1 Предисловие

После того, как на дл мы закончили с cv, я очень ждал эту домашку. В голове я не раз прокручивал следующий алгоритм: берём SOTA модель, которую сможем обучить, накидываем сложных аугментаций, потом немного поэксперементируем с оптимайзерами и шедулерами и топ-1 обеспечен. Ах да, ещё я думал, что можно просто запустить обучение и часов на 10 забыть про это всё.

По итогу я ошибался во всём, о чём написано выше.

Дисклеймер: в отчёте не будет слишком много графиков по одной простой причине: Kaggle решил, что сохранять мои ноутбуки стоит тогда, когда ему вздумается, поэтому какая-то часть экспериментов была бесследно утрачена. Также везде, где я пишу SGD, подразумевается наличие momentum=0.9.

2 Первые шаги

Первое, что я решил сделать, это посчитать среденее и стандартное отклонение по датасету. Зачем? Хочу. Ну а вообще это странно брать значения по ImageNet.

```
mean, std = torch.zeros(3), torch.zeros(3)
for inputs, labels in tqdm(loader):
    for i in range(3):
        mean[i] += inputs[:,i,:,:].mean()
        std[i] += inputs[:,i,:,:].std()
mean.div_(len(dataset))
std.div_(len(dataset))
```

2.1 Реализую исходую идею

Как я писал выше, хотелось взять SOTA модель. Я решил не мелочиться и сразу взять EfficientNet_B7. В качестве аугментаций были взяты RandomHorizontalFlip, CutOut, CutMix, MixUp – в общем всё самое крутое, что я знал. В качестве оптимайзера был взят Adam с lr=0.001. Далее запуск, долгое ожидание и по истечению 10 эпох качество не побило даже 5%.

Я очень удивился и стал думать, а что могло пойти не так. Решил я посмотреть, а что там за картинки такие: они оказались очень шакального качества. В условии соревнования написано, что картинки ресайзить нельзя. Я опять задумался: логично, что современные сетки хотят на вход получать большую картинку, а значит, либо надо брать не современные, либо их тюнить. Первое звучит достаточно странно, поэтому я решил заняться вторым.

3 Пора бы начать думать

Было принято решение взять ResNet50 и чуток его изменить: первоначальная свёртка 7x7 мне показалась слишком огромной и я решил уменьшить её, убрать stride, а также убрать MaxPooling, потому что таким образом мы сразу теряем четверть картинки(ну условно). В качестве оптимайзера я всё так же оставил Adam с косинусным расписанием на 20 эпох. Результат улучшился: вышли за 15%.

Пока сетка училась, я решил провести небольшой research. Очевидно, что мы не первые, кто столкнулся с задачей классификации маленьких картинок, поэтому я решил поискать известные датасеты с небольшими картинками. И тут я наткнулся на TinyImagenet. Меня сразу зацепило то, что в нём также 200 классов и 100k картинок для train(такие совпадения неслучайны). Я решил почитать статьи, где всякие умные люди пытались изобрести что-то новое и улучшить старое(дальнейшие улучшения основаны на статьях).

Посмотрев на типичные параметры оптимайзеров из статей, а также вспомнив фразу Ильдуса о том, что для свёрточных сеток обычно берут SGD, решил выкинуть Adam и использовать SGD с lr=0.1 и шедулером ReduceLROnPlateau с дефолтными параметрами. В результате запуск выглядил следующим образом:

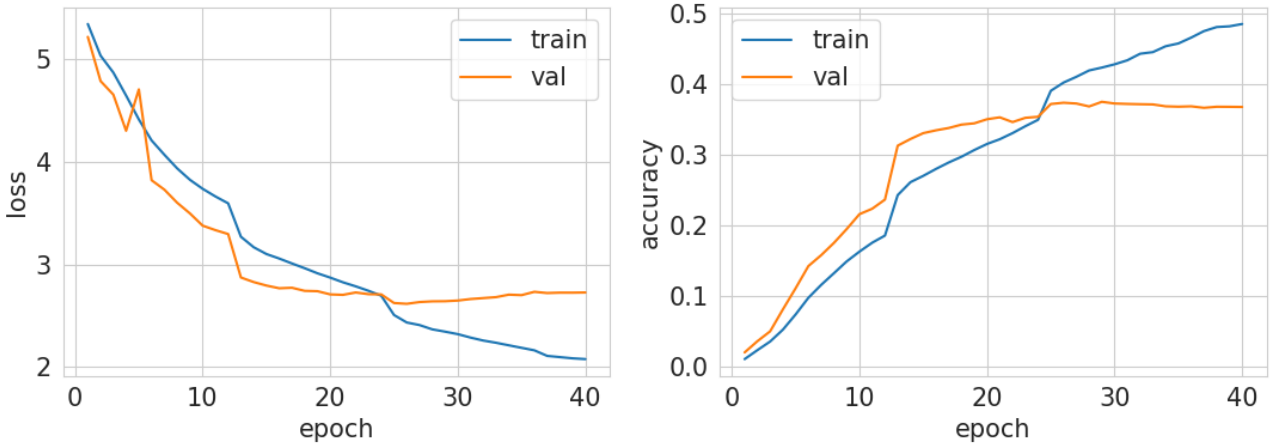
```
train_transform = v2.Compose([
        v2.RandomChoice([v2.RandomHorizontalFlip(0.25), v2.RandomVerticalFlip(0.25)], p=[0.5,
    0.5]),
        torchvision.transforms.ToTensor(),
        v2.RandomErasing(ratio = (1.0, 1.0)),
        v2.Normalize(mean=mean, std=std),
    ])
batch_size=100
model = resnet50()
model.conv1 = Conv2d(3, 64, kernel_size = 3, padding = 1)
model.maxpool = Sequential()
```

```
optimizer = SGD(model.parameters(), lr = 0.1, momentum = 0.9, weight_decay = 0.0001)
scheduler = ReduceLROnPlateau(optimizer)
num_epochs = 100
```

После 20 эпох качество перестало улучаться и остановилось на $\approx 30\%$.

4 А где Dropout?

Переобучение на 20 эпохе меня не устраивало, поэтому был добавлен Dropout(0.5), а дальнейшем и Dropout(0.8) перед линейным слоем.



Разница между сетками с Dropout 0.5 и 0.8 заключается в том, что первая учится и переобучается быстрее, вторая же и учится, и переобучается позже. По цифрам при большем Dropout результат получался на пару процентов выше.

Далее пришла мысль, что можно увеличить количество линейных слоёв для получения логитов:

```
model.fc = torch.nn.Sequential(
    torch.nn.Dropout(inplace = True, p = 0.8),
    torch.nn.Linear(in_features = 2048, out_features = 2048, bias = True),
    torch.nn.BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True),
    torch.nn.ReLU(inplace=True),
    torch.nn.Linear(in_features = 2048, out_features = NUM_CLASSES, bias = True),
)
```

Качество данное изменение не повысило, поэтому в дальнейшем я его не использовал.

Ещё хочется отметить, что ни в одной из статей по обучению свёрточной сетки на TinyImagenet, не было рекомендаций использовать Dropout. Вместо этого были иные способы регуляризации, такие как label smoothing или stochastic depth. Первое осталось в итоговой реализации, в то время, как при использовании второго, лосс слишком сильно прыгал, из-за чего lr рано понижался и модель переставала учится.

5 Про аугментации

При каждом запуске у меня стабильно было две аугментации: рандомный выбор из горизонтального и вертикального флипов и CutOut, который в torch имеет более логичное название RandomErasing. Изначально мне показалось, что вериткальный флип – это достаточно сложная аугментация, но на деле что с ним, что без качество не менялось. Эксперименты с RE тоже ничего не дали: я пробывал закрашивать вырезанный кусок разными типами и брать разные размеры выреза; неизменным лишь была лишь форма выреза – квадрат.

Время от времени я добавлял ColorJitter для изменения яркости, контраста и сатурации, но в итоговую модель это не вошло.

Эксперименты с CutMix, MixUp и Augmix тоже не привели ни к чему хорошему, что очень меня огорчило. Хочется верить, что это сложные агументации для данной задачи, но в статьях нередко видел, их использовали на CIFAR-10/100.

Сильно улучшить положение(с 49% до 52%) помог RandAugment с дефолтным количеством трансформаций(увеличивая их, увеличивалось только время прогона одной эпохи). Делает он следующее: к картинке рандомно применяется 2 аугментации из достаточно большого списка.

Ещё были использованы RandomGrayscale, GaussianBlur. Всё, что про них могу сказать, – без них хуже.

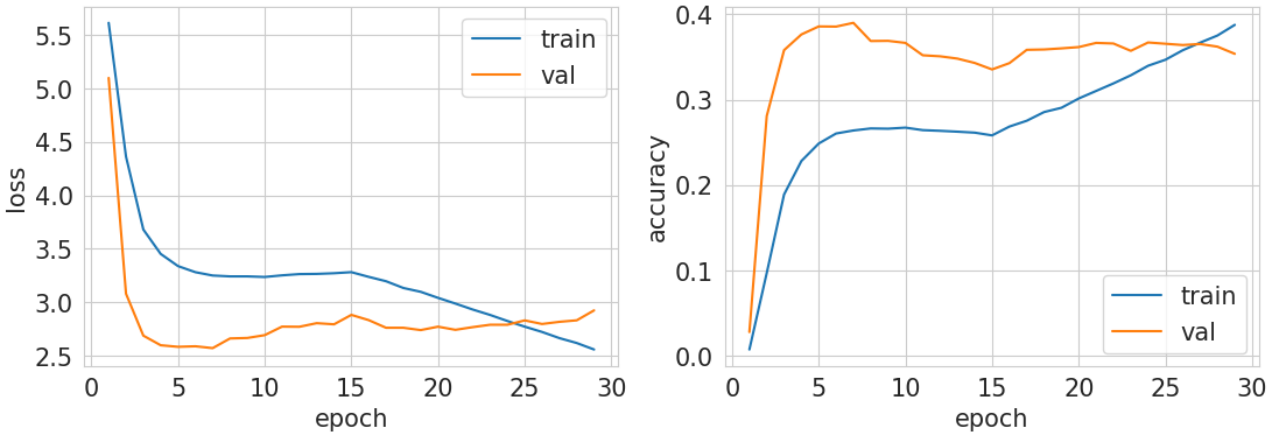
6 Попытки реализовать статьи

Первой попыткой был запуск WideResNet50_2: графиков нет, но там было всё плохо.

Второй попыткой была идея запуска resnext50_32x4d на 400 эпох с SGD с lr=0.2, косинусным расписанием. Единого графика нет, есть по частям, потому что КАК МОЖНО ОБУЧИТЬ 400 ЭПОХ В ДОЛБАННОМ KAGGLE. По итогу выйти за 45% не удалось.

Последней попыткой был запуск ResNet34, но с удвоенным количеством свёрток на каждом слое. Ели вышел за 40%.

Ещё был запуск Swin с AdamW, lr=1e-3, косинусное расписание с warm up каждые 7 эпох. После 2 эпохе выбивалось 30%, под конец за 40% выйти не удалось. Остался лишь такой график



7 Итоговая реализация

В качестве модели взят всё тот же ResNet50, в котором первая свёртка это Conv2d(3, 64, kernel_size=4, bias=False), удалён MaxPooling и добавлен Dropout(0.8), а все веса инициализируются из $\mathcal{N}(0, \frac{2}{in\_features})$[1] В конце будет график с точно таким же запуском, но с дефолтной инициализацей, $\mathcal{U}[-\frac{1}{\sqrt{in\,features}}, \frac{1}{\sqrt{in\,features}}]$.

Училась модель в два(хотя по факту в три) этапа с разницей в аугментациях, остальное было неизменно: данные делились на train и val 99:1, batch_size = 200, SGD, lr=0.1, weight_decay=0.0001, в качестве расписания ReduceLROnPlateau(optimizer, cooldown=1, patience=7). Также был добавлен label smoothing=0.1 для дополнительной регуляризации.

1. Ставим с рандомным флипом, RandomGrayscale, GaussianBlur и RE на 70 эпох(этого должно хватить) и ждём скачка после понижения lr, а после ещё одной эпохи выключаем.

2. Оставляем флип и RE, а остальные заменяем на RandAugment и ставим максимум на 30 эпох со сброшенным оптимайзером и шедулером. Зачем? За первые 70 эпох мы 100% попадём в какой-то локальный минимум, поэтому было бы прикольно начать второй этап с поиска нового направления(хотя возможно я ошибаюсь и тут что-то другое улучшает положение). Тут также ждём скачка при понижении lr, и после ещё одной эпохи выключаем.

3. Этот этап является необязательным, но качество он постоянно улучшал: прогон одной эпохи на тех же аугментациях, но со сброшенным SGD с lr=0.01.

8 А что дальше?

Выбив лучший скор, я решил посмотреть, будут ли отличия, если что-нибудь поменять.

Сперва, я заменил ResNet50 на ResNet34, потому что мне казалось, что первая модель достаточно глубокая для данной задачи. Результат стал значительно хуже.

[1] Я так и не понял, как называется этот способ: кто-то его называет Variance Initialization, кто-то kaiming_normal, хотя реализация последнего в torch имеет совершенно другую дисперсию.

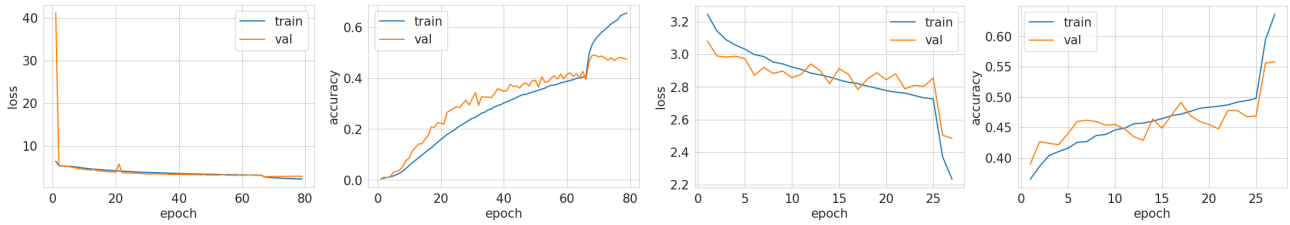


Рис. 1: Левый график – первый этап обучения, второй график – второй этап

Далее, я попробовал добавить нестеровский момент с надеждой, что так SGD лучше сойдётся. Эффекта это не дало.

Ну и напоследок я решил поиграться с weight decay: увеличил его на два порядка и уменьшил на порядок. Результата это так же не дало.

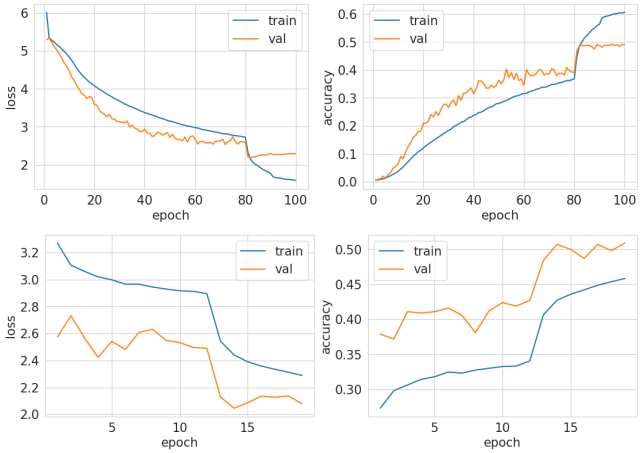


Рис. 2: Лучший запуск, но с дефолтной инициализацией весов