
#                            num_workers=0)
# valid_loader = torch.utils.data.DataLoader(valid_dataset,
#                            batch_size=128,
#                            shuffle=False,
#                            num_workers=0)
```

```
Files already downloaded and verified
```

In [44]:

```
def set_parameter_requires_grad(model):
      for param in model.parameters():
          param.requires_grad = False
```

In [45]:

```
set_parameter_requires_grad(resnet50)
resnet50.fc = nn.Linear(2048, 100)
# resnet
```

In [46]:

```
from torchvision.transforms.functional import InterpolationMode
trans_actions = transforms.Compose([transforms.Scale(256),
                                    transforms.RandomCrop(224, padding=0),
                                    # transforms.AutoAugment(),
                                    transforms.RandAugment(num_ops=3, interpolation= Interp
                                    transforms.ToTensor(),
                                    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                         std=[0.229, 0.224, 0.225])])
valid_transforms = transforms.Compose([transforms.ToTensor(),
                                       transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                            std=[0.229, 0.224, 0.225])])

train_dataset, valid_dataset = train_valid_split(dataset)

train_dataset = MyOwnCifar(train_dataset, trans_actions)
valid_dataset = MyOwnCifar(valid_dataset, valid_transforms)

train_loader = torch.utils.data.DataLoader(train_dataset,
                          batch_size=128,
                          shuffle=True,
                          num_workers=0)
valid_loader = torch.utils.data.DataLoader(valid_dataset,
                          batch_size=128,
                          shuffle=False,
                          num_workers=0)
```

In [47]:

```
params_to_update = []
for name,param in resnet50.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)

optimizer = torch.optim.Adam(params_to_update, lr=0.001)
criterion = nn.CrossEntropyLoss()
```

In [48]:

```
epochs = 1
for epoch in tqdm(range(epochs)):
    resnet50.train()
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data[0], data[1]
        optimizer.zero_grad()

        outputs = resnet50(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    resnet50.eval()
    loss_accumed = 0
    for X, y in valid_loader:
        output = resnet50(X)
        loss = criterion(output, y)
        loss_accumed += loss
    print("Epoch {} valid_loss {}".format(epoch, loss_accumed))

print('Training is finished!')
```

```
100%|██████████████████████████████████████████████████████
████████| 1/1 [3:00:23<00:00, 10823.03s/it]

Epoch 0 valid_loss 721.6709594726562
Training is finished!
```

In [ ]: