Глубинное обучение, ИИ ВШЭ

Домашнее задание 2. Классификация при помощи CNN.

Общая информация

Оценивание и штрафы

Максимально допустимая оценка за работу без бонусов — 10 баллов. Сдавать задание после указанного срока жесткого дедлайна нельзя.

Сдача работы после мягкого дедлайна штрафуется ступенчато, -1 балл в сутки. Один раз за модуль студентам предоставляется возможность использовать отсрочку и сдать в жесткий дедлайн без штрафа.

Задание выполняется самостоятельно. «Похожие» решения считаются плагиатом и все задействованные студенты (в том числе те, у кого списали) не могут получить за него больше 0 баллов. Если вы нашли решение какого-то из заданий (или его часть) в открытом источнике, необходимо указать ссылку на этот источник в отдельном блоке в конце вашей работы (скорее всего вы будете не единственным, кто это нашел, поэтому чтобы исключить подозрение в плагиате, необходима ссылка на источник).

Неэффективная реализация кода может негативно отразиться на оценке. Также оценка может быть снижена за плохо читаемый код и плохо оформленные графики. Все ответы должны сопровождаться кодом или комментариями о том, как они были получены.

Использование генеративных моделей допустимо на следующих условиях:

- Количество кода, написанное генеративными моделями, не превышает 30%
- Указана модель, использованная для генерации, а также промпт
- В конце работы необходимо описать свой опыт использования генеративного ИИ для решения данного домашнего задания. Укажите как часто Вам приходилось исправлять код своими руками или просить модель что-то исправить. Было ли это быстрее, чем написать код самим?

В случае невыполнения этих требований работа не оценивается и оценка за неё не превышает 0 баллов.

О задании

В этом задании вам предстоит познакомиться со сверточными сетями и их обучением для классификации изображений с использованием библиотеки PyTorch.

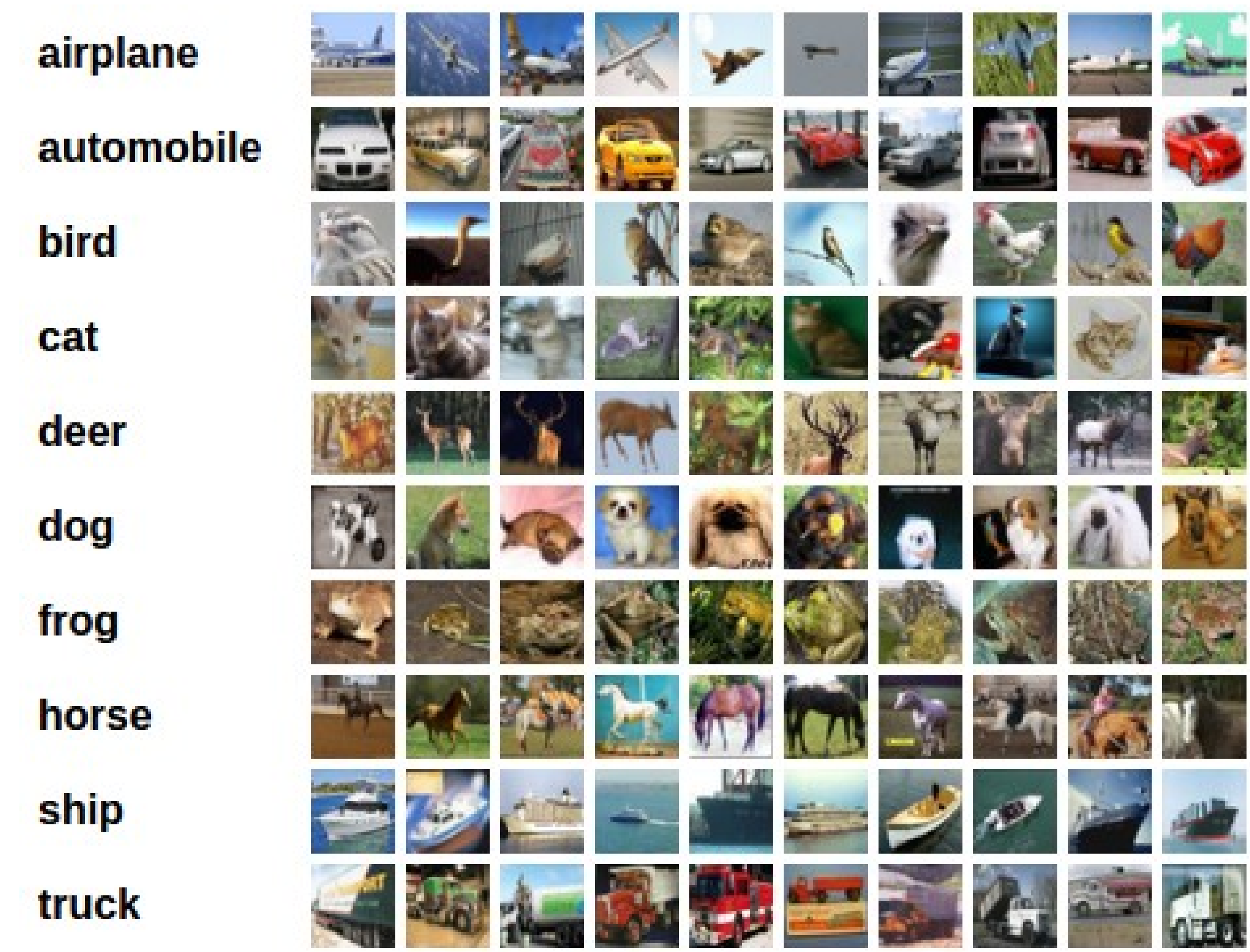
```python
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
```

```python
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split
```

Для VSCode `pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu126`

0. Загрузка данных

Работать мы будем с набором данных CIFAR10. CIFAR10 представляет собой набор изображений 32x32 пикселя, разделенных на 10 классов.



Набор данных уже определен в torchvision.datasets, так что возьмем его оттуда.

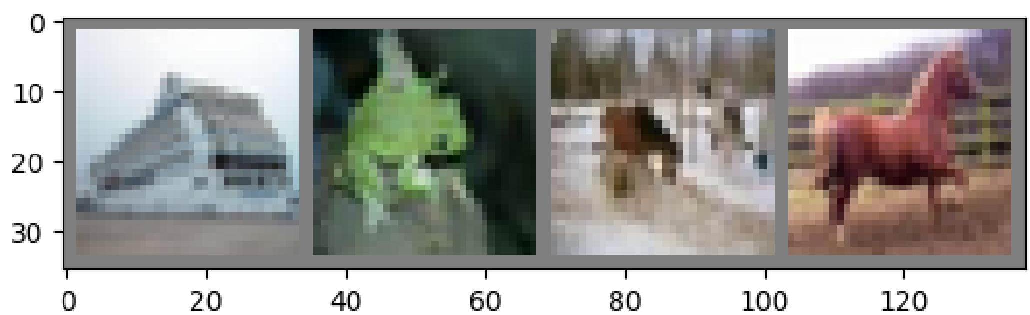
```python
# Для тру нормализации
# mean = [0.4914, 0.4822, 0.4465]
# std = [0.2470, 0.2435, 0.2616]
# Тут взяты значения как в одном из туториалов пайторча :)
```

```
def get_cifar10_data(batch_size, transform_train):
    torch.manual_seed(0)
    np.random.seed(0)

    transform_test = transforms.Compose(
        [
            transforms.ToTensor(),
            # Переводим цвета пикселей в отрезок [-1, 1]
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ]
    )

    # Загружаем данные
    trainvalset = torchvision.datasets.CIFAR10(
        root="./data", train=True, download=True,
transform=transform_train
    )
    testset = torchvision.datasets.CIFAR10(
        root="./data", train=False, download=True,
transform=transform_test
    )

    # В датасете определено разбиение только на train и test,
    # так что валидацию дополнительно выделяем из обучающей выборки
    train_idx, valid_idx = train_test_split(
        np.arange(len(trainvalset)), test_size=0.3, shuffle=True,
random_state=0
    )
    trainset = torch.utils.data.Subset(trainvalset, train_idx)
    valset = torch.utils.data.Subset(trainvalset, valid_idx)

    train_loader = torch.utils.data.DataLoader(
        trainset, batch_size=batch_size, shuffle=True, num_workers=2
    )
    val_loader = torch.utils.data.DataLoader(
        valset, batch_size=batch_size, shuffle=False, num_workers=2
    )
    test_loader = torch.utils.data.DataLoader(
        testset, batch_size=batch_size, shuffle=False, num_workers=2
    )

    return train_loader, val_loader, test_loader

transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5),
(0.5, 0.5, 0.5))]
)
```

```
train_loader, val_loader, test_loader = get_cifar10_data(
    batch_size=64, transform_train=transform
)
```

Посмотрим на изображения:

```
def imshow(img):
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


dataiter = iter(train_loader)
images, labels = next(dataiter)

imshow(torchvision.utils.make_grid(images[:4]))

classes = (
    "plane",
    "car",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
)
print(*[classes[labels[i]] for i in range(4)])
```




```
ship frog horse horse
```

1. Задание сверточной сети (3 балла)

Теперь нам нужно задать сверточную нейронную сеть, которую мы будем обучать классифицировать изображения.

Используем сеть, основанную на одном блоке архитектуры, похожей на ResNet. Обратите внимание, это не ResNet 1 в 1.

Указания:

- Все сверточные слои должны иметь 32 выходных канала, а также не должны изменять ширину и высоту изображения.
- Выход блока сократите до размерности 32x4x4, применив average pooling.
- Для получения итоговых логитов, распрямите выход пулинга в вектор из 512 элементов, а затем пропустите его через линейный слой.

Задание 1.1 (3 балла).

Определите архитектуру сети соответственно схеме и указаниям выше.

Ключевые слова: Conv2d, BatchNorm2d, AvgPool2d.

```
n_classes = 10

class BasicBlockNet(nn.Module):
    def __init__(self):
        super().__init__()
        # первый слой
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32,
kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.relu = nn.ReLU()

        # второй слой
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=32,
kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(32)

        self.downsample = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=1, bias=False),
            nn.BatchNorm2d(32),
        )

        self.avg_pool = nn.AvgPool2d(kernel_size=8)
        self.fc = nn.Linear(32 * 4 * 4, n_classes)

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
```

```
        if identity.shape[1] != out.shape[1]:
            identity = self.downsample(identity)

        out += identity
        out = self.relu(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)

        return out

net = BasicBlockNet()
net
```

```
BasicBlockNet(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU()
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (downsample): Sequential(
    (0): Conv2d(3, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  )
  (avg_pool): AvgPool2d(kernel_size=8, stride=8, padding=0)
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

Проверим, что выход сети имеет корректную размерность:

```
assert net(torch.zeros((10, 3, 32, 32))).shape == (10, 10)
```

Чтобы проводить вычисления на GPU, в PyTorch необходимо руками перекладывать объекты, с которыми вы хотите проводить вычисления, на графический ускоритель. Это делается следующим образрм:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
print(device)
```

```
cuda:0
```

```
net = net.to(device)
```

Подключение GPU в google.colab:

Среда выполнения -> **Сменить среду выполнения** -> **Аппаратный ускоритель** -> **GPU**

2. Обучение и тестирование модели (3 балла)

Задание 2.1 (2 балла). Переходим к обучению модели. Заполните пропуски в функциях test и train_epoch. В качестве функции потерь будем использовать кросс-энтропию, а в качестве метрики качества accuracy.

```
def test(model, loader):
    loss_log = []
    acc_log = []
    model.eval()

    with torch.no_grad():
        for data, target in loader:
            data, target = data.to(device), target.to(device)

            output = model(data)
            loss = F.cross_entropy(output, target)
            loss_log.append(loss.item())

            pred = output.argmax(dim=1)
            acc = (pred == target).float().mean()
            acc_log.append(acc.item())

    return np.mean(loss_log), np.mean(acc_log)

def train_epoch(model, optimizer, train_loader):
    loss_log = []
    acc_log = []
    model.train()

    for data, target in train_loader:
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()

        loss_log.append(loss.item())

        pred = output.argmax(dim=1)
        acc = (pred == target).float().mean()
        acc_log.append(acc.item())

    return loss_log, acc_log

def train(model, optimizer, n_epochs, train_loader, val_loader,
scheduler=None):
```

```
    train_loss_log, train_acc_log, val_loss_log, val_acc_log = [], [],
[], []

    for epoch in range(n_epochs):
        train_loss, train_acc = train_epoch(model, optimizer,
train_loader)
        val_loss, val_acc = test(model, val_loader)

        train_loss_log.extend(train_loss)
        train_acc_log.extend(train_acc)

        val_loss_log.append(val_loss)
        val_acc_log.append(val_acc)

        print(f"Epoch {epoch}")
        print(f" train loss: {np.mean(train_loss)}, train acc:
{np.mean(train_acc)}")
        print(f" val loss: {val_loss}, val acc: {val_acc}\n")

        if scheduler is not None:
            scheduler.step()

    return train_loss_log, train_acc_log, val_loss_log, val_acc_log
```

Запустим обучение модели. В качестве оптимизатора будем использовать стохастический градиентный спуск, который является де-факто стандартом в задачах компьютерного зрения (наравне с Adam).

Замечание: Для достижения наилучшего качества в нашем случае потребуется обучать модель несколько сотен эпох. Однако в целях экономии вашего времени и сил, во всех экспериментах мы ограничимся 20 эпохами.

```
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
train_loss_log, train_acc_log, val_loss_log, val_acc_log = train(
    model=net,
    optimizer=optimizer,
    n_epochs=20,
    train_loader=train_loader,
    val_loader=val_loader
)

Epoch 0
 train loss: 1.4731886318222478, train acc: 0.4781927396653993
 val loss: 1.2513684853594353, val acc: 0.5621675531914894

Epoch 1
 train loss: 1.159637240757672, train acc: 0.5937704035307415
 val loss: 1.078228619758119, val acc: 0.6195478723404255

Epoch 2
```

```
 train loss: 1.0403122053067906, train acc: 0.636177037163234
 val loss: 1.038786394038099, val acc: 0.641533688027808

Epoch 3
 train loss: 0.9795560270166485, train acc: 0.6601348263254113
 val loss: 0.9671502460824682, val acc: 0.6677304965384463

Epoch 4
 train loss: 0.948230747548928, train acc: 0.671352670441162
 val loss: 0.9624964473095347, val acc: 0.6688829787234043

Epoch 5
 train loss: 0.9078494265702787, train acc: 0.6856473623289705
 val loss: 0.961391475606472, val acc: 0.6661125888215734

Epoch 6
 train loss: 0.8933940439381155, train acc: 0.6899933077299617
 val loss: 0.9562981265656492, val acc: 0.6734707446808511

Epoch 7
 train loss: 0.8717328792189332, train acc: 0.6958164665337236
 val loss: 0.9064367509902792, val acc: 0.6849512412192974

Epoch 8
 train loss: 0.8499127786054907, train acc: 0.7051653500230918
 val loss: 0.8836828756839671, val acc: 0.6940159574468086

Epoch 9
 train loss: 0.8324515217509958, train acc: 0.7115557261315298
 val loss: 0.9013725762671613, val acc: 0.6936835106382979

Epoch 10
 train loss: 0.8189055783983994, train acc: 0.7189009859749342
 val loss: 0.9293955333689425, val acc: 0.6802748228641267

Epoch 11
 train loss: 0.8036195475614921, train acc: 0.7211290481321555
 val loss: 0.9149961351080144, val acc: 0.69350620584285

Epoch 12
 train loss: 0.7937193956954823, train acc: 0.7259361126087266
 val loss: 0.8599093574158689, val acc: 0.7054742909492331

Epoch 13
 train loss: 0.7790925521846227, train acc: 0.7315633978006827
 val loss: 0.8729152184851626, val acc: 0.705097517815042

Epoch 14
 train loss: 0.7798509367314291, train acc: 0.730710531501491
 val loss: 0.870877295352043, val acc: 0.702437943346957
```

```
Epoch 15
 train loss: 0.7615348773530022, train acc: 0.7379415317470892
 val loss: 0.8763258885830006, val acc: 0.7014849292471054

Epoch 16
 train loss: 0.7536728207219453, train acc: 0.7398676221723748
 val loss: 0.839640924651572, val acc: 0.713785461161999

Epoch 17
 train loss: 0.7476223928400978, train acc: 0.7419895861004998
 val loss: 0.8681943974596389, val acc: 0.7057845744680851

Epoch 18
 train loss: 0.7434913895247841, train acc: 0.7455846827470406
 val loss: 0.8550271332263947, val acc: 0.7103501773895101

Epoch 19
 train loss: 0.7353402199104455, train acc: 0.7451684513083324
 val loss: 0.8940082240611948, val acc: 0.6991578015875309
```

Посчитайте точность на тестовой выборке:

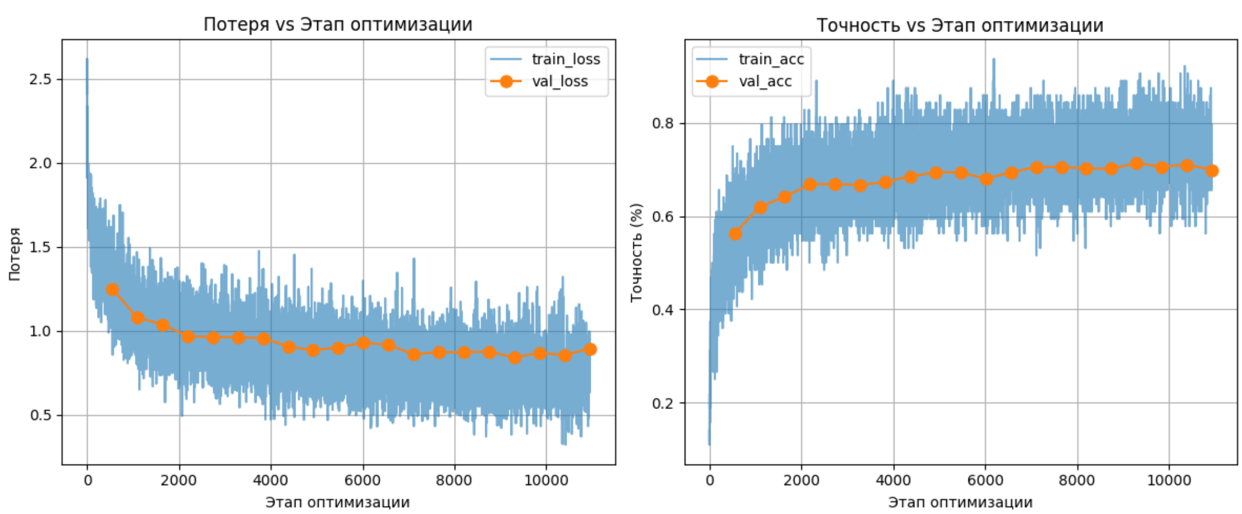
```
test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")

Test Loss: 0.91%
Test Accuracy: 0.70%
```

Если вы все сделали правильно, у вас должна была получиться точность $\geq 67\%$.

Задание 2.2 (1 балл). Постройте графики зависимости функции потерь и точности классификации от номера шага оптимизатора. На каждом графике расположите данные и для обучающей и для валидационной выборки, итого у вас должно получиться два графика. Обратите внимание, что на обучающей выборке эти данные считаются по каждому батчу, на валидационной же они считаются по всей выборке раз в эпоху.

```
def plot_metrics(train_loss, train_acc, val_loss, val_acc, 
train_loader):
    """
    Функция построения графиков (1 - функция потерь, 2 - точность)
    """
    iter_per_epoch = len(train_loader)
    val_iterations = [(i + 1) * iter_per_epoch for i in 
range(len(val_loss))]

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
```

```
    plt.plot(range(len(train_loss)), train_loss, label='train_loss',
alpha=0.6)
    plt.plot(val_iterations, val_loss, 'o-', label='val_loss',
markersize=8)
    plt.xlabel('Этап оптимизации')
    plt.ylabel('Потеря')
    plt.title('Потеря vs Этап оптимизации')
    plt.legend()
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(range(len(train_acc)), train_acc, label='train_acc',
alpha=0.6)
    plt.plot(val_iterations, val_acc, 'o-', label='val_acc',
markersize=8)
    plt.xlabel('Этап оптимизации')
    plt.ylabel('Точность (%)')
    plt.title('Точность vs Этап оптимизации')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

plot_metrics(train_loss_log, train_acc_log, val_loss_log, val_acc_log,
train_loader)
```



3. Расписание длины шага (2 балла)

С курса "Машинное обучение 1" вы уже должны знать, что сходимость стохастического градиентного спуска мы можем теоретически гарантировать только если будем определенным образом со временем уменьшать длину шага. На практике при обучении нейронных сетей такая техника оказывается очень полезной, однако теоретически обоснованными способами уменьшения длины шага фантазия не ограничивается.

Одним из простейших способов является кусочно постоянная функция: на нескольких фиксированных эпохах уменьшаем длину шага в константу раз.

```
net = BasicBlockNet().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[10,
15], gamma=0.1)
train_loss_log, train_acc_log, val_loss_log, val_acc_log = train(
    model=net,
    optimizer=optimizer,
    n_epochs=20,
    train_loader=train_loader,
    val_loader=val_loader,
    scheduler=scheduler
)
```

```
Epoch 0
 train loss: 1.525838551181326, train acc: 0.4582993275395695
 val loss: 1.2803500695431487, val acc: 0.5474512412192973

Epoch 1
 train loss: 1.204591201689823, train acc: 0.5773211021310015
 val loss: 1.2027166729277752, val acc: 0.582734929120287

Epoch 2
 train loss: 1.0694328468068208, train acc: 0.6294969313759254
 val loss: 1.0348743913021494, val acc: 0.6361258866939139

Epoch 3
 train loss: 1.0020637069803269, train acc: 0.6501044659335609
 val loss: 1.071901992787706, val acc: 0.630407801587531

Epoch 4
 train loss: 0.969796851304593, train acc: 0.6605265735490231
 val loss: 0.9989283282706078, val acc: 0.6514849292471053

Epoch 5
 train loss: 0.9430607734912057, train acc: 0.6733318099809957
 val loss: 0.9557321720934929, val acc: 0.6658687944107867

Epoch 6
 train loss: 0.9110542650196627, train acc: 0.6811259467379485
 val loss: 0.9340220839419263, val acc: 0.6758200356300841

Epoch 7
 train loss: 0.8905453923851306, train acc: 0.6920785453899251
 val loss: 0.9139502456847658, val acc: 0.6871453901554676

Epoch 8
 train loss: 0.8754988217920446, train acc: 0.6950166493490801
```

```
 val loss: 0.9248204969345255, val acc: 0.6822916667512122

Epoch 9
 train loss: 0.8551015797764118, train acc: 0.702863835220581
 val loss: 0.9688874642899696, val acc: 0.6693484042553192

Epoch 10
 train loss: 0.7452409959362456, train acc: 0.7427200314331752
 val loss: 0.8321060393718963, val acc: 0.7162012412192974

Epoch 11
 train loss: 0.7261823047449646, train acc: 0.7493838143740973
 val loss: 0.8232192667240792, val acc: 0.7154476952045522

Epoch 12
 train loss: 0.719956950197708, train acc: 0.7510079329584809
 val loss: 0.826112669199071, val acc: 0.719658688027808

Epoch 13
 train loss: 0.7150959776241121, train acc: 0.751852637779996
 val loss: 0.8198071912248084, val acc: 0.7195478723404255

Epoch 14
 train loss: 0.7118430027033337, train acc: 0.7524769848835752
 val loss: 0.8192460410138394, val acc: 0.7202792553191489

Epoch 15
 train loss: 0.6961201584099416, train acc: 0.7588143118774651
 val loss: 0.8102117204919774, val acc: 0.7221852837724888

Epoch 16
 train loss: 0.694509340269692, train acc: 0.7597447115063013
 val loss: 0.8096306586519201, val acc: 0.7223182624958931

Epoch 17
 train loss: 0.6942930778376145, train acc: 0.7610995038335895
 val loss: 0.8093883830182096, val acc: 0.7239361702127659

Epoch 18
 train loss: 0.6928260411924156, train acc: 0.7600752481376885
 val loss: 0.8089694559574128, val acc: 0.722783688027808

Epoch 19
 train loss: 0.6924927440596237, train acc: 0.7590550731261426
 val loss: 0.8100484964695382, val acc: 0.7220523050490846
```

Посчитайте точность на тестовой выборке:

```
test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")

Test Loss: 0.82%
Test Accuracy: 0.72%
```

Точность увеличилась

Задание 3.0 (0.5 баллов). Здесь может возникнуть вопрос: а что будет, если мы не будем уменьшать длину шага в процессе обучения, а сразу возьмем констатную, равную значению нашей кусочно-постоянной функции на последних эпохах, то есть 0.001 в нашем случае. Запустите обучение и проверьте, что в таком случае мы получим худшее качество на тестовой выборке.

```
net_fixed_lr = BasicBlockNet().to(device)
optimizer_fixed = optim.SGD(net_fixed_lr.parameters(), lr=0.001,
momentum=0.9)

train_loss_fixed, train_acc_fixed, val_loss_fixed, val_acc_fixed =
train(
    model=net_fixed_lr,
    optimizer=optimizer_fixed,
    n_epochs=20,             # число эпох, как и выше
    train_loader=train_loader,
    val_loader=val_loader
)

Epoch 0
 train loss: 1.783916661464754, train acc: 0.37053163363249253
 val loss: 1.5711337515648376, val acc: 0.4496897163543295

Epoch 1
 train loss: 1.4955791605456001, train acc: 0.4735121768396995
 val loss: 1.4203817418281068, val acc: 0.49789450358837206

Epoch 2
 train loss: 1.3851296973620735, train acc: 0.5099813920706238
 val loss: 1.3354440531832106, val acc: 0.5349512412192974

Epoch 3
 train loss: 1.3162379527440673, train acc: 0.5369425764685357
 val loss: 1.2878100354620752, val acc: 0.5499335106382979

Epoch 4
 train loss: 1.2686303367998308, train acc: 0.5536163162705667
 val loss: 1.2396502114356833, val acc: 0.5627437944107867

Epoch 5
 train loss: 1.2249772104092447, train acc: 0.5722079851291734
```

```
 val loss: 1.2001482339615517, val acc: 0.5796985816448292

Epoch 6
 train loss: 1.1929136477615105, train acc: 0.5843521481894052
 val loss: 1.1673513833512652, val acc: 0.591688829787234

Epoch 7
 train loss: 1.1552030911611246, train acc: 0.5954924589755113
 val loss: 1.1462752098732807, val acc: 0.5999556739279565

Epoch 8
 train loss: 1.1227362753050418, train acc: 0.6094443719609346
 val loss: 1.1180266773447078, val acc: 0.6091312058428501

Epoch 9
 train loss: 1.0928159502132284, train acc: 0.6210743667220721
 val loss: 1.0815169968503586, val acc: 0.6277703901554675

Epoch 10
 train loss: 1.0596181721966271, train acc: 0.6326105054161448
 val loss: 1.0851333729764248, val acc: 0.6224955675449777

Epoch 11
 train loss: 1.038743169586663, train acc: 0.6396089057817755
 val loss: 1.0422708049733589, val acc: 0.6376773050490846

Epoch 12
 train loss: 1.0165180536685086, train acc: 0.6466725973150194
 val loss: 1.029409066666948, val acc: 0.641156914893617

Epoch 13
 train loss: 0.99808814974546, train acc: 0.655992915904936
 val loss: 1.0019103435759849, val acc: 0.6561835106382978

Epoch 14
 train loss: 0.9787810576681245, train acc: 0.6626158918933415
 val loss: 0.9994603895126505, val acc: 0.6531914893617021

Epoch 15
 train loss: 0.9634825472857877, train acc: 0.668406405113297
 val loss: 0.970779483622693, val acc: 0.662876773134191

Epoch 16
 train loss: 0.9539930064237968, train acc: 0.6695571624600691
 val loss: 0.979974373604389, val acc: 0.6603058510638298

Epoch 17
 train loss: 0.941192568765044, train acc: 0.6744091146824782
 val loss: 0.975410969713901, val acc: 0.6594636526513606
```

```
Epoch 18
 train loss: 0.93549297594719, train acc: 0.676657580370441
 val loss: 0.9529539734759229, val acc: 0.6674867021276596

Epoch 19
 train loss: 0.9244376789281311, train acc: 0.6804036628829496
 val loss: 0.9606405598052005, val acc: 0.6673093973322117
```

```python
test_loss, test_acc = test(net_fixed_lr, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")
```

```
Test Loss: 0.96%
Test Accuracy: 0.66%
```

Точность сильно упало, как и должно было быть

Задание 3.1 (1.5 балла). Изучите, какие еще способы уменьшения длины шага представлены в torch.optim.lr_scheduler. Выберите несколько из них, объясните, как они устроены, и обучите модель с ними. Удалось ли добиться улучшения качества на тестовой выборке?

```python
"""
Попробую StepLR. Он уменьшает скорость обучения на определенный коэф
каждые step_size эпох
"""
net = BasicBlockNet().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5,
gamma=0.1)
train_loss_log, train_acc_log, val_loss_log, val_acc_log = train(
    model=net,
    optimizer=optimizer,
    n_epochs=20,
    train_loader=train_loader,
    val_loader=val_loader,
    scheduler=scheduler
)
```

```
Epoch 0
 train loss: 1.5301858602973617, train acc: 0.4574342191328296
 val loss: 1.3103928766352064, val acc: 0.5407801420130628

Epoch 1
 train loss: 1.2162379593038473, train acc: 0.573925959889588
 val loss: 1.1611407820214616, val acc: 0.5962765957446808

Epoch 2
 train loss: 1.0831429479980816, train acc: 0.6238410812845178
```

```
 val loss: 1.0546985009883312, val acc: 0.6410682624958931

Epoch 3
 train loss: 1.0193673513925054, train acc: 0.6478437581925331
 val loss: 1.0210476233604107, val acc: 0.6482712765957447

Epoch 4
 train loss: 0.9730204699026384, train acc: 0.6648357926818527
 val loss: 0.9946882803389366, val acc: 0.6561391845662543

Epoch 5
 train loss: 0.8465631381358264, train acc: 0.708425829244924
 val loss: 0.8918167877704539, val acc: 0.6941046100981692

Epoch 6
 train loss: 0.8269208684917778, train acc: 0.7160159311303272
 val loss: 0.8801337475472308, val acc: 0.699468085106383

Epoch 7
 train loss: 0.8212207802035038, train acc: 0.7179338600779365
 val loss: 0.8785634342660296, val acc: 0.6986258866939139

Epoch 8
 train loss: 0.8124967874513029, train acc: 0.7204516519140282
 val loss: 0.874403150284544, val acc: 0.7005984042553192

Epoch 9
 train loss: 0.8060937342957561, train acc: 0.7209576587354461
 val loss: 0.8717615487727713, val acc: 0.7016179079705096

Epoch 10
 train loss: 0.7891412772905674, train acc: 0.728580406204656
 val loss: 0.862265282235247, val acc: 0.7043439718002968

Epoch 11
 train loss: 0.7879702040765577, train acc: 0.7291639462467522
 val loss: 0.8606881367399337, val acc: 0.7066710994598714

Epoch 12
 train loss: 0.7864101390734015, train acc: 0.730143314196815
 val loss: 0.8603919046990415, val acc: 0.7064494680851063

Epoch 13
 train loss: 0.7847974832140767, train acc: 0.7312695874374571
 val loss: 0.8602598000080027, val acc: 0.7058732271194458

Epoch 14
 train loss: 0.7833282525840145, train acc: 0.7313756856665533
 val loss: 0.8587102403032019, val acc: 0.7063386526513606
```

```
Epoch 15
 train loss: 0.7814008797326515, train acc: 0.7318449661108432
 val loss: 0.8589118744464631, val acc: 0.7064051420130628

Epoch 16
 train loss: 0.7823976157897133, train acc: 0.7313960890883283
 val loss: 0.8583493770437037, val acc: 0.7057402483960415

Epoch 17
 train loss: 0.7820949442207922, train acc: 0.7319061765941013
 val loss: 0.8592408530255582, val acc: 0.7058067377577437

Epoch 18
 train loss: 0.7828240618518327, train acc: 0.7302902194873943
 val loss: 0.8597467853667888, val acc: 0.7076684398854033

Epoch 19
 train loss: 0.7805861570281564, train acc: 0.7308533559987488
 val loss: 0.8591143963184763, val acc: 0.7058067377577437
```

```
test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")
```

```
Test Loss: 0.87%
Test Accuracy: 0.70%
```

```
"""
Попробую CosineAnnealingLR. Так lr меняется по косинусоиде: плавно
уменьшается и может снова возрастать
"""
net = BasicBlockNet().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=20)
train_loss_log, train_acc_log, val_loss_log, val_acc_log = train(
    model=net,
    optimizer=optimizer,
    n_epochs=20,
    train_loader=train_loader,
    val_loader=val_loader,
    scheduler=scheduler
)
```

```
Epoch 0
 train loss: 1.4764457865868452, train acc: 0.4770746279024337
 val loss: 1.335786426320989, val acc: 0.5362145390916377

Epoch 1
 train loss: 1.16857524011863, train acc: 0.591105706513037
 val loss: 1.108017977247847, val acc: 0.6207446808510638
```

```
Epoch 2
 train loss: 1.0693213752244464, train acc: 0.632924719310112
 val loss: 1.1942332369215944, val acc: 0.5930407803109352

Epoch 3
 train loss: 1.0065474744470726, train acc: 0.6506431183387854
 val loss: 1.0290361784874125, val acc: 0.6407801420130628

Epoch 4
 train loss: 0.9610704853722121, train acc: 0.6661375360035591
 val loss: 0.9596770740569907, val acc: 0.668439716481148

Epoch 5
 train loss: 0.9317161150998564, train acc: 0.6763637700072155
 val loss: 0.9935091995178384, val acc: 0.6489804965384462

Epoch 6
 train loss: 0.9025886454355564, train acc: 0.6867287477148081
 val loss: 0.9283259698685179, val acc: 0.6775709220703612

Epoch 7
 train loss: 0.8698466751431634, train acc: 0.6993177070042333
 val loss: 0.965915406511185, val acc: 0.6615026595744681

Epoch 8
 train loss: 0.8436068283356521, train acc: 0.7063569144313471
 val loss: 0.9341451997452593, val acc: 0.6827127659574468

Epoch 9
 train loss: 0.8290798636851406, train acc: 0.7129472447269796
 val loss: 0.8999330320256822, val acc: 0.6872562057160316

Epoch 10
 train loss: 0.803193877394936, train acc: 0.7206516062101891
 val loss: 0.8872460464213757, val acc: 0.6940602837724889

Epoch 11
 train loss: 0.7835484602006741, train acc: 0.7278622030340122
 val loss: 0.8932960586344942, val acc: 0.6909352837724888

Epoch 12
 train loss: 0.7649687491561639, train acc: 0.7351136067013854
 val loss: 0.8494441367210226, val acc: 0.7086214539852548

Epoch 13
 train loss: 0.7422302118494044, train acc: 0.7440340168079467
 val loss: 0.8928580253682238, val acc: 0.6941932624958931

Epoch 14
```

```
 train loss: 0.7285593326937346, train acc: 0.7480167798847778
 val loss: 0.8374484708968629, val acc: 0.7131870569066798

Epoch 15
 train loss: 0.712224946978759, train acc: 0.7540235701383141
 val loss: 0.8238839957308262, val acc: 0.7169991135597229

Epoch 16
 train loss: 0.6966982072515505, train acc: 0.760862823269267
 val loss: 0.8340395819633565, val acc: 0.7157358156873824

Epoch 17
 train loss: 0.6870146085400904, train acc: 0.7643885479548731
 val loss: 0.8100489837058047, val acc: 0.7218528369639782

Epoch 18
 train loss: 0.6763651926186228, train acc: 0.7660249087012883
 val loss: 0.8093144664104949, val acc: 0.7230496454746165

Epoch 19
 train loss: 0.6717294282407603, train acc: 0.7682529708585094
 val loss: 0.8057762361587362, val acc: 0.7235815603682335
```

```
test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")
```

```
Test Loss: 0.82%
Test Accuracy: 0.72%
```

```
"""
Попробую OneCycleLR. Меняет lr от малого к большому и обратно
"""
net = BasicBlockNet().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
scheduler = optim.lr_scheduler.OneCycleLR(optimizer, max_lr=0.1,
steps_per_epoch=len(train_loader), epochs=20)
train_loss_log, train_acc_log, val_loss_log, val_acc_log = train(
    model=net,
    optimizer=optimizer,
    n_epochs=20,
    train_loader=train_loader,
    val_loader=val_loader,
    scheduler=scheduler
)
```

```
Epoch 0
 train loss: 1.5527983651082737, train acc: 0.4461837294332724
 val loss: 1.378259952524875, val acc: 0.504144503588372
```

```
Epoch 1
 train loss: 1.2702747326228474, train acc: 0.5517595978062158
 val loss: 1.2888460557511512, val acc: 0.5505097518575952

Epoch 2
 train loss: 1.1430292569741034, train acc: 0.5968105250346377
 val loss: 1.0776779337132232, val acc: 0.6250886526513607

Epoch 3
 train loss: 1.0572762786797238, train acc: 0.631068000845761
 val loss: 1.037229437270063, val acc: 0.637876773134191

Epoch 4
 train loss: 1.0160412405918895, train acc: 0.6464236745886655
 val loss: 1.0369043114337515, val acc: 0.6395168441407224

Epoch 5
 train loss: 0.9699010309615127, train acc: 0.6606530753088607
 val loss: 1.0149409923147648, val acc: 0.6414228723404255

Epoch 6
 train loss: 0.9395238482538045, train acc: 0.673768444841479
 val loss: 0.9375949879910084, val acc: 0.6755540781832756

Epoch 7
 train loss: 0.9112292120617966, train acc: 0.6805260838494658
 val loss: 0.931787953224588, val acc: 0.6816710994598714

Epoch 8
 train loss: 0.8895849216136897, train acc: 0.6911563072387653
 val loss: 0.9385803783193548, val acc: 0.6751329787234043

Epoch 9
 train loss: 0.8802813796064317, train acc: 0.6960286628829496
 val loss: 0.908600819618144, val acc: 0.6871675531914894

Epoch 10
 train loss: 0.8566826723611333, train acc: 0.7046756660480604
 val loss: 0.9057179346997687, val acc: 0.6873670212765958

Epoch 11
 train loss: 0.8535556200435436, train acc: 0.7042227083411905
 val loss: 0.8747739241478291, val acc: 0.6977171986661059

Epoch 12
 train loss: 0.826815294064377, train acc: 0.7188112105922245
 val loss: 0.8711366044714096, val acc: 0.7027925531914894

Epoch 13
 train loss: 0.8176264791754525, train acc: 0.718484754536226
```

```
 val loss: 0.8792217746694038, val acc: 0.6969414893617021

Epoch 14
 train loss: 0.8079394773874684, train acc: 0.721663619853025
 val loss: 0.8452978915356575, val acc: 0.710815602921425

Epoch 15
 train loss: 0.797754635486132, train acc: 0.7231041068786677
 val loss: 0.8748536260838204, val acc: 0.7027703901554676

Epoch 16
 train loss: 0.793527649787052, train acc: 0.7267032842095635
 val loss: 0.826990258693695, val acc: 0.7188164893617022

Epoch 17
 train loss: 0.7793055891554795, train acc: 0.7344851789134512
 val loss: 0.8506421200772549, val acc: 0.7125664893617021

Epoch 18
 train loss: 0.7767857180761899, train acc: 0.7317103029822954
 val loss: 0.8537225281938593, val acc: 0.7072030143534883

Epoch 19
 train loss: 0.7613677501242601, train acc: 0.7380721141912819
 val loss: 0.811569200170801, val acc: 0.7238918441407224
```

```
test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")
```

```
Test Loss: 0.83%
Test Accuracy: 0.72%
```

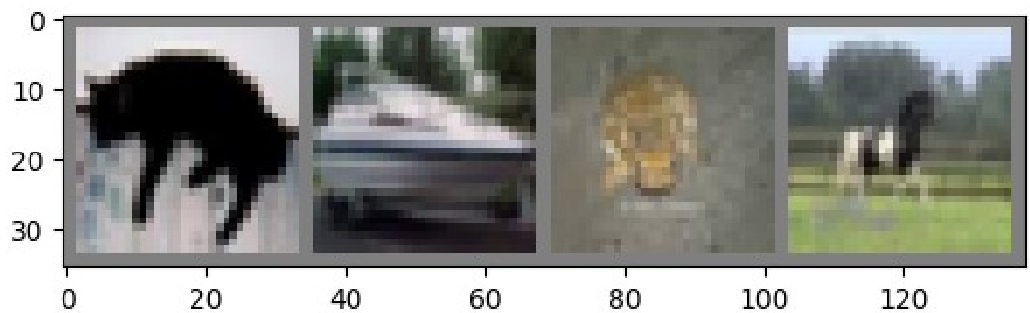
Лучше всего себя показал OneCycleLR в этом задании, но если смотреть на всё, то MultiStepLR выигрывает.

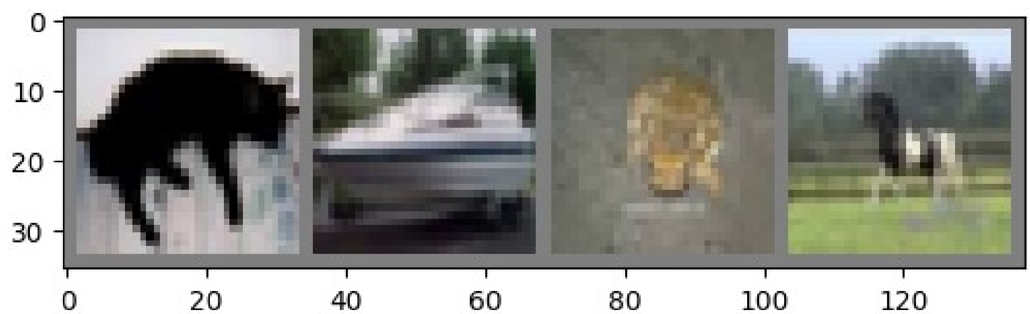
4. Аугментации данных (2 балла)

Еще одной стандартной техникой, применяющейся в глубинном обучении, а особенно часто в компьютерном зрении, являются аугментации данных. Суть аугментаций состоит в том, что мы можем некоторым синтетическим образом видоизменять объекты обучающей выборки, тем самым расширяя ее, а также делая итоговую модель более устойчивой к таким изменениям.

Простейшая аугментация, которую можно применить к картинкам — разворот картинки по горизонтальной оси. То есть при обучении модели с вероятностью 0.5 мы будем разворачивать картинку из обучающей выборки.

```
dataiter = iter(train_loader)
images, labels = next(dataiter)

imshow(torchvision.utils.make_grid(images[:4]))

imshow(torchvision.utils.make_grid(transforms.functional.hflip(images[
:4])))
```





Наиболее удобным способом работы с аугментациями в PyTorch является их задание в списке transforms, который затем передается в загрузчик данных. Обучим нашу сеть, применяя горизонтальные повороты:

```
transform = transforms.Compose(
    [
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]
)

train_loader, val_loader, test_loader = get_cifar10_data(
    batch_size=64, transform_train=transform
)

net = BasicBlockNet().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[10,
15], gamma=0.1)
```

```
tr_loss_log, tr_acc_log, val_loss_log, val_acc_log = train(
    net, optimizer, 20, train_loader, val_loader, scheduler
)
```

```
Epoch 0
 train loss: 1.47574619695517, train acc: 0.4812002154765225
 val loss: 1.2024817745736305, val acc: 0.5813829787234043

Epoch 1
 train loss: 1.1697046641239954, train acc: 0.5932195090724519
 val loss: 1.2202834235860947, val acc: 0.5695478723404256

Epoch 2
 train loss: 1.0684003543374307, train acc: 0.628182946001154
 val loss: 1.1962097609296758, val acc: 0.589029255319149

Epoch 3
 train loss: 1.0139897169317145, train acc: 0.6497290415563357
 val loss: 0.9877897901737943, val acc: 0.6563386526513607

Epoch 4
 train loss: 0.9753334014463686, train acc: 0.6608163032278936
 val loss: 0.9764566132362853, val acc: 0.6634308510638298

Epoch 5
 train loss: 0.9455718250989478, train acc: 0.6704916427732603
 val loss: 0.9751426633368148, val acc: 0.6598625886947551

Epoch 6
 train loss: 0.9291987395155801, train acc: 0.6765759663564413
 val loss: 0.9242675857341036, val acc: 0.6833998228641267

Epoch 7
 train loss: 0.912326129306605, train acc: 0.6846435100548446
 val loss: 0.9697775815395598, val acc: 0.6702792553191489

Epoch 8
 train loss: 0.8918589478872812, train acc: 0.6903850549535734
 val loss: 0.9415779415597307, val acc: 0.6792331561129143

Epoch 9
 train loss: 0.8780838824822218, train acc: 0.693841407678245
 val loss: 0.9319775558532553, val acc: 0.6802304965384462

Epoch 10
 train loss: 0.7848241655547614, train acc: 0.7277275399054645
 val loss: 0.8163163050692133, val acc: 0.7192597518575953

Epoch 11
 train loss: 0.766037070664017, train acc: 0.735444143441739
 val loss: 0.8102733636156042, val acc: 0.7220301420130628
```

```
Epoch 12
 train loss: 0.762621571747433, train acc: 0.736264364157124
 val loss: 0.8039837843560158, val acc: 0.7251551420130629

Epoch 13
 train loss: 0.7563821675027746, train acc: 0.7386393314959581
 val loss: 0.8103648510385066, val acc: 0.722406914893617

Epoch 14
 train loss: 0.7509217372324175, train acc: 0.7391290154709894
 val loss: 0.802845560870272, val acc: 0.7263741135597229

Epoch 15
 train loss: 0.7377962059251133, train acc: 0.7450215461267196
 val loss: 0.7933235098706916, val acc: 0.7293882978723404

Epoch 16
 train loss: 0.7378903682968516, train acc: 0.7463436929791676
 val loss: 0.7934817971067226, val acc: 0.7289893617021277

Epoch 17
 train loss: 0.737279828276887, train acc: 0.744474732352785
 val loss: 0.7891865581908124, val acc: 0.7287234042553191

Epoch 18
 train loss: 0.7366923140651347, train acc: 0.7467476822558443
 val loss: 0.790211171926336, val acc: 0.7284574468085107

Epoch 19
 train loss: 0.7360092815577003, train acc: 0.7473883520968435
 val loss: 0.7938388471907758, val acc: 0.7275265957446808
```

Посчитайте точность на тестовой выборке:

```
test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")

Test Loss: 0.81%
Test Accuracy: 0.72%
```

Задание 4.1 (2 балла). Изучите, какие еще способы аугментаций изображений представлены в torchvision.transforms. Выберите несколько из них, объясните, как они устроены, и обучите модель с ними (по отдельности и вместе). Удалось ли добиться улучшения качества на тестовой выборке?

```
"""
RandomVerticalFlip - С вероятностью 0.5 переворачивает изображение по
```

```
вертикали
"""
transform = transforms.Compose(
    [
        transforms.RandomVerticalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]
)

train_loader, val_loader, test_loader = get_cifar10_data(
    batch_size=64, transform_train=transform
)

net = BasicBlockNet().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[10,
15], gamma=0.1)
tr_loss_log, tr_acc_log, val_loss_log, val_acc_log = train(
    net, optimizer, 20, train_loader, val_loader, scheduler
)
```

```
Epoch 0
 train loss: 1.6574516738790481, train acc: 0.4023978193673617
 val loss: 1.43232293230422, val acc: 0.48238031914893614

Epoch 1
 train loss: 1.3911623620245968, train acc: 0.5034685949956695
 val loss: 1.4961104032841135, val acc: 0.4786790781832756

Epoch 2
 train loss: 1.2915629682418852, train acc: 0.5424147950012026
 val loss: 1.3337535642563028, val acc: 0.5403147164811479

Epoch 3
 train loss: 1.2173416262570858, train acc: 0.5697718073070812
 val loss: 1.2379948430872978, val acc: 0.5631427305809995

Epoch 4
 train loss: 1.1718965969312343, train acc: 0.587955406204656
 val loss: 1.1933608813488736, val acc: 0.5797429079705096

Epoch 5
 train loss: 1.1434111421897164, train acc: 0.5984060786106032
 val loss: 1.1800613408393048, val acc: 0.5770611702127659

Epoch 6
 train loss: 1.1221139891927814, train acc: 0.6045108057245258
 val loss: 1.1663645204077375, val acc: 0.5925088654173182
```

```
Epoch 7
 train loss: 1.100636741360973, train acc: 0.6136638157110545
 val loss: 1.133533762617314, val acc: 0.6056294327086591

Epoch 8
 train loss: 1.086550572345636, train acc: 0.6170018281535649
 val loss: 1.1034245980546828, val acc: 0.614937943346957

Epoch 9
 train loss: 1.0681216547868368, train acc: 0.6236370463040019
 val loss: 1.1752597793619683, val acc: 0.5949468085106383

Epoch 10
 train loss: 0.9773350824385718, train acc: 0.658922858495381
 val loss: 0.9944710323151121, val acc: 0.6568040781832756

Epoch 11
 train loss: 0.9544589542383686, train acc: 0.666876142595978
 val loss: 0.9898236013473348, val acc: 0.6589317377577437

Epoch 12
 train loss: 0.9466249312953495, train acc: 0.6705487725721635
 val loss: 0.9790339358309482, val acc: 0.6605496454746165

Epoch 13
 train loss: 0.9435856672483999, train acc: 0.6728829329584809
 val loss: 0.980957362246006, val acc: 0.6616134752618505

Epoch 14
 train loss: 0.9369147815477695, train acc: 0.6714546879859031
 val loss: 0.9768274035859615, val acc: 0.6660682624958931

Epoch 15
 train loss: 0.9237757114867188, train acc: 0.6778654675161163
 val loss: 0.9682830229718634, val acc: 0.667154255319149

Epoch 16
 train loss: 0.922686630672685, train acc: 0.6792977931730491
 val loss: 0.9658344109007653, val acc: 0.6667109930768926

Epoch 17
 train loss: 0.9194459772415091, train acc: 0.6797629930419503
 val loss: 0.9646937519945997, val acc: 0.6663342199427016

Epoch 18
 train loss: 0.9204566644358243, train acc: 0.6799955929219178
 val loss: 0.9672591807994436, val acc: 0.6635195037151905

Epoch 19
 train loss: 0.9220033950517993, train acc: 0.6777675307646965
```

```
 val loss: 0.9642079160568562, val acc: 0.6685283688788718
```

```python
test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")
```

```
Test Loss: 0.98%
Test Accuracy: 0.66%
```

```python
"""
RandomRotation(degrees) - поворачивает изображение на случайный угол.
Подберу возможные значения degrees
"""
degree_values = [0, 10, 20, 30, 45]
results = {}

for degrees in degree_values:
    print(f"Training with RandomRotation({degrees})")

    transform = transforms.Compose([
        transforms.RandomRotation(degrees),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])

    train_loader, val_loader, test_loader = get_cifar10_data(
        batch_size=64, transform_train=transform
    )

    net = BasicBlockNet().to(device)
    optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
    scheduler = optim.lr_scheduler.MultiStepLR(optimizer,
milestones=[10, 15], gamma=0.1)

    tr_loss_log, tr_acc_log, val_loss_log, val_acc_log = train(
        net, optimizer, 20, train_loader, val_loader, scheduler
    )

    test_loss, test_acc = test(net, test_loader)
    results[degrees] = [test_loss, test_acc]
```

```
Training with RandomRotation(0)
Epoch 0
 train loss: 1.4706973873719, train acc: 0.4797760511882998
 val loss: 1.1779870824610934, val acc: 0.5894281914893617

Epoch 1
 train loss: 1.1627644865996658, train acc: 0.593737757946935
 val loss: 1.2886434270980511, val acc: 0.5627659574468085
```

```
Epoch 2
 train loss: 1.0480259359427737, train acc: 0.6357199987720092
 val loss: 1.1880480248877343, val acc: 0.5912455674181594

Epoch 3
 train loss: 0.9909449227331325, train acc: 0.6564744385112574
 val loss: 1.022571464041446, val acc: 0.6469636526513607

Epoch 4
 train loss: 0.9488466262163584, train acc: 0.6728829329584809
 val loss: 0.972342460713488, val acc: 0.6588874114320633

Epoch 5
 train loss: 0.916697858020636, train acc: 0.6846435100548446
 val loss: 0.9627609506566474, val acc: 0.6630762412192973

Epoch 6
 train loss: 0.898497750488888, train acc: 0.6884875294277394
 val loss: 0.9399797543566277, val acc: 0.6775930851063829

Epoch 7
 train loss: 0.8763467466264582, train acc: 0.6962653434472721
 val loss: 0.9532974070691048, val acc: 0.6767065603682335

Epoch 8
 train loss: 0.858766891457263, train acc: 0.7007133063930044
 val loss: 0.944990417551487, val acc: 0.679410461161999

Epoch 9
 train loss: 0.8428262853535263, train acc: 0.7052673674588665
 val loss: 0.9713051451013444, val acc: 0.66875

Epoch 10
 train loss: 0.7407124270795686, train acc: 0.744164599143173
 val loss: 0.8297185645458546, val acc: 0.7188829787234042

Epoch 11
 train loss: 0.7222922681890415, train acc: 0.7522484656879628
 val loss: 0.8226695882513168, val acc: 0.722406914893617

Epoch 12
 train loss: 0.7163872755314794, train acc: 0.7535991774398623
 val loss: 0.8195551981317236, val acc: 0.721875

Epoch 13
 train loss: 0.7104427564950486, train acc: 0.7532115109006056
 val loss: 0.8217475324235064, val acc: 0.7228280143534883

Epoch 14
 train loss: 0.7068131189568605, train acc: 0.7559170149797931
```

```
 val loss: 0.8195371306957082, val acc: 0.723404255319149

Epoch 15
 train loss: 0.6910926810784139, train acc: 0.7618136263198783
 val loss: 0.8104290635027784, val acc: 0.7250664893617021

Epoch 16
 train loss: 0.6894213097949786, train acc: 0.7621686472753284
 val loss: 0.8095546839085032, val acc: 0.7268617021276595

Epoch 17
 train loss: 0.6885152823637347, train acc: 0.7616667211382655
 val loss: 0.8111489739823848, val acc: 0.7267287234042553

Epoch 18
 train loss: 0.6877466164515703, train acc: 0.7638580570273251
 val loss: 0.8105732845499161, val acc: 0.7258643617021276

Epoch 19
 train loss: 0.6863139012940841, train acc: 0.7631316923790048
 val loss: 0.8109001658064254, val acc: 0.7269946808510638

Training with RandomRotation(10)
Epoch 0
 train loss: 1.516676207674051, train acc: 0.46121294733811125
 val loss: 1.2736045959148001, val acc: 0.5537455674181593

Epoch 1
 train loss: 1.2486681624348028, train acc: 0.5584437842778793
 val loss: 1.2493258980994528, val acc: 0.554188829787234

Epoch 2
 train loss: 1.1435235671195076, train acc: 0.5992589449097949
 val loss: 1.246274598354989, val acc: 0.5747562057160317

Epoch 3
 train loss: 1.0845536079659541, train acc: 0.6202908723838151
 val loss: 1.1270873318327235, val acc: 0.6156693263256804

Epoch 4
 train loss: 1.0301943532509603, train acc: 0.6384703904880684
 val loss: 1.044733223762918, val acc: 0.6337322696726373

Epoch 5
 train loss: 0.9977637183949541, train acc: 0.6499167537994315
 val loss: 1.0258906785477984, val acc: 0.639472517815042

Epoch 6
 train loss: 0.9759898829721664, train acc: 0.6585637568555759
 val loss: 0.9900530168350706, val acc: 0.6520168441407224
```

```
Epoch 7
 train loss: 0.957812852898725, train acc: 0.6659130974923017
 val loss: 1.0126246972286954, val acc: 0.6524601063829787

Epoch 8
 train loss: 0.9351892453878845, train acc: 0.6748335074898966
 val loss: 0.9959463888026299, val acc: 0.6527039007937655

Epoch 9
 train loss: 0.9203955971344734, train acc: 0.6794242949328867
 val loss: 1.056945626279141, val acc: 0.6392952127659575

Epoch 10
 train loss: 0.8203399893461677, train acc: 0.7158812680017795
 val loss: 0.8774057157496189, val acc: 0.6955230497299356

Epoch 11
 train loss: 0.805731820068586, train acc: 0.7218554126715093
 val loss: 0.871132197532248, val acc: 0.6973625888215734

Epoch 12
 train loss: 0.7997735123093865, train acc: 0.7228347807305386
 val loss: 0.8737036664435204, val acc: 0.6987367021276596

Epoch 13
 train loss: 0.7919774054936125, train acc: 0.7251648603235346
 val loss: 0.8678706722056612, val acc: 0.6970744680851064

Epoch 14
 train loss: 0.7886054792068559, train acc: 0.724295671286923
 val loss: 0.8693095136196055, val acc: 0.7014406029214251

Epoch 15
 train loss: 0.7746739274731937, train acc: 0.7312695874374571
 val loss: 0.8539740798321176, val acc: 0.7037234042553191

Epoch 16
 train loss: 0.7733601533843568, train acc: 0.7313756856665533
 val loss: 0.8551720647101707, val acc: 0.7061170212765957

Epoch 17
 train loss: 0.7757563001494957, train acc: 0.7281968203497543
 val loss: 0.857105756059606, val acc: 0.7039671986661059

Epoch 18
 train loss: 0.7735853647946004, train acc: 0.7297066793363317
 val loss: 0.8596148561924062, val acc: 0.7042331561129144

Epoch 19
```

```
 train loss: 0.7729699929845835, train acc: 0.730702370132781
 val loss: 0.854025813873778, val acc: 0.7043661348363186

Training with RandomRotation(20)
Epoch 0
 train loss: 1.5685648684963667, train acc: 0.44034832857644535
 val loss: 1.354257038806347, val acc: 0.5212322695458189

Epoch 1
 train loss: 1.3280493986454045, train acc: 0.5284873662308758
 val loss: 1.2631131030143576, val acc: 0.5543661348363187

Epoch 2
 train loss: 1.2293664490501013, train acc: 0.5648015148025108
 val loss: 1.2185594553643084, val acc: 0.5737367021276596

Epoch 3
 train loss: 1.1727327117100494, train acc: 0.5885103813473006
 val loss: 1.1966330452168241, val acc: 0.5843528369639782

Epoch 4
 train loss: 1.1165817358158188, train acc: 0.6080773374716151
 val loss: 1.116284712578388, val acc: 0.6032579787234043

Epoch 5
 train loss: 1.085235089227094, train acc: 0.6181117785478205
 val loss: 1.1403754449905232, val acc: 0.5939716312479466

Epoch 6
 train loss: 1.0650846616005767, train acc: 0.6240451162650337
 val loss: 1.0335387498774427, val acc: 0.6404033688788718

Epoch 7
 train loss: 1.0439269751909663, train acc: 0.6336878102265938
 val loss: 1.0689279317855835, val acc: 0.6282579787234043

Epoch 8
 train loss: 1.0240762706648496, train acc: 0.6411514103521595
 val loss: 1.0670943825802905, val acc: 0.6267508866939139

Epoch 9
 train loss: 1.0086045745739771, train acc: 0.6452117067586352
 val loss: 1.0492300725997763, val acc: 0.6348404255319149

Epoch 10
 train loss: 0.9104739022211358, train acc: 0.6831540545990088
 val loss: 0.9489236489255377, val acc: 0.6722739361702128

Epoch 11
 train loss: 0.897419281698886, train acc: 0.6878999085923218
```

```
 val loss: 0.9312070724811959, val acc: 0.6784574468085106

Epoch 12
 train loss: 0.8913740584902615, train acc: 0.6877244385112574
 val loss: 0.9415268172609045, val acc: 0.6717863476022761

Epoch 13
 train loss: 0.8826777306836527, train acc: 0.6919275594149908
 val loss: 0.9340785891451734, val acc: 0.6802969859001484

Epoch 14
 train loss: 0.880879512761585, train acc: 0.6926457626946012
 val loss: 0.9260318335066451, val acc: 0.6763297872340426

Epoch 15
 train loss: 0.8670128181821903, train acc: 0.6971059676933986
 val loss: 0.9112623978168406, val acc: 0.6860150710065314

Epoch 16
 train loss: 0.8666011509664315, train acc: 0.6990524615404593
 val loss: 0.9182973600448446, val acc: 0.6807624114320633

Epoch 17
 train loss: 0.8658268144405302, train acc: 0.6972528728750114
 val loss: 0.9193766649733198, val acc: 0.6806737590343395

Epoch 18
 train loss: 0.8635754754164101, train acc: 0.6966081222406907
 val loss: 0.9158316526007145, val acc: 0.682535461161999

Epoch 19
 train loss: 0.8641592507715417, train acc: 0.6953920737263054
 val loss: 0.9094095765276158, val acc: 0.686724290949233

Training with RandomRotation(30)
Epoch 0
 train loss: 1.6069919254487786, train acc: 0.4265596435100548
 val loss: 1.3827009789487148, val acc: 0.5065159574468086

Epoch 1
 train loss: 1.3890623710709036, train acc: 0.5043459455099577
 val loss: 1.3288495997165113, val acc: 0.5241578015875309

Epoch 2
 train loss: 1.3195354124527745, train acc: 0.5303685689103232
 val loss: 1.3126810137261735, val acc: 0.5416445037151905

Epoch 3
 train loss: 1.2555721878351633, train acc: 0.555885185489271
 val loss: 1.287776989125191, val acc: 0.5518395390916377
```

```
Epoch 4
 train loss: 1.2027003859907008, train acc: 0.5740647035935244
 val loss: 1.175245957678937, val acc: 0.5853945037151905

Epoch 5
 train loss: 1.1755920865418052, train acc: 0.5853437581925331
 val loss: 1.1689246263909847, val acc: 0.5839095744680851

Epoch 6
 train loss: 1.1512442679030368, train acc: 0.5937989684301931
 val loss: 1.1062441686366467, val acc: 0.607535461161999

Epoch 7
 train loss: 1.1263740125914161, train acc: 0.6018257051760797
 val loss: 1.1490312779203373, val acc: 0.5990913122258288

Epoch 8
 train loss: 1.1049529442621542, train acc: 0.6095382280279993
 val loss: 1.1637019205600658, val acc: 0.5867242909492331

Epoch 9
 train loss: 1.089682859302221, train acc: 0.615973491773309
 val loss: 1.1245965201803978, val acc: 0.5996453901554676

Epoch 10
 train loss: 0.9935134940217138, train acc: 0.6507043288220435
 val loss: 1.0093619483582517, val acc: 0.6463430851063829

Epoch 11
 train loss: 0.9787254735582271, train acc: 0.657013090916482
 val loss: 0.9994472080088677, val acc: 0.6472296100981692

Epoch 12
 train loss: 0.970748082702295, train acc: 0.6581312026794476
 val loss: 0.9970464008919736, val acc: 0.6480053191489362

Epoch 13
 train loss: 0.9646171094948457, train acc: 0.6598410159621858
 val loss: 0.9891738376718886, val acc: 0.6536125888215735

Epoch 14
 train loss: 0.9628820033570312, train acc: 0.6621670149797931
 val loss: 0.9871531085765108, val acc: 0.6552304965384463

Epoch 15
 train loss: 0.949743867356337, train acc: 0.6635381301534677
 val loss: 0.9787349201263266, val acc: 0.6544991135597229

Epoch 16
```

```
 train loss: 0.9490137631975972, train acc: 0.6684839384429415
 val loss: 0.9871742431153643, val acc: 0.6561170212765958

Epoch 17
 train loss: 0.9472450776335526, train acc: 0.6689246539877799
 val loss: 0.9921283376977799, val acc: 0.6530141845662543

Epoch 18
 train loss: 0.945303664455902, train acc: 0.6676065879286535
 val loss: 0.9854333436235468, val acc: 0.6566710994598713

Epoch 19
 train loss: 0.9478369868433671, train acc: 0.6664109428360432
 val loss: 0.9744331096081024, val acc: 0.6588209220703612

Training with RandomRotation(45)
Epoch 0
 train loss: 1.681659529806273, train acc: 0.39763972321201724
 val loss: 1.505280183223968, val acc: 0.4633200355032657

Epoch 1
 train loss: 1.4780595657812394, train acc: 0.4705495886654479
 val loss: 1.4001298929782624, val acc: 0.5058067377577437

Epoch 2
 train loss: 1.4017470857125096, train acc: 0.5023790481321555
 val loss: 1.4037262698437305, val acc: 0.5056294327086591

Epoch 3
 train loss: 1.3406081251949888, train acc: 0.5278670998116516
 val loss: 1.3783931904650748, val acc: 0.5178856382978724

Epoch 4
 train loss: 1.2909921059006966, train acc: 0.5424841669621372
 val loss: 1.2863669872283936, val acc: 0.5506205675449777

Epoch 5
 train loss: 1.265783593140311, train acc: 0.5493234199841236
 val loss: 1.2787012863666454, val acc: 0.5462322695458189

Epoch 6
 train loss: 1.2398442660868714, train acc: 0.5596190259487145
 val loss: 1.2091287714369754, val acc: 0.5745124114320633

Epoch 7
 train loss: 1.2146358572804732, train acc: 0.5690087163905992
 val loss: 1.2664384557845745, val acc: 0.5637189718002968

Epoch 8
 train loss: 1.1971007933128468, train acc: 0.576313169281487
```

```
 val loss: 1.2037150154722498, val acc: 0.5753546100981692

Epoch 9
 train loss: 1.1822182713740486, train acc: 0.5818996475209266
 val loss: 1.1898266929261228, val acc: 0.5826462765957446

Epoch 10
 train loss: 1.0852244675050489, train acc: 0.6169895861004998
 val loss: 1.0914980170574593, val acc: 0.6165558510638298

Epoch 11
 train loss: 1.0768630237640366, train acc: 0.6189891290621087
 val loss: 1.076784722348477, val acc: 0.6231382978723404

Epoch 12
 train loss: 1.0604835329151676, train acc: 0.6274402586154135
 val loss: 1.0767392323372211, val acc: 0.621099290949233

Epoch 13
 train loss: 1.05925298183884, train acc: 0.6259467224752228
 val loss: 1.0702626421096477, val acc: 0.6258421986661059

Epoch 14
 train loss: 1.0548731792561532, train acc: 0.6293500261943127
 val loss: 1.0593098097659173, val acc: 0.6310726952045522

Epoch 15
 train loss: 1.0428364078567058, train acc: 0.6323452599523708
 val loss: 1.0578154457376359, val acc: 0.6294547872340426

Epoch 16
 train loss: 1.0412415738951353, train acc: 0.635046683238237
 val loss: 1.061212781388709, val acc: 0.6301418441407224

Epoch 17
 train loss: 1.0404802098788553, train acc: 0.6353404936014625
 val loss: 1.0629374121097808, val acc: 0.6287455675449777

Epoch 18
 train loss: 1.0367670550642762, train acc: 0.6365279773253627
 val loss: 1.0538571167499462, val acc: 0.6322695035883721

Epoch 19
 train loss: 1.0403385667957814, train acc: 0.634540676416819
 val loss: 1.0497170143939079, val acc: 0.6345301418862445
```

```
for degrees, (loss, acc) in results.items():
    print(f"degrees = {degrees}, Test Loss: {loss:.2f}%, Test 
Accuracy: {acc:.2f}%")
```

```
degrees = 0, Test Loss: 0.82%, Test Accuracy: 0.72%
degrees = 10, Test Loss: 0.84%, Test Accuracy: 0.71%
degrees = 20, Test Loss: 0.86%, Test Accuracy: 0.70%
degrees = 30, Test Loss: 0.90%, Test Accuracy: 0.68%
degrees = 45, Test Loss: 0.97%, Test Accuracy: 0.66%
```

```
"""
ColorJitter(brightness, contrast, saturation, hue) - меняет яркость,
контрастность, насыщенность и оттенок
Подберем параметры
"""
params_to_test = [
    (0.1, 0.1, 0.1, 0.05),
    (0.2, 0.2, 0.2, 0.1),
    (0.5, 0.5, 0.5, 0.3),
    (0.2, 0.1, 0.1, 0.05),
    (0.1, 0.2, 0.5, 0.1),
]
results = {}

for params in params_to_test:
    b, c, s, h = params
    transform = transforms.Compose(
        [
            transforms.ColorJitter(brightness=b, contrast=c,
saturation=s, hue=h),
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ]
    )

    train_loader, val_loader, test_loader = get_cifar10_data(
        batch_size=64, transform_train=transform
    )

    net = BasicBlockNet().to(device)
    optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
    scheduler = optim.lr_scheduler.MultiStepLR(optimizer,
milestones=[10, 15], gamma=0.1)
    tr_loss_log, tr_acc_log, val_loss_log, val_acc_log = train(
        net, optimizer, 20, train_loader, val_loader, scheduler
    )
    test_loss, test_acc = test(net, test_loader)
    results[str(params)] = [test_loss, test_acc]
```

```
Epoch 0
 train loss: 1.4780182518078597, train acc: 0.4765767824497258
 val loss: 1.2147804133435514, val acc: 0.5770611702127659

Epoch 1
```

```
 train loss: 1.1568672716944484, train acc: 0.5974389528225719
 val loss: 1.1407827473701315, val acc: 0.6030585106382979

Epoch 2
 train loss: 1.0542555306467758, train acc: 0.635626142595978
 val loss: 1.1772290942516732, val acc: 0.5934840425531915

Epoch 3
 train loss: 0.9931770686257692, train acc: 0.6568784277879341
 val loss: 1.0009161378474947, val acc: 0.6540558510638298

Epoch 4
 train loss: 0.9511943694878322, train acc: 0.6719484526452896
 val loss: 1.0071004154834342, val acc: 0.6473625888215735

Epoch 5
 train loss: 0.9252889931092968, train acc: 0.6795181509999514
 val loss: 0.9944061210814943, val acc: 0.6533909574468085

Epoch 6
 train loss: 0.911775024107213, train acc: 0.6838844999316841
 val loss: 0.9642606402965302, val acc: 0.6659796099713509

Epoch 7
 train loss: 0.8879960168977107, train acc: 0.692041819012797
 val loss: 1.0435208977536952, val acc: 0.652593085106383

Epoch 8
 train loss: 0.8669237325025869, train acc: 0.6983220162077839
 val loss: 0.9382745778307001, val acc: 0.6809175531914894

Epoch 9
 train loss: 0.8567585255371805, train acc: 0.7026761230864517
 val loss: 0.9866557773123397, val acc: 0.6624778369639782

Epoch 10
 train loss: 0.7546718284024617, train acc: 0.7394758749487631
 val loss: 0.842752828877023, val acc: 0.7130097518575952

Epoch 11
 train loss: 0.7370767181490632, train acc: 0.74576423351246
 val loss: 0.8385878663113777, val acc: 0.7149379433469569

Epoch 12
 train loss: 0.7311348107113916, train acc: 0.749004309312517
 val loss: 0.8340652718188915, val acc: 0.7177304965384463

Epoch 13
 train loss: 0.7245602338579934, train acc: 0.7523912901852206
 val loss: 0.8324902151493316, val acc: 0.7177304965384463
```

```
Epoch 14
 train loss: 0.7187694863057006, train acc: 0.7525259533137683
 val loss: 0.8328854043433007, val acc: 0.7168439718002969

Epoch 15
 train loss: 0.7054378081392326, train acc: 0.7578879930419503
 val loss: 0.8257328392343318, val acc: 0.7214539007937655

Epoch 16
 train loss: 0.705956066657246, train acc: 0.7583980805477233
 val loss: 0.8201356449025743, val acc: 0.7221852837724888

Epoch 17
 train loss: 0.7009993718572682, train acc: 0.7599936341236888
 val loss: 0.8230844038598081, val acc: 0.7177969859001484

Epoch 18
 train loss: 0.7010025007742197, train acc: 0.7583654348549503
 val loss: 0.8215525124935393, val acc: 0.7210549646235527

Epoch 19
 train loss: 0.7012030495912981, train acc: 0.7585939540505627
 val loss: 0.8211456989988367, val acc: 0.7215868795171697

Epoch 0
 train loss: 1.513709056529964, train acc: 0.46381643387255767
 val loss: 1.2510801340671296, val acc: 0.5572473404255319

Epoch 1
 train loss: 1.1910890981527744, train acc: 0.5840175307646965
 val loss: 1.2056227458284257, val acc: 0.5860372340425531

Epoch 2
 train loss: 1.0957205873084896, train acc: 0.6212498368031364
 val loss: 1.1657402649838873, val acc: 0.6027703901554676

Epoch 3
 train loss: 1.0393475734337592, train acc: 0.6412126208354176
 val loss: 1.0676944687011394, val acc: 0.6332668441407224

Epoch 4
 train loss: 0.9965891774873193, train acc: 0.6565356491034819
 val loss: 1.041095822922727, val acc: 0.6415558510638298

Epoch 5
 train loss: 0.9665314156350967, train acc: 0.6635993406367259
 val loss: 0.99688288125586, val acc: 0.6510859929500742

Epoch 6
```

```
 train loss: 0.9514400211505087, train acc: 0.6710098916477434
 val loss: 1.0042313174998507, val acc: 0.6494015957446808

Epoch 7
 train loss: 0.9305035461019554, train acc: 0.6783673936531792
 val loss: 1.0161430746951001, val acc: 0.658222517815042

Epoch 8
 train loss: 0.91002862468716, train acc: 0.6891037151626085
 val loss: 0.9613570213317871, val acc: 0.6670877659574468

Epoch 9
 train loss: 0.8975495180656531, train acc: 0.6882345260170304
 val loss: 1.0308201817756004, val acc: 0.6566267731341909

Epoch 10
 train loss: 0.7939392105970784, train acc: 0.7277275399054645
 val loss: 0.868858795216743, val acc: 0.7033687944107867

Epoch 11
 train loss: 0.7831667085779215, train acc: 0.7297352442357832
 val loss: 0.8618415089363748, val acc: 0.7050310284533399

Epoch 12
 train loss: 0.7755967461975662, train acc: 0.7353584487433843
 val loss: 0.8631306351499355, val acc: 0.7080230497299357

Epoch 13
 train loss: 0.7697993610633139, train acc: 0.735974634369287
 val loss: 0.8620160939845632, val acc: 0.7077570922831271

Epoch 14
 train loss: 0.7637461190685711, train acc: 0.7375253004173474
 val loss: 0.8558644421557162, val acc: 0.7073359930768927

Epoch 15
 train loss: 0.7519282631616924, train acc: 0.7419814247317899
 val loss: 0.8504124108781206, val acc: 0.7123005319148936

Epoch 16
 train loss: 0.7536289806884647, train acc: 0.7414264494801788
 val loss: 0.8432830985556258, val acc: 0.7117242909492331

Epoch 17
 train loss: 0.7491051417926984, train acc: 0.7429526313131427
 val loss: 0.8463648808763382, val acc: 0.7114140071767442

Epoch 18
 train loss: 0.7470308111289301, train acc: 0.7443033429560757
 val loss: 0.8438747427564987, val acc: 0.7135195037151905
```

```
Epoch 19
 train loss: 0.7476325329849446, train acc: 0.7440176939615601
 val loss: 0.8425608830249056, val acc: 0.7144503547790203

Epoch 0
 train loss: 1.7449411561327834, train acc: 0.3773831288165105
 val loss: 1.4227810905334797, val acc: 0.5030363476022761

Epoch 1
 train loss: 1.392063911152932, train acc: 0.5170614064502542
 val loss: 1.4660834642166787, val acc: 0.49986702127659577

Epoch 2
 train loss: 1.2591600576092163, train acc: 0.5693841407678245
 val loss: 1.350609952845472, val acc: 0.5437278369639782

Epoch 3
 train loss: 1.2026670668853048, train acc: 0.5875677396653993
 val loss: 1.1651562853062407, val acc: 0.5993572695458189

Epoch 4
 train loss: 1.1533183168885477, train acc: 0.6050943458755884
 val loss: 1.1841431379318237, val acc: 0.5977171986661058

Epoch 5
 train loss: 1.1176543845990894, train acc: 0.6176751436873371
 val loss: 1.197576804617618, val acc: 0.5828900710065315

Epoch 6
 train loss: 1.1038046720041, train acc: 0.6241634565471949
 val loss: 1.1476268291473388, val acc: 0.6036569148936171

Epoch 7
 train loss: 1.0806230665124967, train acc: 0.6299050013369573
 val loss: 1.138932077428128, val acc: 0.6093971631628402

Epoch 8
 train loss: 1.0649317865833723, train acc: 0.6368299491662648
 val loss: 1.1103681863622463, val acc: 0.6228280143534883

Epoch 9
 train loss: 1.0461956565515216, train acc: 0.6398251828153565
 val loss: 1.1393349122493825, val acc: 0.6084663122258288

Epoch 10
 train loss: 0.9430021703570155, train acc: 0.6807464416763682
 val loss: 0.9891208547226926, val acc: 0.6631870569066799

Epoch 11
```

```
 train loss: 0.9254670904802884, train acc: 0.6868715722120659
 val loss: 0.9725050951572175, val acc: 0.6636524824385948

Epoch 12
 train loss: 0.9223309321638871, train acc: 0.6861207234576154
 val loss: 0.9788365409729328, val acc: 0.6632978723404256

Epoch 13
 train loss: 0.913874135374805, train acc: 0.6911440650767339
 val loss: 0.973333216981685, val acc: 0.6666445037151905

Epoch 14
 train loss: 0.9107307654412181, train acc: 0.69042586190609
 val loss: 0.965677498756571, val acc: 0.668218085106383

Epoch 15
 train loss: 0.8968081017734799, train acc: 0.694396382929856
 val loss: 0.9574314302586494, val acc: 0.6727171986661059

Epoch 16
 train loss: 0.8995233555162628, train acc: 0.6955879473381112
 val loss: 0.9525639873869876, val acc: 0.6732712765957447

Epoch 17
 train loss: 0.8933717697804328, train acc: 0.6954165579414019
 val loss: 0.9514725687655997, val acc: 0.674689716481148

Epoch 18
 train loss: 0.8915975453430818, train acc: 0.697860897132204
 val loss: 0.9536554075301962, val acc: 0.6734707446808511

Epoch 19
 train loss: 0.892778107300537, train acc: 0.6988035388141053
 val loss: 0.9574739468858597, val acc: 0.6727171986661059

Epoch 0
 train loss: 1.4902870749860622, train acc: 0.4719778336380256
 val loss: 1.209699612475456, val acc: 0.5734929078436912

Epoch 1
 train loss: 1.165449416070795, train acc: 0.5926318882370344
 val loss: 1.2153038585439642, val acc: 0.5798537234042553

Epoch 2
 train loss: 1.0674378777116917, train acc: 0.6325207300334351
 val loss: 1.1262683404252885, val acc: 0.6130540781832756

Epoch 3
 train loss: 1.005638793138089, train acc: 0.6510185427160106
 val loss: 1.0319710843106533, val acc: 0.6438164893617021
```

```
Epoch 4
 train loss: 0.9618354222892189, train acc: 0.6696999869573269
 val loss: 0.9993014274759495, val acc: 0.6560283688788718

Epoch 5
 train loss: 0.9288648726517366, train acc: 0.6770615696471177
 val loss: 0.9556813110696508, val acc: 0.6654698582405739

Epoch 6
 train loss: 0.9146271086480108, train acc: 0.6838804192473291
 val loss: 0.9332719597410648, val acc: 0.6771941489361702

Epoch 7
 train loss: 0.8924285923650104, train acc: 0.6924294855520537
 val loss: 1.0316279449361436, val acc: 0.6532579787234043

Epoch 8
 train loss: 0.8746394534106664, train acc: 0.6988729107750399
 val loss: 0.9286450030955863, val acc: 0.6822473404255319

Epoch 9
 train loss: 0.8580909198119392, train acc: 0.7029332071815156
 val loss: 0.9603729899893416, val acc: 0.6753989361702127

Epoch 10
 train loss: 0.7544372398848943, train acc: 0.7410795900024052
 val loss: 0.8348586423599974, val acc: 0.715093085106383

Epoch 11
 train loss: 0.7379794556109517, train acc: 0.746853780375974
 val loss: 0.8262390868460878, val acc: 0.7203014186088075

Epoch 12
 train loss: 0.7321611301227723, train acc: 0.7484860604380341
 val loss: 0.8282480198018094, val acc: 0.7201684398854032

Epoch 13
 train loss: 0.7256593382772624, train acc: 0.7516281992687386
 val loss: 0.8252474120322694, val acc: 0.7199689718002968

Epoch 14
 train loss: 0.720258582226755, train acc: 0.7517465396598663
 val loss: 0.8271905070923744, val acc: 0.7167774824385947

Epoch 15
 train loss: 0.7084657432719166, train acc: 0.7569575934131141
 val loss: 0.8178456094670803, val acc: 0.7226950356300841

Epoch 16
```

```
 train loss: 0.7074579009190338, train acc: 0.7564842322844691
 val loss: 0.812306476527072, val acc: 0.7234264186088075

Epoch 17
 train loss: 0.7037329785457694, train acc: 0.7579206386257569
 val loss: 0.8136510859144495, val acc: 0.7249556739279565

Epoch 18
 train loss: 0.7013342205087707, train acc: 0.7584307261315298
 val loss: 0.8131662419501772, val acc: 0.7236258866939139

Epoch 19
 train loss: 0.7026268377273567, train acc: 0.7594794659335609
 val loss: 0.8112093557702734, val acc: 0.7256427305809995

Epoch 0
 train loss: 1.5250827655931714, train acc: 0.4623106555703353
 val loss: 1.258714462594783, val acc: 0.5584663120990104

Epoch 1
 train loss: 1.1969560694215067, train acc: 0.5824627840322811
 val loss: 1.2368128358049595, val acc: 0.5690159574468086

Epoch 2
 train loss: 1.0966715657950756, train acc: 0.6202908723838151
 val loss: 1.1621009436059506, val acc: 0.5990248227373083

Epoch 3
 train loss: 1.0364436587860204, train acc: 0.6412207822041276
 val loss: 1.0532431762269203, val acc: 0.638497340425532

Epoch 4
 train loss: 0.9951732305765588, train acc: 0.6547034147452611
 val loss: 1.0168297219783702, val acc: 0.6485593973322118

Epoch 5
 train loss: 0.9625284465836869, train acc: 0.6674515213783306
 val loss: 1.0005063308046218, val acc: 0.6505097518575952

Epoch 6
 train loss: 0.9471555204016635, train acc: 0.6720137438129028
 val loss: 1.0063075304031373, val acc: 0.6499113476022761

Epoch 7
 train loss: 0.9177192512770239, train acc: 0.6824521742747515
 val loss: 0.9937597186007399, val acc: 0.6593085106382979

Epoch 8
 train loss: 0.8990988861054345, train acc: 0.690429942590445
 val loss: 0.9638062588712002, val acc: 0.665846631374765
```

```
Epoch 9
 train loss: 0.8848935936116213, train acc: 0.6924580504515053
 val loss: 1.0119473355881712, val acc: 0.6588652483960415

Epoch 10
 train loss: 0.7843985673060583, train acc: 0.7315878820157792
 val loss: 0.855086743324361, val acc: 0.7065824468085107

Epoch 11
 train loss: 0.7689288305408122, train acc: 0.7343545965782249
 val loss: 0.8521326267972905, val acc: 0.7086657803109352

Epoch 12
 train loss: 0.7627862697327594, train acc: 0.7382026966354747
 val loss: 0.8487565984117224, val acc: 0.7142287234042554

Epoch 13
 train loss: 0.7561455061710295, train acc: 0.7414672565416619
 val loss: 0.8504322154724852, val acc: 0.7116578015875309

Epoch 14
 train loss: 0.7511771322386356, train acc: 0.7410673479493401
 val loss: 0.8417396726760459, val acc: 0.7162898936170212

Epoch 15
 train loss: 0.7381954432296579, train acc: 0.7489063724521309
 val loss: 0.838630439119136, val acc: 0.7138741135597229

Epoch 16
 train loss: 0.738087098875255, train acc: 0.746335531501491
 val loss: 0.8297304196560636, val acc: 0.7177304965384463

Epoch 17
 train loss: 0.7353405863645962, train acc: 0.7474740467951982
 val loss: 0.833250691535625, val acc: 0.7166445037151905

Epoch 18
 train loss: 0.7351855519593743, train acc: 0.7470251697726834
 val loss: 0.8288438374691821, val acc: 0.716622340425532

Epoch 19
 train loss: 0.7337800252917044, train acc: 0.7486819339408735
 val loss: 0.8303554631294088, val acc: 0.7184175531914894


for params, (loss, acc) in results.items():
    print(f"params = {params}, Test Loss: {loss:.2f}%, Test Accuracy: 
{acc:.2f}%")
```

```
params = (0.1, 0.1, 0.1, 0.05), Test Loss: 0.82%, Test Accuracy: 0.73%
params = (0.2, 0.2, 0.2, 0.1), Test Loss: 0.83%, Test Accuracy: 0.72%
params = (0.5, 0.5, 0.5, 0.3), Test Loss: 0.89%, Test Accuracy: 0.69%
params = (0.2, 0.1, 0.1, 0.05), Test Loss: 0.81%, Test Accuracy: 0.72%
params = (0.1, 0.2, 0.5, 0.1), Test Loss: 0.82%, Test Accuracy: 0.72%
```

```python
"""
RandomGrayscale(p=0.1) - переводит изображение в оттенки серого с
заданной вероятностью
"""
transform = transforms.Compose(
    [
        transforms.RandomGrayscale(p=0.1),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]
)

train_loader, val_loader, test_loader = get_cifar10_data(
    batch_size=64, transform_train=transform
)

net = BasicBlockNet().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[10,
15], gamma=0.1)
tr_loss_log, tr_acc_log, val_loss_log, val_acc_log = train(
    net, optimizer, 20, train_loader, val_loader, scheduler
)
```

```
Epoch 0
 train loss: 1.4747867451289454, train acc: 0.47942919171052617
 val loss: 1.2119913248305625, val acc: 0.5747340425531915

Epoch 1
 train loss: 1.164226971988922, train acc: 0.5963045182132198
 val loss: 1.2188036008084073, val acc: 0.5762411348363187

Epoch 2
 train loss: 1.059858493111905, train acc: 0.6325533756172417
 val loss: 1.208918119237778, val acc: 0.5846631206096486

Epoch 3
 train loss: 1.0091937092801988, train acc: 0.6511858514283649
 val loss: 1.0554466937450653, val acc: 0.6404033688788718

Epoch 4
 train loss: 0.9596223910722261, train acc: 0.6676188300906849
 val loss: 1.021777170262438, val acc: 0.6468085106382979
```

```
Epoch 5
 train loss: 0.9255593892534211, train acc: 0.680901508226691
 val loss: 1.0261248426234468, val acc: 0.6462544327086591

Epoch 6
 train loss: 0.9104073151156061, train acc: 0.6839497910992973
 val loss: 0.9141882269940478, val acc: 0.6841976952045522

Epoch 7
 train loss: 0.8921682049522016, train acc: 0.6889119222351576
 val loss: 1.0011442552221583, val acc: 0.6595301420130628

Epoch 8
 train loss: 0.864072321864761, train acc: 0.7005623205270366
 val loss: 0.9492383079325899, val acc: 0.6758865249917863

Epoch 9
 train loss: 0.8542671619865096, train acc: 0.707132247400894
 val loss: 1.0206419320816689, val acc: 0.6534352837724888

Epoch 10
 train loss: 0.7537212157162277, train acc: 0.7400267694070527
 val loss: 0.8362952411174774, val acc: 0.7129432624958931

Epoch 11
 train loss: 0.7336792578644901, train acc: 0.7491104074326468
 val loss: 0.834407621114812, val acc: 0.7131648936170213

Epoch 12
 train loss: 0.7270939291286295, train acc: 0.7506365892656106
 val loss: 0.8316900945724325, val acc: 0.7138076241980208

Epoch 13
 train loss: 0.7234742461654342, train acc: 0.7516934905453182
 val loss: 0.8321522630275564, val acc: 0.7135638297872341

Epoch 14
 train loss: 0.7172858504207305, train acc: 0.7542643314959581
 val loss: 0.8312628393477582, val acc: 0.7150265957446809

Epoch 15
 train loss: 0.7035480839569782, train acc: 0.7595937255313671
 val loss: 0.8206670738281088, val acc: 0.7189716313747649

Epoch 16
 train loss: 0.7035323723033752, train acc: 0.7604669953613002
 val loss: 0.8214154298001147, val acc: 0.7199024824385948

Epoch 17
 train loss: 0.7002553105136377, train acc: 0.7621237595294904
```

```
 val loss: 0.8185260046035685, val acc: 0.7209884752618506

Epoch 18
 train loss: 0.7016619834538134, train acc: 0.7612219248001056
 val loss: 0.8183917732948952, val acc: 0.7219414893617021

Epoch 19
 train loss: 0.7006305539520828, train acc: 0.760879146006687
 val loss: 0.8185247540473938, val acc: 0.7204787234042553


test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")

Test Loss: 0.81%
Test Accuracy: 0.72%
```

По-отдельности Accuracy особо не изменилось. Оно почти везде равно 72% при лучших параметрах. Объединю способы аргументаций изображений. Для более быстрой работы в этот раз не буду перебирать каждый параметр, а возьму тот, где результат лучше. А для RandomRotation сделаю случайный поворот от 0 до 20.

```
transform = transforms.Compose(
    [
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(degrees=(0, 20)),
        transforms.ColorJitter(brightness=0.1, contrast=0.1,
saturation=0.1, hue=0.05),
        transforms.RandomGrayscale(p=0.1),

        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]
)

train_loader, val_loader, test_loader = get_cifar10_data(
    batch_size=64, transform_train=transform
)

net = BasicBlockNet().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[10,
15], gamma=0.1)
tr_loss_log, tr_acc_log, val_loss_log, val_acc_log = train(
    net, optimizer, 20, train_loader, val_loader, scheduler
)
```

```
Epoch 0
 train loss: 1.7609466272908547, train acc: 0.3593178702555801
 val loss: 1.6098424343352622, val acc: 0.41312056741815933

Epoch 1
 train loss: 1.5473078781333658, train acc: 0.4424662118202154
 val loss: 1.5373180582168255, val acc: 0.45352393617021275

Epoch 2
 train loss: 1.44269846584069, train acc: 0.4845382280279993
 val loss: 1.4461071156440897, val acc: 0.4968528369639782

Epoch 3
 train loss: 1.3703879759778053, train acc: 0.5111647950012026
 val loss: 1.3868611371263544, val acc: 0.5147163122258288

Epoch 4
 train loss: 1.3260471996267273, train acc: 0.5293075868372943
 val loss: 1.376766805953168, val acc: 0.4977615249917862

Epoch 5
 train loss: 1.295532605656758, train acc: 0.5399826978634654
 val loss: 1.3763425263952702, val acc: 0.49563386529049974

Epoch 6
 train loss: 1.2861165399961105, train acc: 0.5416721076154621
 val loss: 1.2840679323419613, val acc: 0.5376994680851064

Epoch 7
 train loss: 1.264879765523635, train acc: 0.5527430465496
 val loss: 1.2352286584833836, val acc: 0.5735372340425532

Epoch 8
 train loss: 1.2430562029373058, train acc: 0.559553734672135
 val loss: 1.2380704943169938, val acc: 0.5589317377577436

Epoch 9
 train loss: 1.2273346826406895, train acc: 0.5658176091297017
 val loss: 1.3068524900903093, val acc: 0.5415558510638298

Epoch 10
 train loss: 1.1400926711354438, train acc: 0.5986998890827953
 val loss: 1.1328845112881762, val acc: 0.6039893617021277

Epoch 11
 train loss: 1.1216600163545207, train acc: 0.606783755627585
 val loss: 1.1138920319841263, val acc: 0.6113918441407225

Epoch 12
 train loss: 1.1151038370141164, train acc: 0.6079100287592607
 val loss: 1.1070946094837595, val acc: 0.6091312057160316
```

```
Epoch 13
 train loss: 1.1126691573716605, train acc: 0.6070204361919075
 val loss: 1.1115633814892871, val acc: 0.6126773050490846

Epoch 14
 train loss: 1.1036544225333595, train acc: 0.6121213111406706
 val loss: 1.111688945648518, val acc: 0.6185505319148936

Epoch 15
 train loss: 1.0974916121641505, train acc: 0.6159898146196955
 val loss: 1.1024098386155798, val acc: 0.6145611702127659

Epoch 16
 train loss: 1.0947648749926844, train acc: 0.6170018281535649
 val loss: 1.0945931011057914, val acc: 0.6148049646235527

Epoch 17
 train loss: 1.0847953816218612, train acc: 0.6186912380145281
 val loss: 1.09390994234288, val acc: 0.6209884752618505

Epoch 18
 train loss: 1.0930580919577828, train acc: 0.6137984787306359
 val loss: 1.0967476910733163, val acc: 0.6163563829787234

Epoch 19
 train loss: 1.0919291694159918, train acc: 0.6153858710468578
 val loss: 1.0856355408404736, val acc: 0.6217863476022761
```

```
test_loss, test_acc = test(net, test_loader)
print(f"Test Loss: {test_loss:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")
```

```
Test Loss: 1.07%
Test Accuracy: 0.63%
```

Как видно, лучше не объединять аргументации из `torchvision.transforms`, а взять что-то одно и просто попытаться подобрать наилучшие параметры тк после обучения они почти одинаковые. Наилучшим образом себя показал transforms.ColorJitter(brightness=b, contrast=c, saturation=s, hue=h) с параметрами (0.1, 0.1, 0.1, 0.05): Test Loss = 0.82%, Test Accuracy = 0.73%. Она даже немного улучшила модель.

Бонус. Логирование в wandb (1 балл)

На практике специалиста по глубинному обучению часто встречаются ситуации, когда нейросеть учится на каком-то удаленном сервере. И обычно вам хочется отслеживать прогресс обучения, особенно когда время обучения модели исчисляется днями или неделями. Для таких целей существует несколько инструментов. Вероятно, самый популярный из них — wandb.

Ваша задача состоит в том, чтобы разобраться как им пользоваться, и повторить задания 2.1 и 2.2 с его использованием. Обучение вы можете запускать в этом же ноутбуке, но теперь вам необходимо через wandb логировать значения функции потерь и точности на обучающей выборке и на валидационной. Результатом работы должны быть ваш код и публичная ссылка на страничку с графиками, идентичными графикам в задании 2.2.

Если вас смущает, что WandB грозится забанить вас, то можете разобраться с любым его аналогом и приложить ссылку на аналог.

```
# <your code here>
```