Оглавление

Отчет. Бабак Артем. 201 группа

Функции и классы для обучения, тестирования, получения датасета

Код взят с семинара с некоторыми изменениями

Датасет

In [ ]:

```
class MyDataset(Dataset):
    TEST_SIZE = 0.20
    SPLIT_RANDOM_SEED = 42

    def __init__(self, root, final_test=False, final_train=False, train=Tru
        super().__init__()
        self.root = root
        self.transform = transform
        self.train = train
        self.final_test = final_test
        self.final_train = final_train
        self.load_to_ram = load_to_ram
        self.to_tensor = T.ToTensor()
        self.all_files = []
        self.all_labels = []
        self.images = []
        self.classes = []

        if self.final_test:
            self.data_root = self.root + "/test"
        else:
            self.data_root = self.root + "/trainval"
            self.labels_root = self.root + "/labels.csv"
            all_labels = list(pd.read_csv(self.labels_root)['Label'])


        all_files = sorted(os.listdir(self.data_root))

        if not self.final_test and not self.final_train:
            train_files, test_files, train_labels, test_labels = train_test
                                                                 test_size=self.TEST_

        if self.final_test:
            self.files = all_files
        elif self.final_train:
            self.files = all_files
            self.labels = all_labels
        elif self.train:
            self.files = train_files
            self.labels = train_labels
        else:
            self.files = test_files
            self.labels = test_labels

        if not self.final_test:
            self.classes = sorted(pd.unique(all_labels))

        if self.load_to_ram:
            self.images = self._load_images(self.files)

    def _load_images(self, image_files):
        images = []
        for filename in image_files:
            image = Image.open(os.path.join(self.data_root, filename)).conv
            images += [image]

        return images

    def __len__(self):
```

```
        return len(self.files)

    def __getitem__(self, item):
        if not self.final_test:
            label = self.labels[item]
        filename = self.files[item]
        image = Image.open(os.path.join(self.data_root, filename)).convert(

        if self.transform is not None:
            image = self.transform(image)

        if self.final_test:
            return image
        else:
            return image, label
```

Рисование графиков

In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import clear_output
from tqdm.notebook import tqdm


sns.set_style('whitegrid')
plt.rcParams.update({'font.size': 15})


def plot_losses(train_losses, test_losses, train_accuracies, test_accuracie
    clear_output()
    fig, axs = plt.subplots(1, 2, figsize=(13, 4))
    axs[0].plot(range(1, len(train_losses) + 1), train_losses, label='train
    axs[0].plot(range(1, len(test_losses) + 1), test_losses, label='test')
    axs[0].set_ylabel('loss')

    axs[1].plot(range(1, len(train_accuracies) + 1), train_accuracies, labe
    axs[1].plot(range(1, len(test_accuracies) + 1), test_accuracies, label=
    axs[1].set_ylabel('accuracy')

    for ax in axs:
        ax.set_xlabel('epoch')
        ax.legend()

    plt.show()
```

Обучение + валидация

In [ ]:

```python
def training_epoch(model, optimizer, criterion, train_loader, tqdm_desc):
    train_loss, train_accuracy = 0.0, 0.0
    model.train()
    for images, labels in tqdm(train_loader, desc=tqdm_desc):
        images = images.to(device)  # images: batch_size x num_channels x h
        labels = labels.to(device)  # labels: batch_size

        optimizer.zero_grad()
        logits = model(images)  # logits: batch_size x num_classes
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * images.shape[0]
        train_accuracy += (logits.argmax(dim=1) == labels).sum().item()

    train_loss /= len(train_loader.dataset)
    train_accuracy /= len(train_loader.dataset)
    return train_loss, train_accuracy


@torch.no_grad()
def validation_epoch(model, criterion, test_loader, tqdm_desc):
    test_loss, test_accuracy = 0.0, 0.0
    model.eval()
    for images, labels in tqdm(test_loader, desc=tqdm_desc):
        images = images.to(device)  # images: batch_size x num_channels x h
        labels = labels.to(device)  # labels: batch_size
        logits = model(images)  # logits: batch_size x num_classes
        loss = criterion(logits, labels)

        test_loss += loss.item() * images.shape[0]
        test_accuracy += (logits.argmax(dim=1) == labels).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy /= len(test_loader.dataset)
    return test_loss, test_accuracy


def train(model, optimizer, scheduler, criterion, train_loader, test_loader
    train_losses, train_accuracies = [], []
    test_losses, test_accuracies = [], []

    for epoch in range(1, num_epochs + 1):
        train_loss, train_accuracy = training_epoch(
            model, optimizer, criterion, train_loader,
            tqdm_desc=f'Training {epoch}/{num_epochs}'
        )
        test_loss, test_accuracy = validation_epoch(
            model, criterion, test_loader,
            tqdm_desc=f'Validating {epoch}/{num_epochs}'
        )

        if scheduler is not None:
            scheduler.step()

        train_losses += [train_loss]
        train_accuracies += [train_accuracy]
        test_losses += [test_loss]
```

```
        test_accuracies += [test_accuracy]
        plot_losses(train_losses, test_losses, train_accuracies, test_accur

    return train_losses, test_losses, train_accuracies, test_accuracies
```

Обучения на всей выборке + получение предсказаний

In [ ]:
```
def train_only(model, optimizer, scheduler, criterion, train_loader, num_ep
    train_losses, train_accuracies = [], []
    test_losses, test_accuracies = [], []

    for epoch in range(1, num_epochs + 1):
        train_loss, train_accuracy = training_epoch(
            model, optimizer, criterion, train_loader,
            tqdm_desc=f'Training {epoch}/{num_epochs}'
        )

        if scheduler is not None:
            scheduler.step()

        train_losses += [train_loss]
        train_accuracies += [train_accuracy]
        plot_train_loss(train_losses, train_accuracies)

    return train_losses, train_accuracies

def plot_train_loss(train_losses, train_accuracies):
    clear_output()
    fig, axs = plt.subplots(1, 2, figsize=(13, 4))
    axs[0].plot(range(1, len(train_losses) + 1), train_losses, label='train
    axs[0].set_ylabel('loss')

    axs[1].plot(range(1, len(train_accuracies) + 1), train_accuracies, labe
    axs[1].set_ylabel('accuracy')

    for ax in axs:
        ax.set_xlabel('epoch')
        ax.legend()

    plt.show()

def make_predictions(model, test_loader):
    predictions = []
    model.eval()
    for images in tqdm(test_loader):
        images = images.to(device)  # images: batch_size x num_channels x h
        logits = model(images)  # logits: batch_size x num_classes
        predictions += (torch.argmax(logits, dim=1).tolist())

    return predictions
```

Модель, чтобы пробить 10%

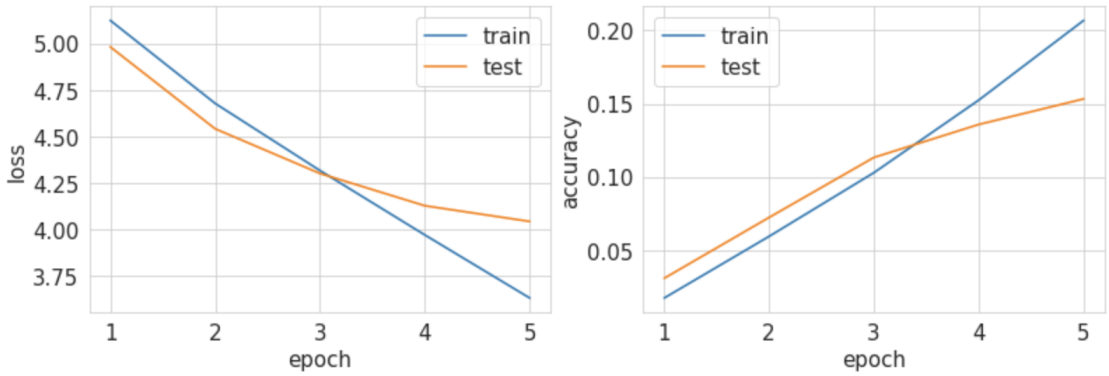
Код взят с семинара

Модель + параметры

In [ ]:
```
model = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5), #60*60
    nn.LeakyReLU(),
    nn.MaxPool2d(kernel_size=2), #30*30

    nn.Conv2d(in_channels=16, out_channels=16, kernel_size=5), # 26x26
    nn.LeakyReLU(),
    nn.MaxPool2d(kernel_size=2), # 13x13

    nn.Flatten(),
    nn.Linear(13 * 13 * 16, 256),
    nn.LeakyReLU(),
    nn.Linear(256, 200))
```

In [ ]:
```
test_transform = T.Compose([
    T.ToTensor()
])

train_transform = T.Compose([
    T.ToTensor()
])

train_dataset = MyDataset(root=root, final_test=False, final_train=False, t
val_dataset = MyDataset(root=root, final_test=False, final_train=False, tra
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, pin_m
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, pin_memo

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

num_epochs = 5
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
criterion = torch.nn.CrossEntropyLoss()
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, num_epoch

train_losses, test_losses, train_accuracies, test_accuracies = train(
    model, optimizer, scheduler, criterion, train_loader, val_loader, num_e
)
```

Результаты



ResNet18 из PyTorch

Модель + параметры

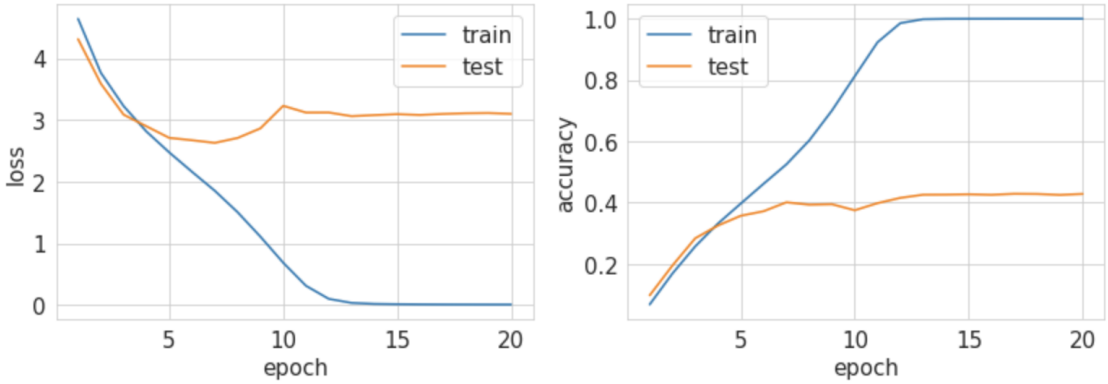
Во-первых, ResNet из PyTorch преполагает картинки 224x224, так что сделаем сначала Resize(256), а затем CenterCrop(224), как часто делается и как показывали на лекции.

Во-вторых, нормализуем стандартными значениямию. Так рекомендуется в документации PyTorch и так же рекомендовали на лекциию.

В-третьих, расписание, оптимизатор, гиперпараметры оставим из предыдущей модели. Лишь увеличим количество эпох до 20, так как нам уже надо не 10%, а как можно лучше

In [ ]:
```
test_transform = T.Compose([
        T.Resize(256),
        T.CenterCrop(224),
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_transform = T.Compose([
        T.Resize(256),
        T.CenterCrop(224),
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_dataset = MyDataset(root=root, final_test=False, final_train=False, t
val_dataset = MyDataset(root=root, final_test=False, final_train=False, tra
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, pin_m
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, pin_memo

model = models.resnet18()
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

num_epochs = 20
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
criterion = torch.nn.CrossEntropyLoss()
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, num_epoch

train_losses, test_losses, train_accuracies, test_accuracies = train(
    model, optimizer, scheduler, criterion, train_loader, val_loader, num_e
)
```

Результаты



Получили хороший результат немногим больше 0.4

Самописный ResNet

Модель + параметры

Код реализован на основе изначальной статьи про ResNet. Реализуем BasicBlock - основной блок ResNet. StartingBlock - Начальный блок модели ResNet. FinalBlock - блок с полносвязным линейным блоком.

Параметры оставим такие же. Добавим DropOut с вероятностью 0.15. Не будем ресайзить картинки до 224x224

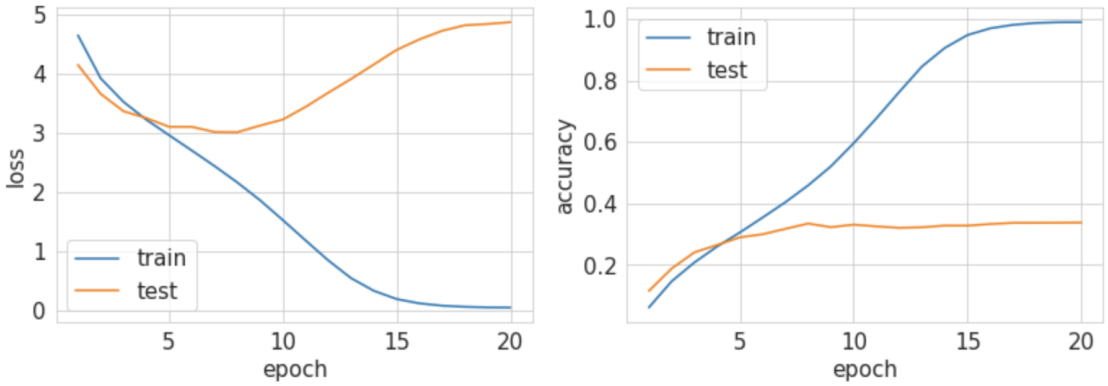
In [ ]:

```
class BasicBlock(nn.Module):

    def __init__(self, in_channels, out_channels, is_dims_changed=False):
        super(BasicBlock, self).__init__()
        self.is_dims_changed = True
        stride = 2
        self.dropout_percantage = 0.15
        if in_channels == out_channels:
            stride = 1
            self.is_dims_changed = False

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, st
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, s
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.drop = nn.Dropout(p=self.dropout_percantage)

        self.dims_change = nn.Conv2d(in_channels=in_channels, out_channels=
        self.relu2 = nn.ReLU()

    def forward(self, x):
        identity = x
        x = self.drop(self.bn2(self.conv2(self.relu1(self.bn1(self.conv1(x)
        if self.is_dims_changed:
            identity = self.dims_change(identity)
        x += self.relu2(x + identity)
        return x

class StartingBlock(nn.Module):
    def __init__(self, out_channels):
        super(StartingBlock, self).__init__()
        self.conv = nn.Conv2d(3, out_channels, kernel_size=7, stride=2, pad
        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        x = self.maxpool(x)
        return x

class FinalBlock(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(FinalBlock, self).__init__()
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(in_channels, num_classes)

    def forward(self, x):
        x = self.avgpool(x)
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x


class ResNet18(nn.Module):
```

```
    def __init__(self, num_classes):
        super(ResNet18, self).__init__()
        self.start = StartingBlock(64)

        self.layer1 = self.make_basic_layer(64, 64)
        self.layer2 = self.make_basic_layer(64, 128)
        self.layer3 = self.make_basic_layer(128, 256)
        self.layer4 = self.make_basic_layer(256, 512)

        self.final = FinalBlock(512, num_classes)

    def make_basic_layer(self, in_channels, out_channels):
        return nn.Sequential(
            BasicBlock(in_channels, out_channels),
            BasicBlock(out_channels, out_channels)
        )


    def forward(self, x):

        x = self.start(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.final(x)
        return x
```

In [ ]:
```
test_transform = T.Compose([
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_transform = test_transform = T.Compose([
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_dataset = MyDataset(root=root, final_test=False, final_train=False, t
val_dataset = MyDataset(root=root, final_test=False, final_train=False, tra
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, pin_m
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, pin_memo

model = ResNet18(len(train_dataset.classes))
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

num_epochs = 20
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
criterion = torch.nn.CrossEntropyLoss()
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, num_epoch

train_losses, test_losses, train_accuracies, test_accuracies = train(
    model, optimizer, scheduler, criterion, train_loader, val_loader, num_e
)
```

Результаты



Стало похуже немного, но все еще хорошо. Возможно надо все-таки сделать ресайз до 224x224

Меньше каналов

Модель + параметры

Уменьшим количество каналов на каждоем слое в 2 раза. Возможно у нас слишком много параметров и уменьшение количества каналов поможет

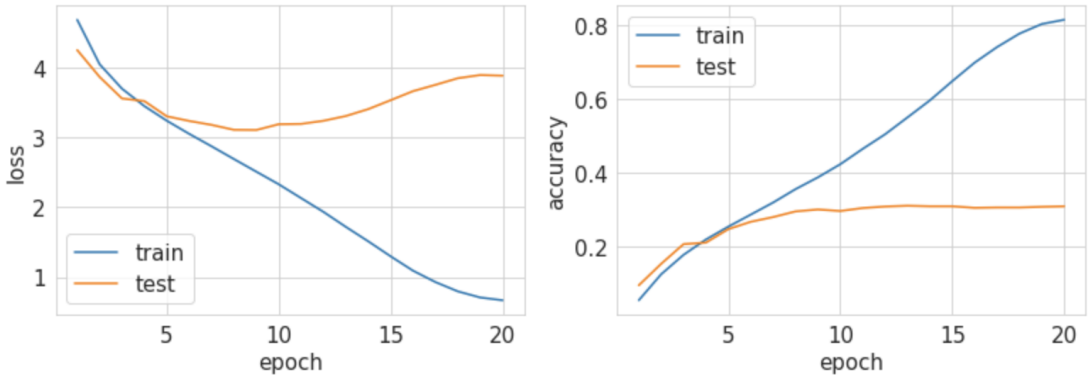
Параметры оставим такие же

In [ ]:

```
class BasicBlock(nn.Module):

    def __init__(self, in_channels, out_channels, is_dims_changed=False):
        super(BasicBlock, self).__init__()
        self.is_dims_changed = True
        stride = 2
        self.dropout_percantage = 0.15
        if in_channels == out_channels:
            stride = 1
            self.is_dims_changed = False

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, st
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, s
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.drop = nn.Dropout(p=self.dropout_percantage)

        self.dims_change = nn.Conv2d(in_channels=in_channels, out_channels=
        self.relu2 = nn.ReLU()

    def forward(self, x):
        identity = x
        x = self.drop(self.bn2(self.conv2(self.relu1(self.bn1(self.conv1(x)
        if self.is_dims_changed:
            identity = self.dims_change(identity)
        x += self.relu2(x + identity)
        return x

class StartingBlock(nn.Module):
    def __init__(self, out_channels):
        super(StartingBlock, self).__init__()
        self.conv = nn.Conv2d(3, out_channels, kernel_size=7, stride=2, pad
        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        x = self.maxpool(x)
        return x

class FinalBlock(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(FinalBlock, self).__init__()
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(in_channels, num_classes)

    def forward(self, x):
        x = self.avgpool(x)
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x


class ResNet18(nn.Module):
```

```
    def __init__(self, num_classes):
        super(ResNet18, self).__init__()
        self.start = StartingBlock(32)

        self.layer1 = self.make_basic_layer(32, 32)
        self.layer2 = self.make_basic_layer(32, 64)
        self.layer3 = self.make_basic_layer(64, 128)
        self.layer4 = self.make_basic_layer(128, 256)

        self.final = FinalBlock(256, num_classes)

    def make_basic_layer(self, in_channels, out_channels):
        return nn.Sequential(
            BasicBlock(in_channels, out_channels),
            BasicBlock(out_channels, out_channels)
        )


    def forward(self, x):

        x = self.start(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.final(x)
        return x
```

Результаты



Не помогло, стало хуже

Добавим аугментации

Модель + параметры

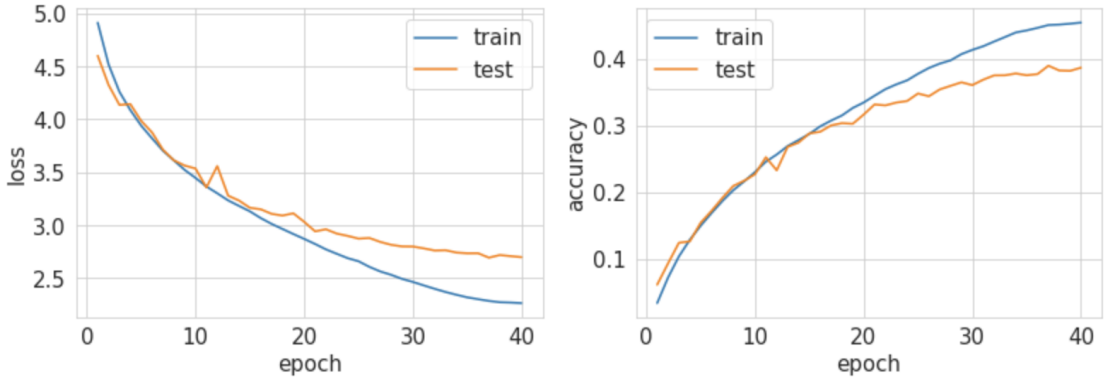
Известно, что аугментации улучшают качество на тесте и помогает бороться с переобучением, которое мы наблюдаем. Добавим RandomCrop и HorizontalFlip, которые являются простыми.

Количество каналов оставим как до уменьшения

Количество эпох увеличим в 2 раза, так как аугментации это шум и скорее обучаться будем медленнее

In [ ]:
```
test_transform = T.Compose([
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_transform = test_transform = T.Compose([
        T.RandomResizedCrop(64),
        T.RandomHorizontalFlip(),
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])
```

Результаты



На тестовой выборке стало немного лучше. Но зато переобучение сильно уменьшилось

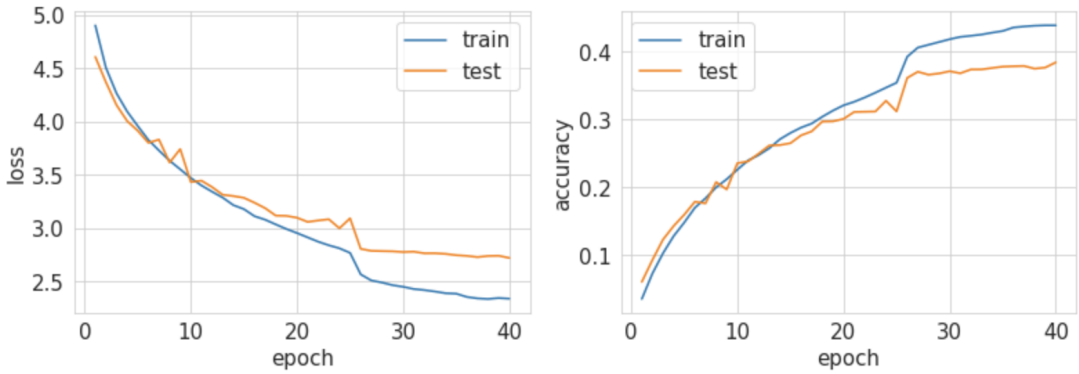
Поменяем шедулер

Модель + параметры

Попробуем поменять расписание на MultiStep. Он попроще, что может помочь. Milestones выберем 25 и 35. Аугментации оставим.

In [ ]:
```
scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[25,
```

Результаты



Ничего не поменялось, кроме вида графиков

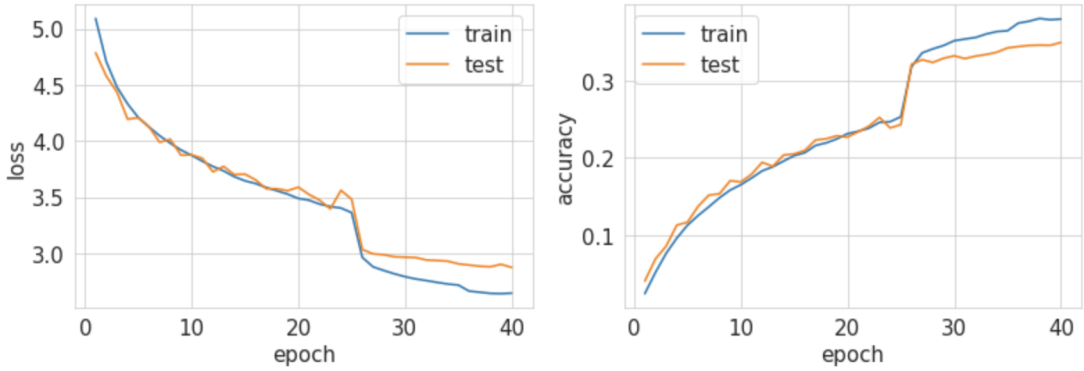
Увеличимм начальный lr

Модель+параметры

Так как мы довольно сильно уменьшаем начальный lr возможно стоит сделать его больше. Например 0.1. Шедулер оставим MultiStep

In [ ]:
```
optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9)
```

Результаты



Стало заметно хуже. Плохая была идея

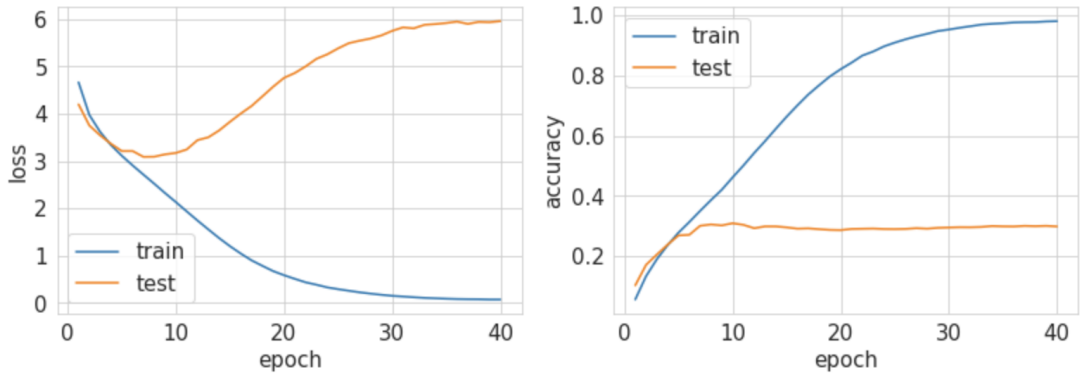
Меняем оптимизатор на Adam

Модель + параметры

Попробуем поменять оптимизатор на Adam. Во всех домашних заданиях Adam лично у меня показывал хорошие результаты и лучше SGD. Уберем аугментации и вернем шедулер CosineAnnealingLR

In [ ]:
```
optimizer = torch.optim.Adam(model.parameters())
```

Результаты



Стало хуже по сравнению с моделью 4 (Самописный ResNet). Почитав статьи я выяснил, что для классификации изображений лучше все-таки SGD

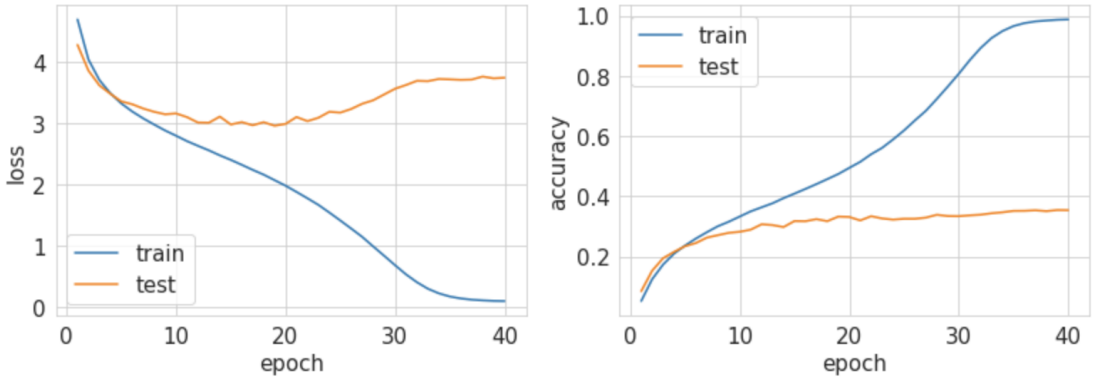
Weight Decay

Модель + параметры

Попробуем бороться с переобучением путем добавления weight decay равного 1e-3. Остальные параметры из Самописного ResNet.

In [ ]:
```
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9, weig
```

Результаты



Результат примерно такой же, но зато переобучение меньше

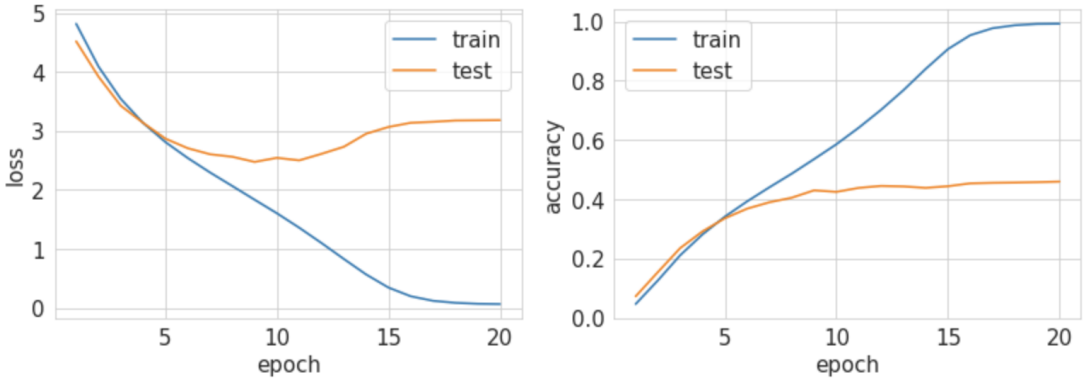
Resize 224x224

Модель + параметры

Вообще архитектура ResNet из исходной статьи работает на картинках 224x224. Сделаем resize 224. Остальные параметры возьмем как Самописного ResNet

In [ ]:
```
test_transform = T.Compose([
        T.Resize(256),
        T.CenterCrop(224),
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_transform = T.Compose([
        T.Resize(256),
        T.CenterCrop(224),
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])
```

Результаты



Резульат даже лучше, чем у ResNet из PyTorch

Итоги ResNet базовый

Если взять resize 224x224, weight_decay и аугментации, то гарантировано можно получить 0.45 и возможно даже больше. Но этого мало. Будем пробовать дальше

ResNet для Cifar-10

Модель + параметры

Возьмем архитектуру из исходный статьи про ResNet, но в этот раз для Cifar-10. Должно помочь, так как Cifar-10 сравним по размеру с нашим датасетом и гораздо меньше ImageNet для которого написан изначальный ResNet. Само собой поменяем число классов в модели. Остальные параметры возьмем как в статье. Лишь добавим DropOut(0.15), так как в статье сказано, что хоть DropOut и не использован, но сказано, что DropOut может помочь. Так же сделаем лишь 40 эпох и будем уменьшать lr после 30 и 35 эпох. Будем использоваь 20 слоев

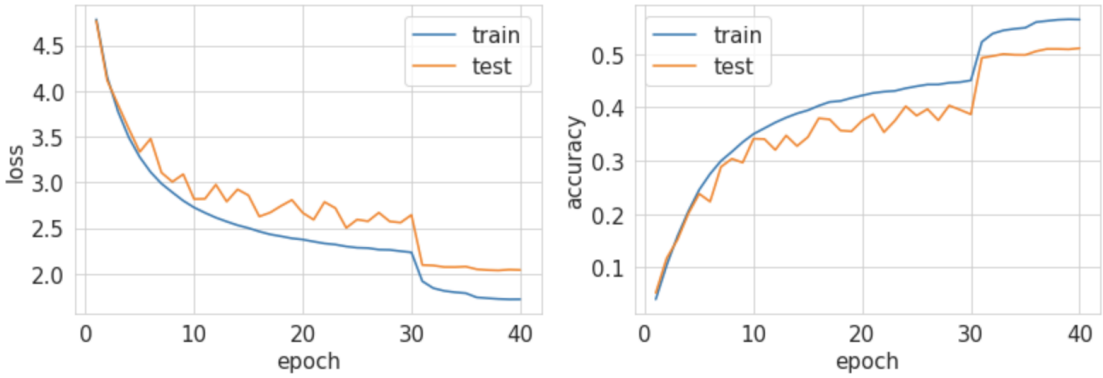
In [ ]:

```
class BasicBlock(nn.Module):

    def __init__(self, in_channels, out_channels):
        super(BasicBlock, self).__init__()
        self.is_dims_changed = True
        stride = 2
        self.dropout_percantage = 0.15
        if in_channels == out_channels:
            stride = 1
            self.is_dims_changed = False

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, st
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, s
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.drop = nn.Dropout(p=self.dropout_percantage)

        self.dims_change = nn.Conv2d(in_channels=in_channels, out_channels=
        self.relu2 = nn.ReLU()

    def forward(self, x):
        identity = x
        x = self.drop(self.bn2(self.conv2(self.relu1(self.bn1(self.conv1(x)
        if self.is_dims_changed:
            identity = self.dims_change(identity)
        x += self.relu2(x + identity)
        return x

class StartingBlock(nn.Module):
    def __init__(self, out_channels):
        super(StartingBlock, self).__init__()
        self.conv = nn.Conv2d(3, out_channels, kernel_size=3, stride=1, pad
        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        return x

class FinalBlock(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(FinalBlock, self).__init__()
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(in_channels, num_classes)

    def forward(self, x):
        x = self.avgpool(x)
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x


class ResNet18(nn.Module):

    def __init__(self, num_classes):
        super(ResNet18, self).__init__()
```

```
        self.in_channels = 16

        self.start = StartingBlock(16)

        self.layer1 = self.make_layer(16, 3)
        self.layer2 = self.make_layer(32, 3)
        self.layer3 = self.make_layer(64, 3)

        self.final = FinalBlock(64, num_classes)

    def make_layer(self, out_channels, num_blocks):
        layers = []
        for _ in range(num_blocks):
            layers.append(BasicBlock(self.in_channels, out_channels))
            self.in_channels = out_channels

        return nn.Sequential(*layers)


    def forward(self, x):
        x = self.start(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.final(x)
        return x
```

In [ ]:
```
test_transform = T.Compose([
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_transform = test_transform = T.Compose([
        T.RandomCrop(64, 4),
        T.RandomHorizontalFlip(),
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_dataset = MyDataset(root=root, final_test=False, final_train=False, t
val_dataset = MyDataset(root=root, final_test=False, final_train=False, tra
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True, pin_
val_loader = DataLoader(val_dataset, batch_size=128, shuffle=False, pin_mem

model = ResNet18(len(train_dataset.classes))
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

num_epochs = 40
optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weigh
criterion = torch.nn.CrossEntropyLoss()
scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[30,

train_losses, test_losses, train_accuracies, test_accuracies = train(
    model, optimizer, scheduler, criterion, train_loader, val_loader, num_e
)
```

Результаты



Получилось пробить 50%. Успех

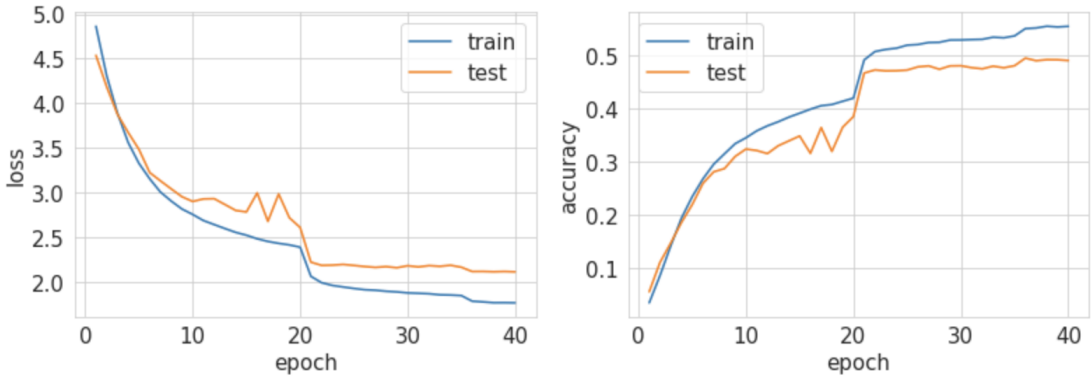
Уменьшаем MileStone

Модель + параметры

Заметим, что после 20 эпохи особо изменения нет, а лишь колебания. Возможно в этот момент lr уже большой и его надо в этот момент уменьшать

In [ ]:
```
scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[20,
```

Результаты



Стало хуже. Идея не оправдалась

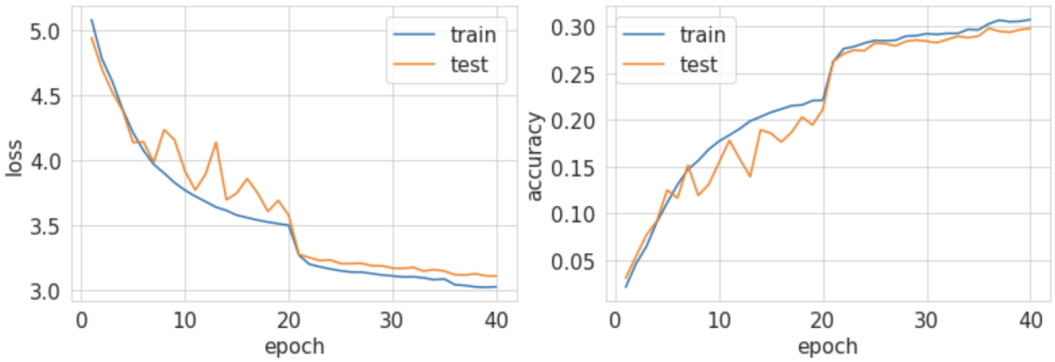
Resize 32x32

Модель + параметры

В оригинальной статье использовались картинки 32x32. Давайте сделаем Resize 32

In [ ]:
```
test_transform = T.Compose([
        T.Resize(36),
        T.CenterCrop(32),
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])

train_transform = test_transform = T.Compose([
        T.RandomCrop(32, 4),
        T.RandomHorizontalFlip(),
        T.ToTensor(),
        T.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )])
```

Результаты



Очень плохо

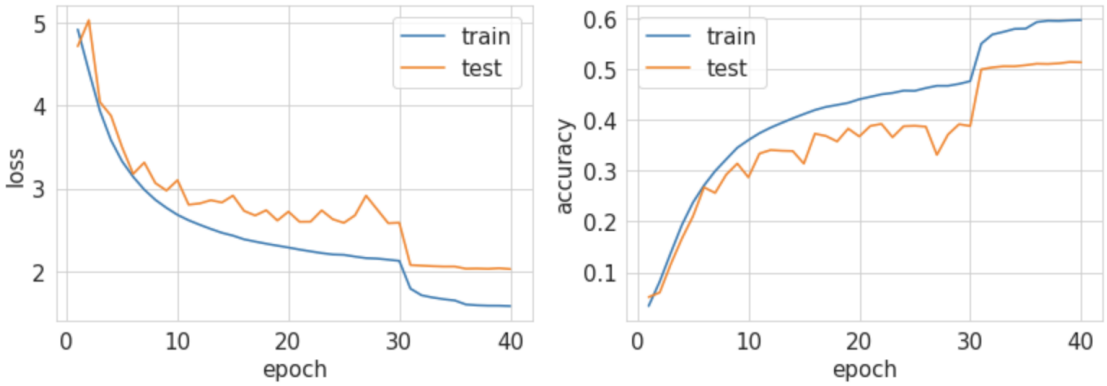
Уберем DropOut

Модель + параметры

В исходной статье нет DropOut. Давайте попробуем убрать

In [ ]:
```
self.dropout_percantage = 0.0
```

Результаты



Результат не поменяется. Но переобучение стало немного больше

32 слоя без dropout

Модель+параметры

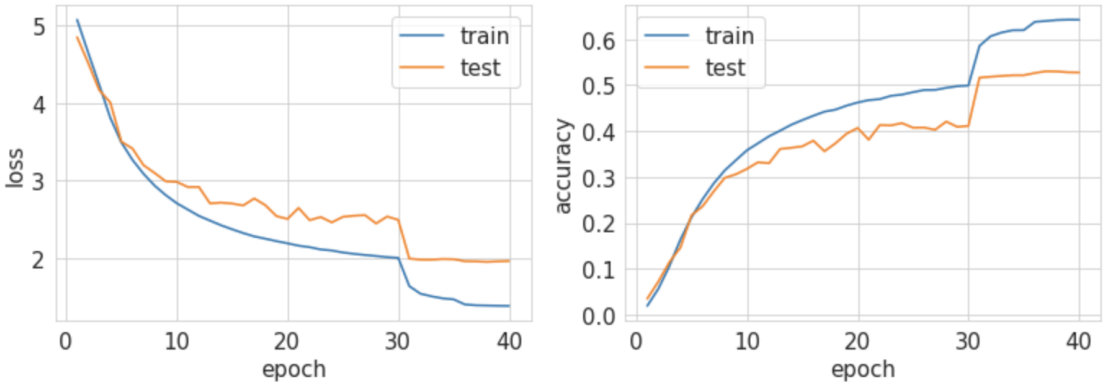
Давайте попробуем углубить модель до 32 слоев. И также пусть будет без dropout

In [ ]:

```
class BasicBlock(nn.Module):

    def __init__(self, in_channels, out_channels):
        super(BasicBlock, self).__init__()
        self.is_dims_changed = True
        stride = 2
        self.dropout_percantage = 0.0
        if in_channels == out_channels:
            stride = 1
            self.is_dims_changed = False

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, st
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, s
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.drop = nn.Dropout(p=self.dropout_percantage)

        self.dims_change = nn.Conv2d(in_channels=in_channels, out_channels=
        self.relu2 = nn.ReLU()

    def forward(self, x):
        identity = x
        x = self.drop(self.bn2(self.conv2(self.relu1(self.bn1(self.conv1(x)
        if self.is_dims_changed:
            identity = self.dims_change(identity)
        x += self.relu2(x + identity)
        return x

class StartingBlock(nn.Module):
    def __init__(self, out_channels):
        super(StartingBlock, self).__init__()
        self.conv = nn.Conv2d(3, out_channels, kernel_size=3, stride=1, pad
        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        return x

class FinalBlock(nn.Module):
    def __init__(self, in_channels, num_classes):
        super(FinalBlock, self).__init__()
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(in_channels, num_classes)

    def forward(self, x):
        x = self.avgpool(x)
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x


class ResNet18(nn.Module):

    def __init__(self, num_classes):
        super(ResNet18, self).__init__()
```

```
        self.in_channels = 16

        self.start = StartingBlock(16)

        self.layer1 = self.make_layer(16, 5)
        self.layer2 = self.make_layer(32, 5)
        self.layer3 = self.make_layer(64, 5)

        self.final = FinalBlock(64, num_classes)

    def make_layer(self, out_channels, num_blocks):
        layers = []
        for _ in range(num_blocks):
            layers.append(BasicBlock(self.in_channels, out_channels))
            self.in_channels = out_channels

        return nn.Sequential(*layers)


    def forward(self, x):
        x = self.start(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.final(x)
        return x
```

Результаты



Стало лучше

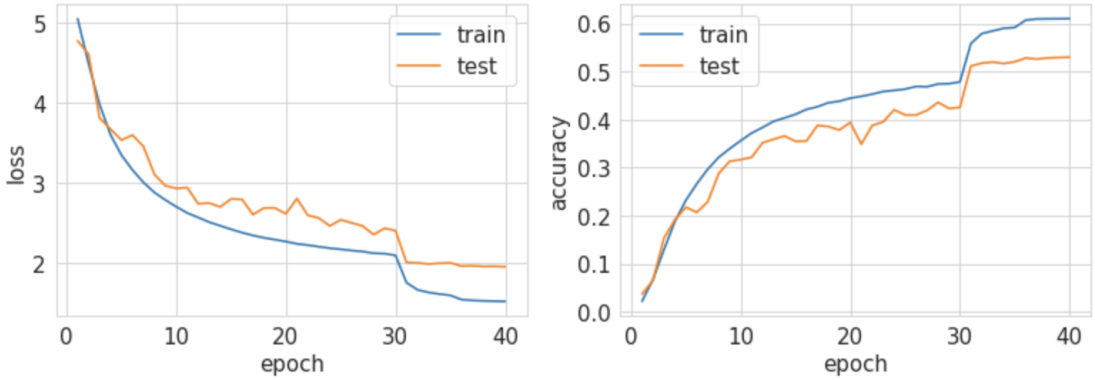
32 слоя + dropout

Модель + параметры

Попробуем не убирать dropout

In [ ]:
```
self.dropout_percantage = 0.15
```

Результаты



Результат такой же как и без dropout, но переобучение меньше

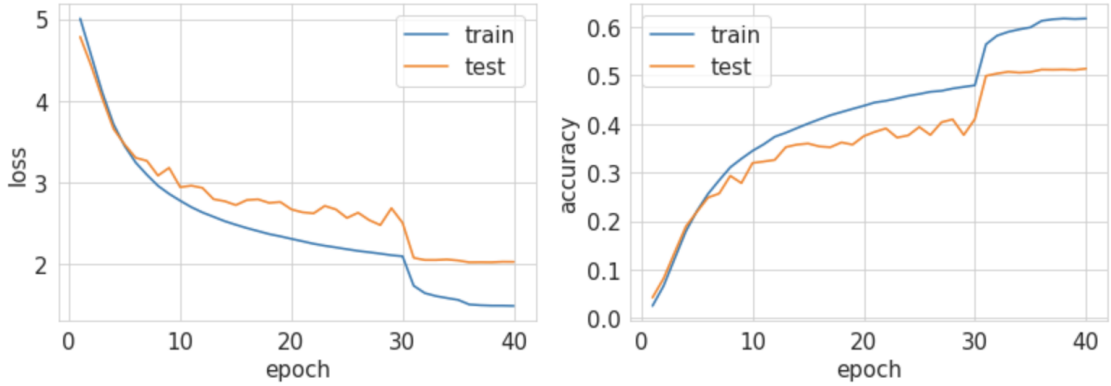
Попробуем LeakyReLu

Модель + параметры

Еще не пробовал менять функцию активации. Вдруг поможет. Оставим глубину в 32 слоя и также пусть будет без dropout

In [ ]:
```
self.relu = nn.LeakyReLU()
```

Результаты



Стало хуже

Итоги

Самый лучший результат для модели ResNet32 для Cifar-10 из исходной статьи про ResNet без изменения размера изображений до 32x32 и с DropOut с вероятностью 0.15

In [ ]:
```

```