Отчет по БДЗ1
Глубинное обучение 1

Потарусов Артём

23 января 2025 г.

1 Введение

Так получиось, что экспериментов я провел много (можно посмотреть по чилсу посылок в каггле), но графики остались не от всего (каггл, если не сделать квиксейв их просто терял, а датасфера при запуске в ночь просто не выводила ничего). Часть я заскриншотил, но теперь остается всомнить а что же я все таки пробовал. В целом, архитектура была собрана быстро, а потом я перебирал все возможные оптимизаторы и аугментации, но об этом позже.

2 Чекпоинт

Перво наперво я попробовал архитектуру из МДЗ2, адаптировав размеры и добавив MaxPool. Хорошего результата это не дало, потому я добавил еще сверток и обучал это все на обычном Adam с lr=3e-4. Выбить порог так мне удалось.

3 Описание экспериментов

На данном этапе я подумал, что надо бы попробовать классику и решил взять 1 блок резнета, поставить дропаут 0,5 аугментаций намешать (ColorJitter, Horisontalflip, Rotation) и из жадности поставил на каггле в ночь на CPU. За 12 часов эта штука словила TL (я был очень удивлен), но доучив ее на завтра получил качество в 17 на тесте, что меня совсем не устроило (были еще попытки MaxPool впихнуть между свертками, но смотря историю версий в коллабе вижу, что это даже на трэйне набирало всего 10 (scheduler был StepLR, от которого я в скором времени откажусь)).

```
100%|██████████| 1563/1563 [03:48<00:00,  6.85it/s]
Epoch 1/10, Train Loss: 4.9358, Train Acc: 0.03351
100%|██████████| 1563/1563 [03:54<00:00,  6.66it/s]
Epoch 2/10, Train Loss: 4.7089, Train Acc: 0.05881
100%|██████████| 1563/1563 [03:56<00:00,  6.60it/s]
Epoch 3/10, Train Loss: 4.6057, Train Acc: 0.0703
100%|██████████| 1563/1563 [03:52<00:00,  6.72it/s]
Epoch 4/10, Train Loss: 4.5338, Train Acc: 0.07944
100%|██████████| 1563/1563 [03:53<00:00,  6.70it/s]
Epoch 5/10, Train Loss: 4.4730, Train Acc: 0.08687
100%|██████████| 1563/1563 [03:59<00:00,  6.52it/s]
Epoch 6/10, Train Loss: 4.3828, Train Acc: 0.1
100%|██████████| 1563/1563 [03:54<00:00,  6.66it/s]
Epoch 7/10, Train Loss: 4.3667, Train Acc: 0.1031
100%|██████████| 1563/1563 [03:53<00:00,  6.69it/s]
Epoch 8/10, Train Loss: 4.3626, Train Acc: 0.1036
100%|██████████| 1563/1563 [03:56<00:00,  6.61it/s]
Epoch 9/10, Train Loss: 4.3473, Train Acc: 0.1047
100%|██████████| 1563/1563 [04:01<00:00,  6.47it/s]
Epoch 10/10, Train Loss: 4.3432, Train Acc: 0.1062
```

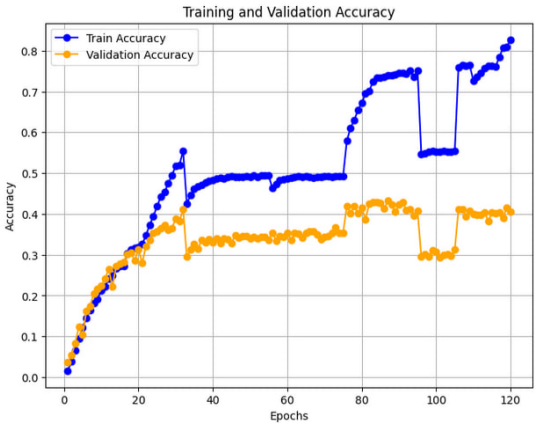


Рис. 1: Остатки эксперимента на 12 часов (перезапуск был в коллабе) и график последней обученной модели

В то же время я решил, что надо бы нормально отцентрировать датасет и посчитал среднее и дисперсию по трэйну, заменил StepLr на CosineAnealingLr (что позволило на тесте набрать 17), но также пробовал с тем же StepLr уменьшить глубину (что привело к полному расколбасу и всего 5% качеству (еще я на валидацию отдавал пятую часть тогда, со временем это стало всего двадцатая, а потом и пятидесятая, но бывали запуски и вовсе без нее))

```
100%|██████████| 1250/1250 [02:34<00:00,  8.11it/s]
Epoch 17
 train loss: 3.7776, train acc: 0.1792
 val loss: 3.8280, val acc: 0.1708

100%|██████████| 1250/1250 [02:33<00:00,  8.12it/s]
Epoch 18
 train loss: 3.7800, train acc: 0.1778
 val loss: 3.8375, val acc: 0.1704

100%|██████████| 1250/1250 [02:32<00:00,  8.19it/s]
Epoch 19
 train loss: 3.7870, train acc: 0.1772
 val loss: 3.8241, val acc: 0.1699

100%|██████████| 1250/1250 [02:35<00:00,  8.02it/s]
Epoch 20
 train loss: 3.7818, train acc: 0.1783
 val loss: 3.8324, val acc: 0.1704
```

Рис. 2: Качество до и после уменьшения глубины

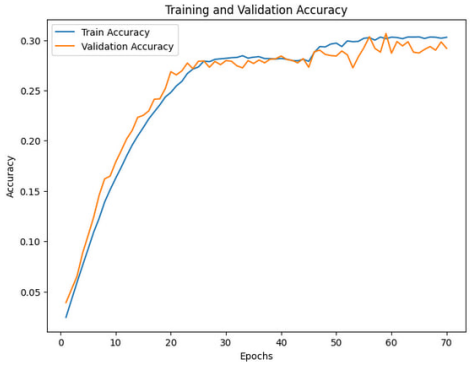
Собственно так я его и посчитал:

```python
def calculate_mean_std(dataset):
    loader = DataLoader(dataset, batch_size=64, shuffle=False,
       num_workers=2)
    mean = 0.0
    std = 0.0
    total_images = 0
    for images, _ in tqdm(loader):
        batch_samples = images.size(0)
```

```
        images = images.view(batch_samples, images.size(1), -1)
        mean += images.mean(2).sum(0)
        std += images.std(2).sum(0)
        total_images += batch_samples
    mean /= total_images
    std /= total_images
    return mean, std
```

4 Нормальные модели

К этому времени я прожег компьют коллаба и понял, что хватит изобретать велосипед и надо попробовать что-то готовое. А потому я решил взять обычный библиотечный resnet50, убрал веса, поправил количество выходных классов и решил обучать в 2 этапа: сначала CosineAnealing на AdamW(lr=1e-3, weight_decay=1e-4) на 45 итераций, а затем ReduceLrOnPlateau на 25 с SGD(lr=1e-3, momentum=0.9, weight_decay=1e-4), а из аугментаций и язык флип (хорошо показал себя с МДЗ) и нашел килер-фичу в видел RandAugment (выбирает случайно из списка аугментаций и применяет). Философия была такая: вот я найду хороший локальный минимум, а потом попытаюсь найти более широкий с помощью SGD (а шедулер подстрахует в плохом случае). Такое решение давало 30. Я еще попробовал заменить RandAugment на ColorJitter и с дуру увеличил размер батча до 512-1024, после чего качество упало до 25.




```
Epoch 29: Train Accuracy: 0.1403, Validation Accuracy: 0.1825
Epoch 30/32: 100%|██████████| 383/383 [01:50<00:00,  3.47it/s]
Epoch 30: Train Accuracy: 0.1895, Validation Accuracy: 0.2125
Epoch 31/32: 100%|██████████| 383/383 [01:49<00:00,  3.50it/s]
Epoch 31: Train Accuracy: 0.2183, Validation Accuracy: 0.2315
Epoch 32/32: 100%|██████████| 383/383 [01:49<00:00,  3.51it/s]
Epoch 32: Train Accuracy: 0.2389, Validation Accuracy: 0.2475
```

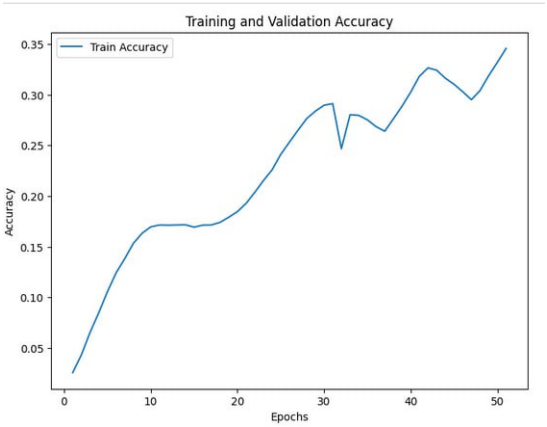
Рис. 3: ResNet50 норм и с не норм батчем

5 Архитектура модели

Вот тут пошла жара. Я решил, что внутри блока лучше сделать 2 свертки 3 на 3, чем две 1 на 1 и между ними 3 на 3, убрать MaxPool в initial_layer и сделать в нем свертку с ядром 3 на 3, вместо 7 на 7 (я решил, что раз картинки маленькие, мы много инфы так теряем, но спустя неделю перебирания всего чего только можно в этой архитектуре мне пришло в голову, что возможно первую свертку надо было оставить 7 на 7, чтобы увеличить рецептивное поле).

В общем взял я CyclicLR (я им полбзовался как обычным шедулером, хотя вроде как его надо обновлять после батча, но в таком режиме он как-то не особо был эффективен). Попробовал на всем трэйне сначала на 30 итераций, потом еще на 25

с RandAugment и флипом, получив качество в 35 на трэйне, но после отправки в каггл, получил 42 на тесте!




```
Epoch 27: Train Accuracy: 0.4623, Validation Accuracy: 0.3848
Epoch 28/30: 100%|██████████| 1485/1485 [08:55<00:00,  2.77it/s]
Epoch 28: Train Accuracy: 0.4626, Validation Accuracy: 0.3812
Epoch 29/30: 100%|██████████| 1485/1485 [08:54<00:00,  2.78it/s]
Epoch 29: Train Accuracy: 0.4655, Validation Accuracy: 0.3830
Epoch 30/30: 100%|██████████| 1485/1485 [08:55<00:00,  2.77it/s]
Epoch 30: Train Accuracy: 0.4666, Validation Accuracy: 0.3880
```

Рис. 4: Успех с RandAugment и проба заменить его на Jitter

Я где-то (мб в документации по resnet) прочитал про инициализацию весов kaiming_normal (там нормированным нормалным распределением проинициализировали веса) и добавил себе в код. В целом, это помогло быстрее выходить на нормальные значения. Пробовал заменить RandAugment на ColorJitter пожирнее, но результат это скорее ухудшило. Также попробовал учить уже в 3 этапа (с CycleLr, CosineAnealingLr, ReduceLrOnPlateau) (возможно это решение дало на тесте 47, но я не помню), а еще попробовать вернуться к StepLr и обучать сразу на SGD, но с объединенными RandAugment и ColorJitter сразу, результаты ниже.

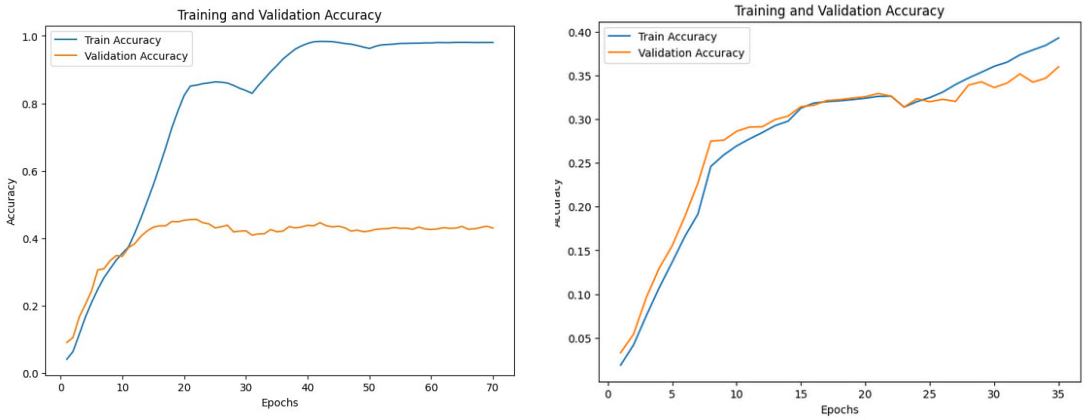


Рис. 5: 3 этапа и объединение RandAugment с ColorJitter

Где-то тогда я выбил в первый раз больше 48 и кажется сделал это взяв RandomAugment и CyclicLr. Дальше я стал делать в датасфере, которая оказалось не имеет истории версий, а потому следов гиперпараметров не осталось (в ночь перед дедлайном я смог выбить ровно 48, но с использованием CutMix (о нем еще скажу дальше). В раз перед этим я выбил 48 используя схему обучения ниже (на этапах 20, 20, 15 итераций соответственно) (я точно не помню какой код давал лучший результат, но вроде один из 2, так что я попробую прикрепить 2 ноутбука, если не найду лучший (там прикол в том, что в ноутбуке ошибка, из-за которой валидация протекла в трэйн; правда у меня пока сейчас 3 модели учатся и 3 посылки остается - если хоть

одна выбьет лучше оставлю ее (там правда не совсем сверху вниз будет работать прогон кода, я некоторые ячейки по несколько раз запускал))):

```
transform_train_stage1 = transforms.Compose([
    transforms.RandomResizedCrop((40, 40), scale=(0.8, 1.0), ratio=(0.75, 1.33)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandAugment(),
    transforms.RandomGrayscale(p=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std)
])

transform_train_stage2 = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomGrayscale(p=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std)
])

transform_train_stage3 = transforms.Compose([
    transforms.Resize((40, 40)),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std)
])
optimizer1 = optim.AdamW(model.parameters(), lr=1e-3, weight_decay=1e-4)
scheduler1 = CyclicLR(optimizer1, base_lr=1e-4, max_lr=1e-2, step_size_up=10, mode='triangular2',

optimizer2 = optim.SGD(model.parameters(), lr=1e-3, momentum=0.9, weight_decay=1e-4)
scheduler2 = CosineAnnealingLR(optimizer2, T_max=30, eta_min=1e-5)

optimizer3 = optim.SGD(model.parameters(), lr=1e-4, momentum=0.9, weight_decay=1e-4)
scheduler3 = ReduceLROnPlateau(optimizer3, patience=5)
```

Рис. 6: Параметры запуска и график точности

После я пробовал запустить без валидации (все под обучение) и с добавлением блюра. Результаты это не сильно улучшило

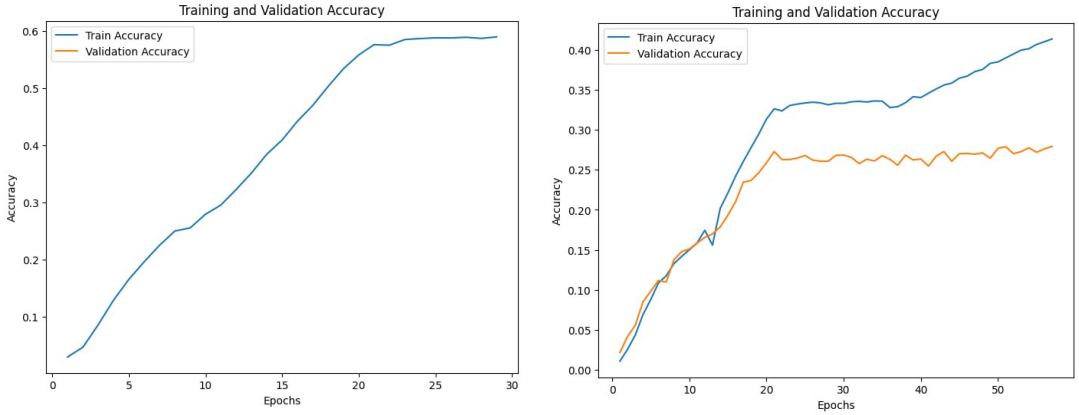


Рис. 7: Дальнейшие игры

5.1 Гиперпараметры

Далее началось жесткое потное и, возможно, безрезультатное перебирание гиперпараметров, аугментаций и прочих прелестей жизни в разных комбинациях. Из новенького я решил попробовать рандомить из CutMix и MixUp, но их добавление не дало прироста, а также попробвать RandomErasing и RandomResizedCrop. Первый я решил оставить в коде, т.к. в целом стоит пробовать отучать нейросетки от вглядывания в части картинки, а вот второй дал мало толку.

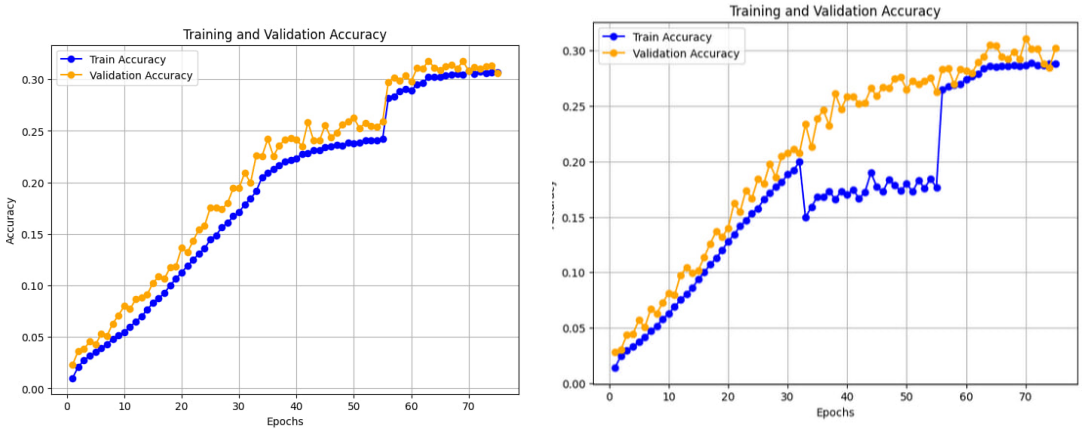


Рис. 8: Запуск с CutMix/Mixup во втором этапе и различием в RandAugment с ColorJitter с Dropout 0,6

Также мне пришло в голову учесть баланс классов на трэйне (оказалось что каждого по 500), а также сделать балансовое разбиение на трэйн и валидацию.

```
full_train_dataset = ImageDataset(labels_csv=train_labels_csv,
    image_dir=train_image_dir, transform=transform_train_stage1)
labels_df = pd.read_csv(train_labels_csv)
labels = labels_df['Category'].values
train_idx, val_idx = train_test_split(np.arange(len(labels)),
                                      test_size=0.02,
                                      random_state=42,
                                      shuffle=True,
                                      stratify=labels)
train_dataset = Subset(full_train_dataset, train_idx)
val_dataset = Subset(full_train_dataset, val_idx)
```

После я пробовал вернуться к изначальному resnet50 и для него пообучать с CutMix и MixUp и увеличить Dropout до 0,7 (получилось не очень) и попробовать заменить ядро первой свертки в моей архитектуре с 3 на 3 в 7 на 7 (из аугментаций на первом этапе RandAugment и флип, потом GS и MixUp/CutMix, затем CutOut + ColorJitter) с Dropout 0,65.

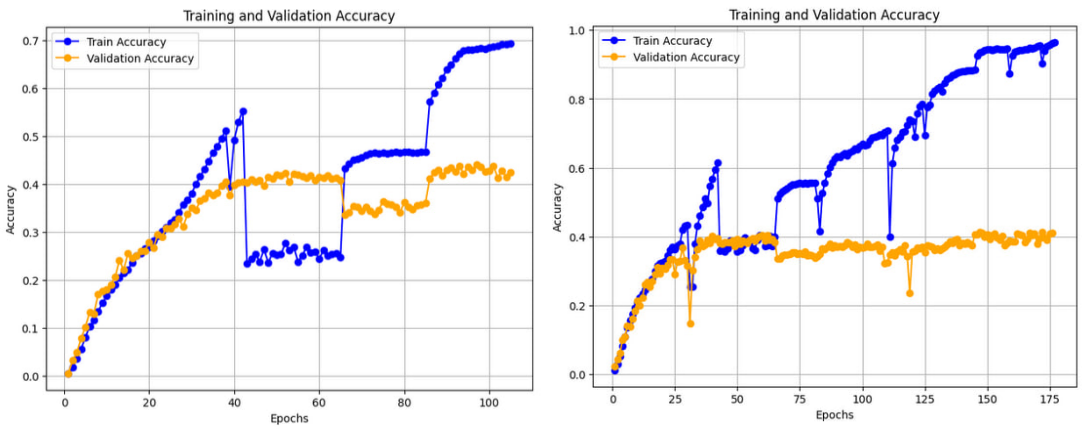


Рис. 9: Запуск с CutMix/Mixup во втором этапе ResNet50 и увеличение ядра

Также вернуться к обучению в 2 этапа (на первом RandAugment, флипы, кроп и Grayscale), а на втором мягкие флипы и grayscale (на трэйне результаты были хорошие, но на тесте всего 44). Ну и обратная ситуация, когда на первом этапе более мягкие аугментации (сравнение ниже).

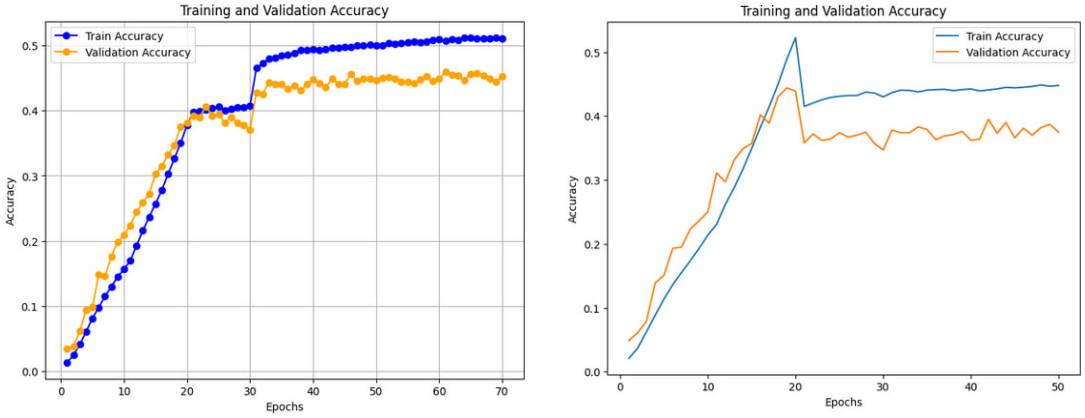


Рис. 10: Запуск в 2 этапа с чередованием жесткости

Наверняка еще что-то было, но я уже не помню...

6 Заключение

Вообще, домашка прикольная - много можно попробовать, потыркать, но я не понимаю, почему не удалось набрать на процента 4 выше. Как ни странно, по-сути простой запуск без хитростей (2 оптимайзера, причем второй не особо понадобился, 1 блок аугментаций в основном с RandAugment) позволяет набрать больше 48, а остальное это пляски с бубном. В качестве основоного кода сдам его.