ЧЕКПОИНТ

В качестве оптимайзера я выбрал SGD(lr=0.1)

Размер батча выбрал 64

Сначала я взял блок из 2 домашки и он дал качество что-то около 20% на валидации на 15 эпохах

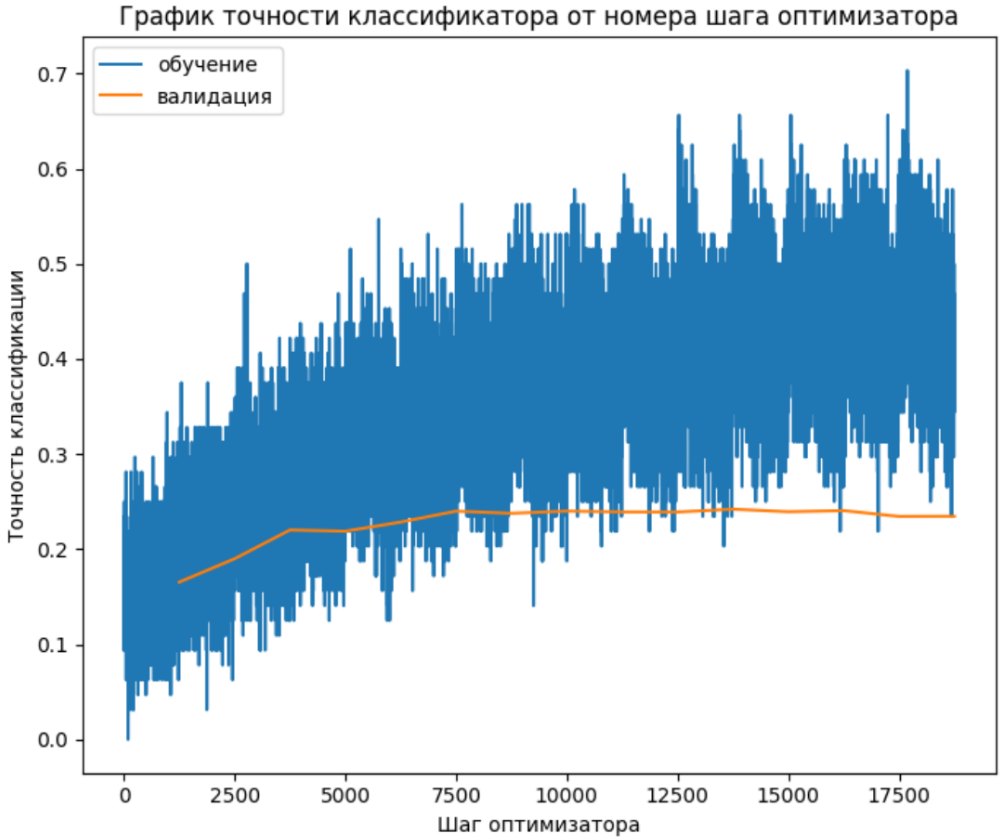
Затем я добавил в сеть второй блок. Получил следующее:

My Net

- SGD(lr=0.1, momentum=0.9)
- batch_size = 64
- 15 эпох

Результаты (везде буду писать accuracy в процентах):

- Train: 44.8%
- Val: 23.5%
- Test: 20.7%



Дальнейшие эксперименты

Затем я решил добавить аугментацию RandomHorizontalFlip и начал эксперименты с ResNet

ResNet18

- RandomHorizontalFlip(0.5)
- SGD(lr=0.01, momentum=0.9)
- batch_size = 64
- 20 эпох

Результаты:

- Train: -
- Val: -
- Test: 28%

На трейне и вале не помню результатов

Затем я посмотрел на архитектуру ResNet18 и увидел, что там в самом начале используется MaxPooling. Для больших картинок он очевидно нужен, но для маленьких - только мешает. Я его убрал

Помимо этого, там есть свёртки с stride = 2. Для маленьких картинок, как у нас, — это не есть хорошо. Я сделал stride = 1 в первых трёх свёртках, где stride равнялся 2

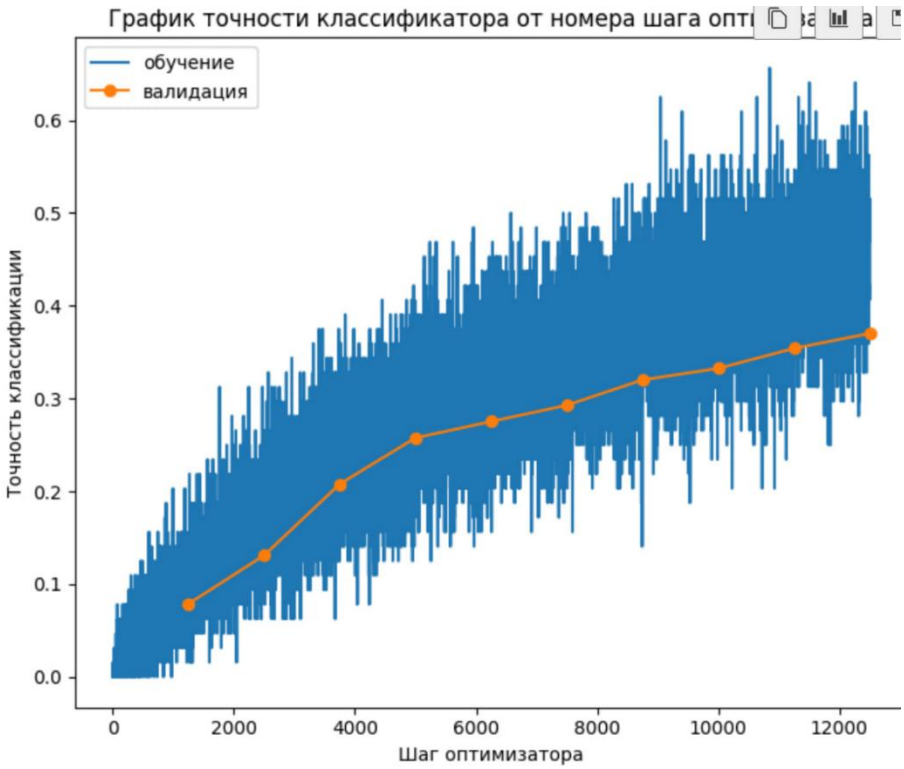
Кроме этого, я решил поиграться с SGD, накинув туда ещё параметров

My ResNet18

- RandomHorizontalFlip(0.5)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 64
- 10 эпох

Результаты

- Train: 44%
- Val: 37%
- Test: 34.8%



Потом я решил поэкспериментировать с EfficientNet. Я поставил efficientnet_v2_s на 10 эпох и сделал stride = 1 в первых двух свёрточных слоях со stride = 2

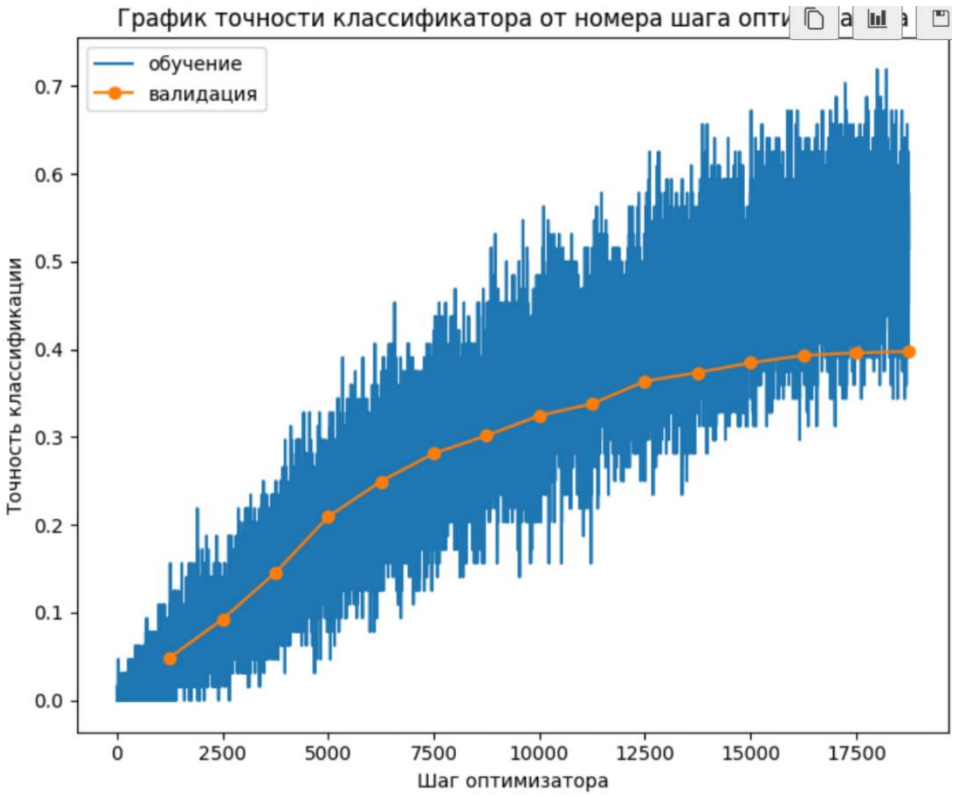
Помимо этого я добавил шедулер CosineAnnealingLR(T_max = 15). Сделал это потому, что заметил, что на поздних эпохах возникает переобучение, попытался так с ним бороться, плавно уменьшая lr

My efficientnet_v2_s

- RandomHorizontalFlip(0.5)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 64
- CosineAnnealingLR(T_max = 15)
- 15 эпох

Результаты

- Train: 51.9%
- Val: 39.8%
- Test: 37.56%



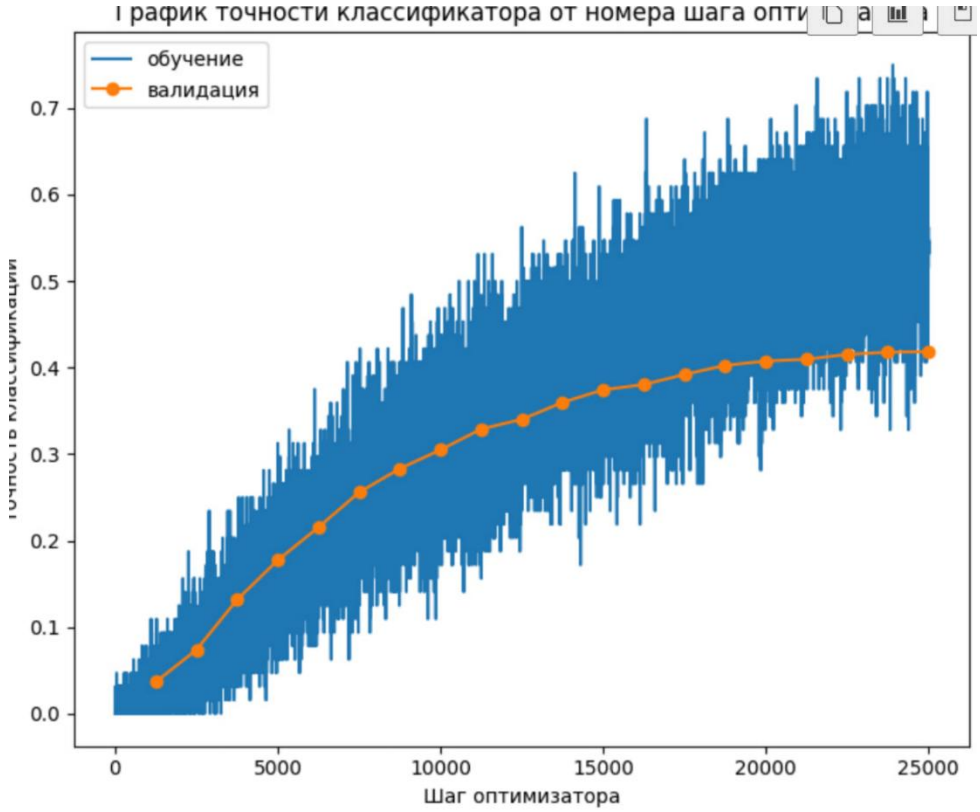
Затем я решил изменить dropout перед линейным слоем с 0.2 на 0.4 (чтобы снизить переобучение) и увеличить количество эпох до 20. А также добавил ещё одну аугментацию - RandomSolarize

****My efficientnet_v2_s****

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomSolarize(0.4, 0.5)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 64
- CosineAnnealingLR(T_max = 20)
- dropout = 0.4
- 20 эпох

Результаты

- Train: 55%
- Val: 41.8%
- Test: 39.6%



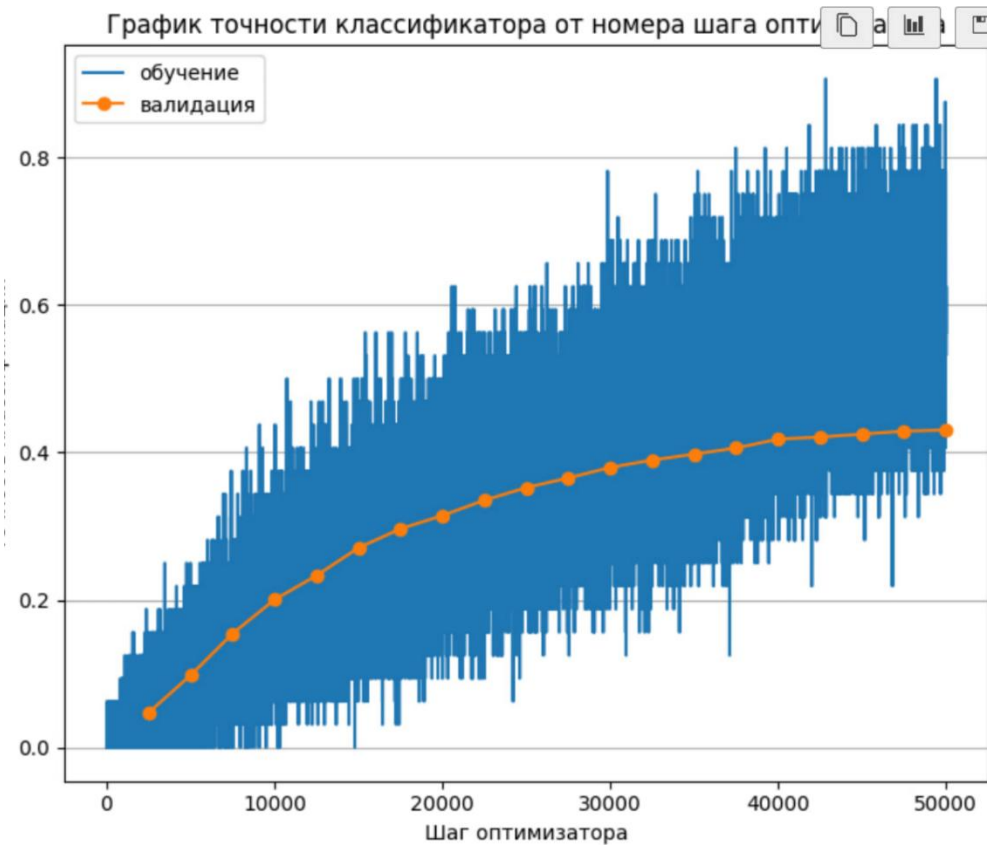
Я решил сделать batch_size = 32

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomSolarize(0.4, 0.5)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 20)
- dropout = 0.4
- 20 эпох

Результаты

- Train: 59.8%
- Val: 43%
- Test: 40.5%



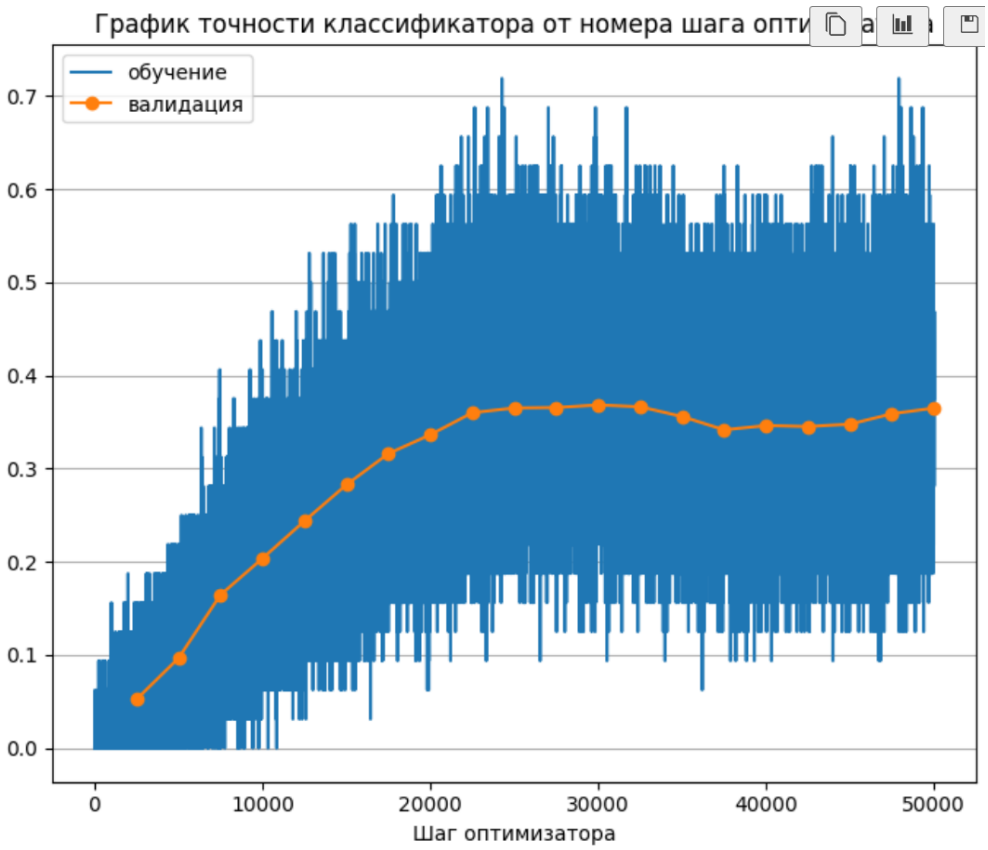
Дальше я посмотрел на динамику изменения lr и качества на трейне и валидации. Я увидел, что CosineAnnealingLR плохо справляется со своей задачей и не успевает достаточно понизить lr к 11-й эпохе (тогда начинается переобучение). Я решил попробовать поставить параметр T_max = 10. Но от этого стало только хуже

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomSolarize(0.4, 0.5)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 10)
- dropout = 0.4
- 20 эпох

Результаты

- Train: 37.6%
- Val: 36.5%
- Test: -



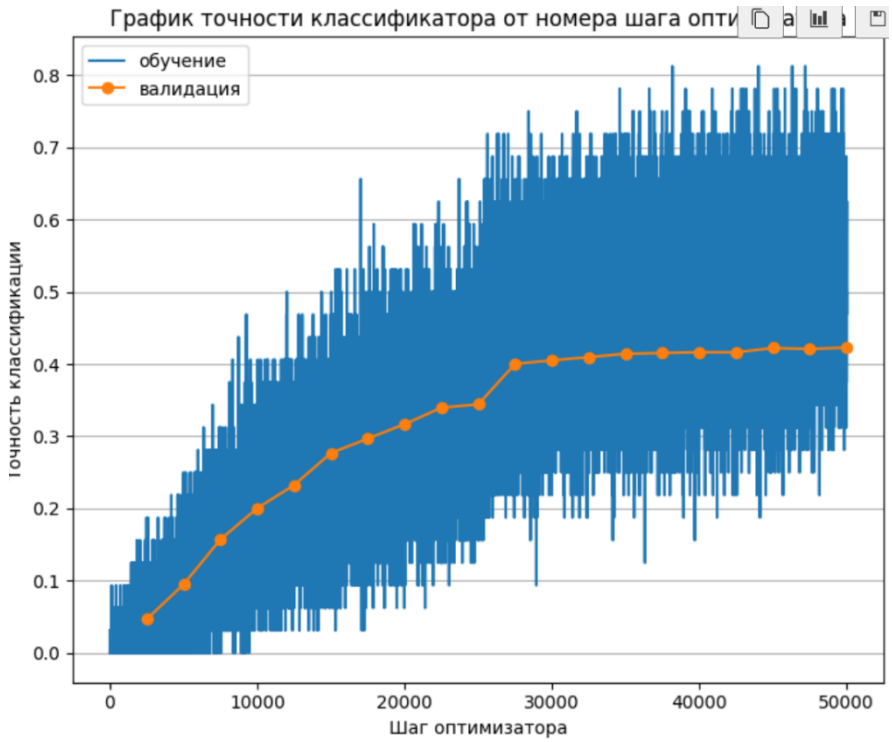
Я решил попробовать MultuStepLr. Я увидел, что на 11-й эпохе наступает переобучение. Я попробую уменьшить на 11-й эпохе lr в 10 раз и посмотреть что будет

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomSolarize(0.4, 0.5)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- MultiStepLR([10], gamma=0.1)
- dropout = 0.4
- 20 эпох

Результаты

- Train: 52%
- Val: 42.3%
- Test: 39%



Как видим на 11-й эпохе, когда понижаем lr то получаем хороший скачок качества вверх. Я попробовал ещё поиграться с эпохой, на которой понижаем lr. Но всё равно это приводило к переобучению. В итоге все-таки решил оставить понижение lr на 11-й эпохе, потому что мне нравится этот скачок качества

Я решил попробовать заменить аугментацию RandomSolarize(0.4, 0.5) на RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6). Она смещает картинку на 1/8 от размера. Также я решил попробовать уменьшить lr на 12-й эпохе, вдруг это тоже даст скачок качества

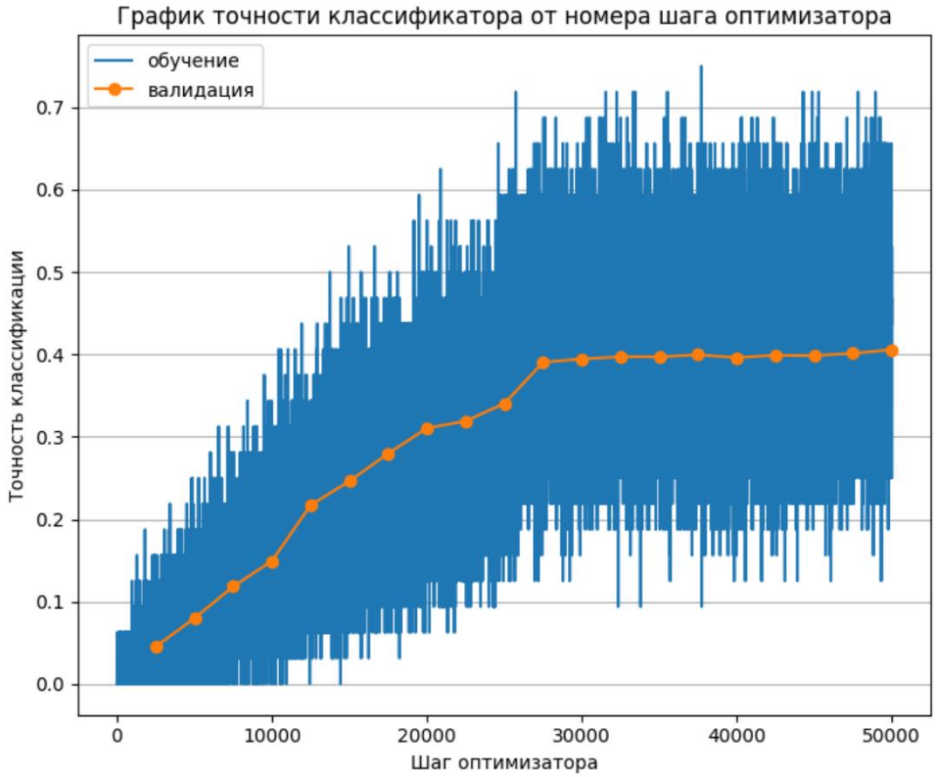
Чтобы повысить качество на тесте я решил попробовать сделать Test-Time augmentation. Я прогонял одну и ту же картинку 5 раз с аугментациями и выбирал итоговый класс тот, к которому картинка относится чаще всего

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- MultiStepLR([10, 11], gamma=0.1)
- dropout = 0.4
- 20 эпох

Результаты

- Train: 42.3%
- Val: 40.6%
- Test: 37.6%



Как видно мы выходим на плато. Надо что-то менять

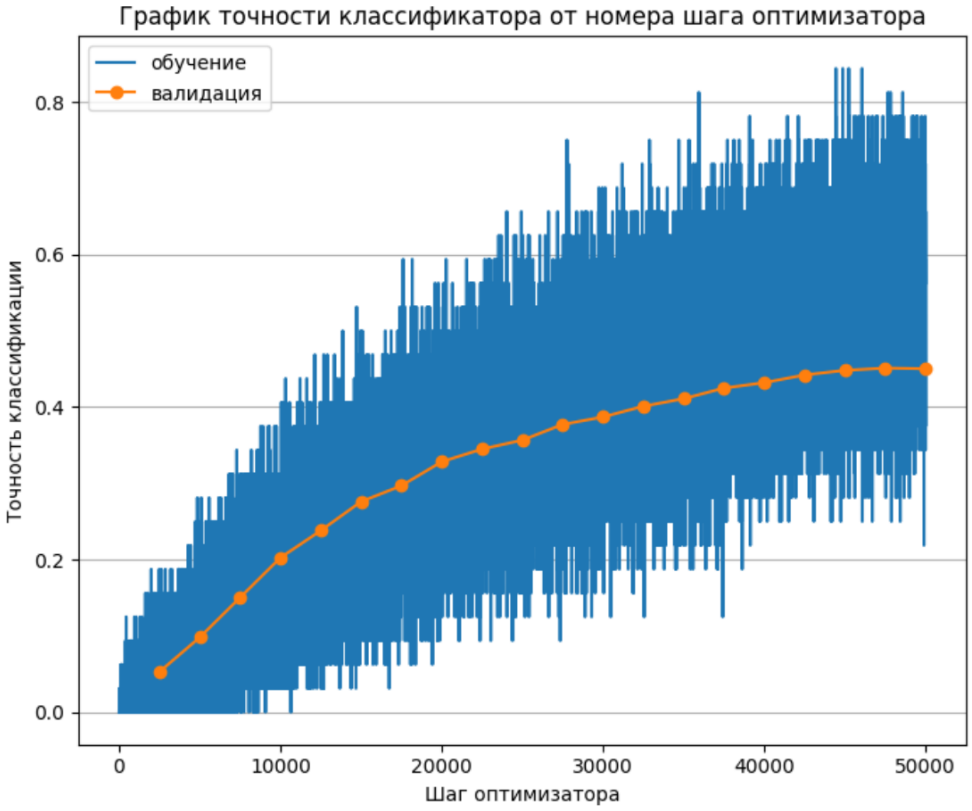
Я решил вернуться к модели, которая давала 43% на валидации. Я решил сделать в ней такую же замену аугментации и использовать Test-Time augmentation

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 20)
- dropout = 0.4
- 20 эпох

Результаты

- Train: 54.7%
- Val: 45%
- Test (with test-time aug): 41.8
- Test: 41.48



Как мы видим переобучение стало меньше, при этом качество на валидации и тесте стало выше. При этом можно увидеть, что test-time augmentation действительно помогает, буду в дальнейшем только его и использовать

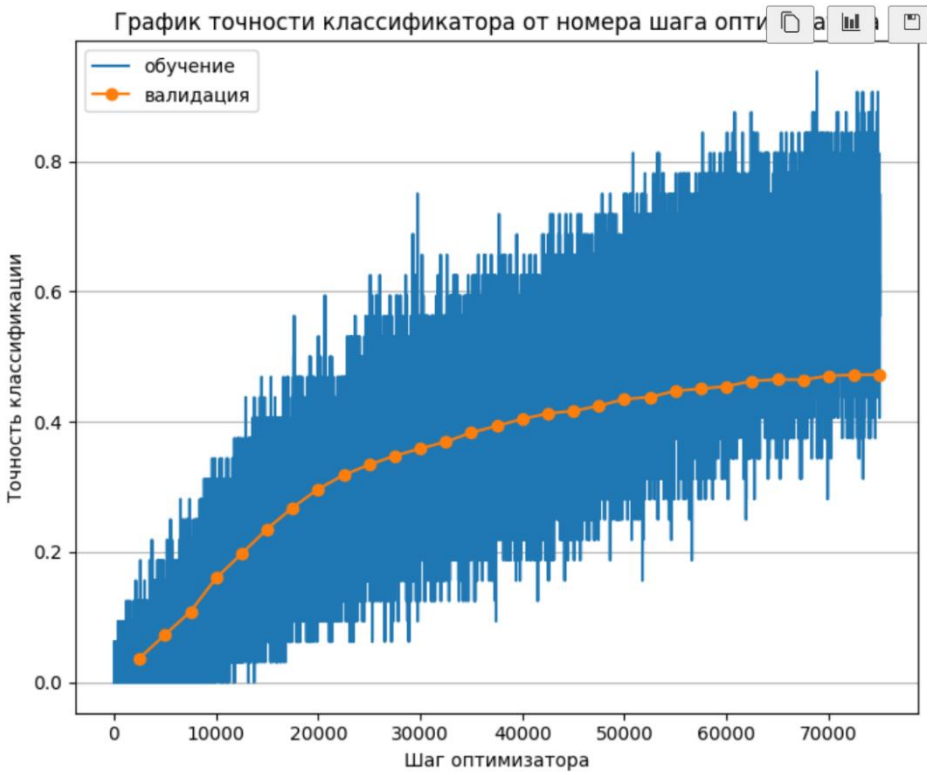
Я решил попробовать сделать dropout=0.6 и поставить 30 эпох обучения

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 30)
- dropout = 0.6
- 30 эпох

Результаты

- Train: 64.2%
- Val: 47.3%
- Test: 44.6%



Хочу попробовать поменять нормализацию. Я сейчас не учитываю распределение пикселей, просто считаю, что они равномерно распределены от 0 до 256, но это не так. По идее разумнее взять дисперсию и среднее

means = (0.5669, 0.5426, 0.4914)

stds = (0.2377, 0.2326, 0.2506)

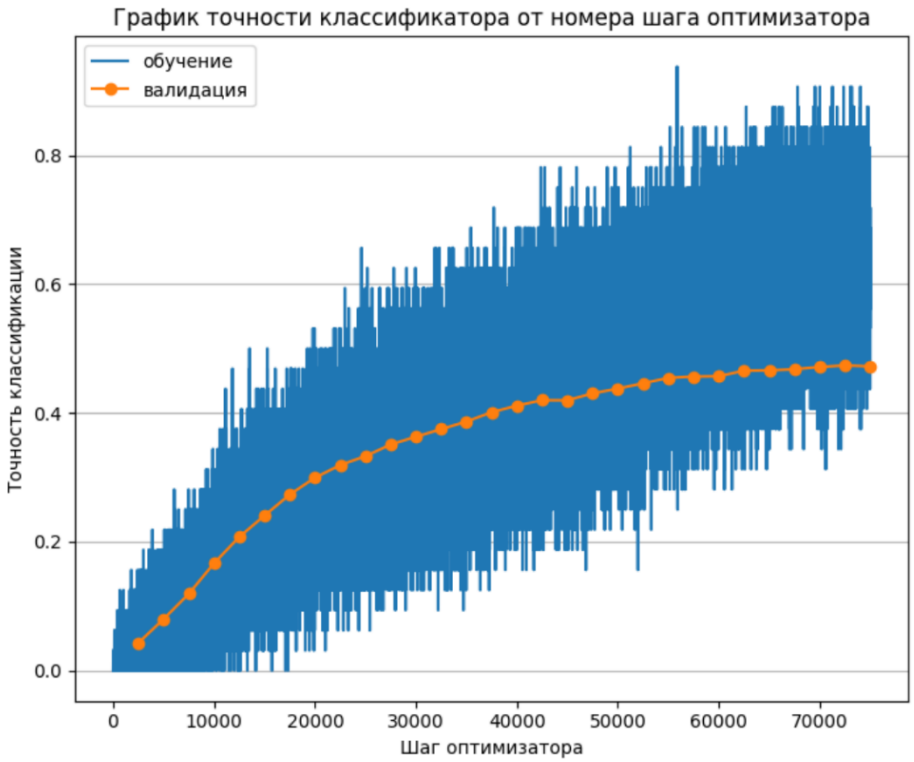
Вот такие значения у меня получились, код можно посмотреть в ноутбуке,

My efficientnet_v2_s

- Аугментации:
- RandomHorizontalFlip(0.5)
    - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
    - SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 30)
- dropout = 0.6
- 30 эпох

Результаты

- Train: 64.5%
- Val: 47.2%
- Test: -



Несмотря на то, что качество на валидации чуть ниже, была эпоха (29-я) когда качество было выше, чем в предыдущем эксперименте

```
100%|██████████| 2500/2500 [05:22<00:00,  7.75it/s]
100%|██████████| 625/625 [00:24<00:00, 26.02it/s]
Epoch 29 | lr=0.00010926199633097158
 train loss: 1.3681669235229492, train acc: 0.6414374709129333
 val loss: 2.316840887069702, val acc: 0.4738500118255615
```

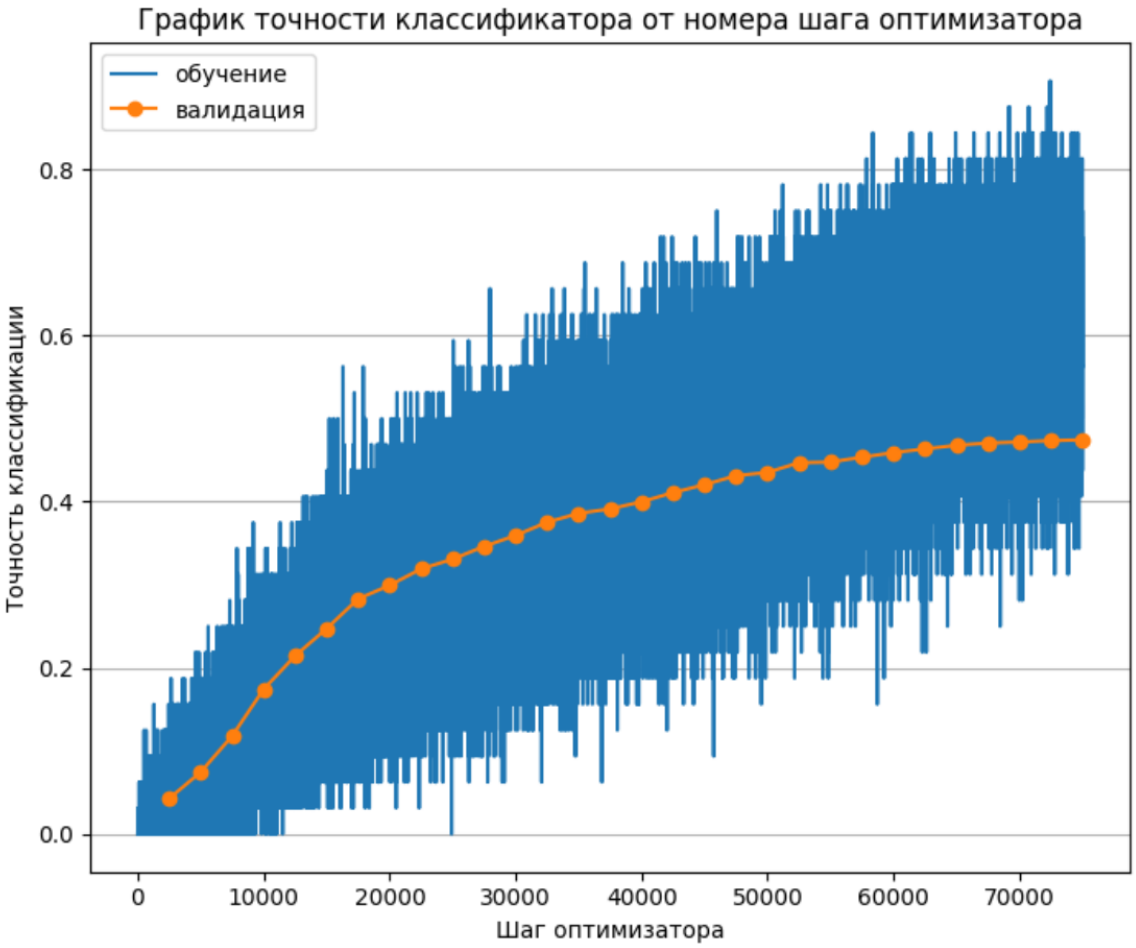
Добавим аугментацию поворотом картинки на 10 градусов

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
  - RandomApply([RandomAffine(degrees=10)], p=0.6)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 30)
- dropout = 0.6
- 30 эпох

Результаты

- Train: 60.8%
- Val: 47.4%
- Test: -



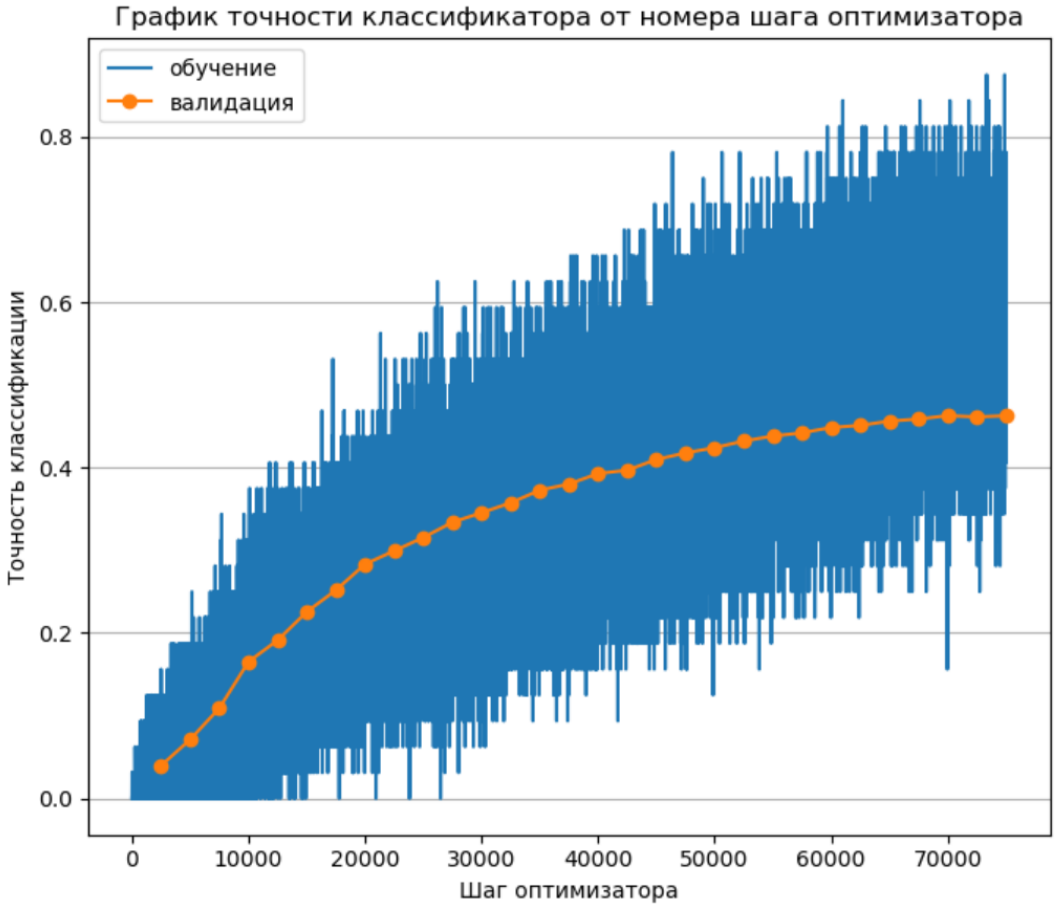
Качество выросло! Добавим еще одну аугментацию - гауссовый блюр

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
  - RandomApply([RandomAffine(degrees=10)], p=0.6),
  - RandomApply([GaussianBlur(3)], p=0.3)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 30)
- dropout = 0.6
- 30 эпох

Результаты

- Train: 57%
- Val: 46.3%
- Test: -



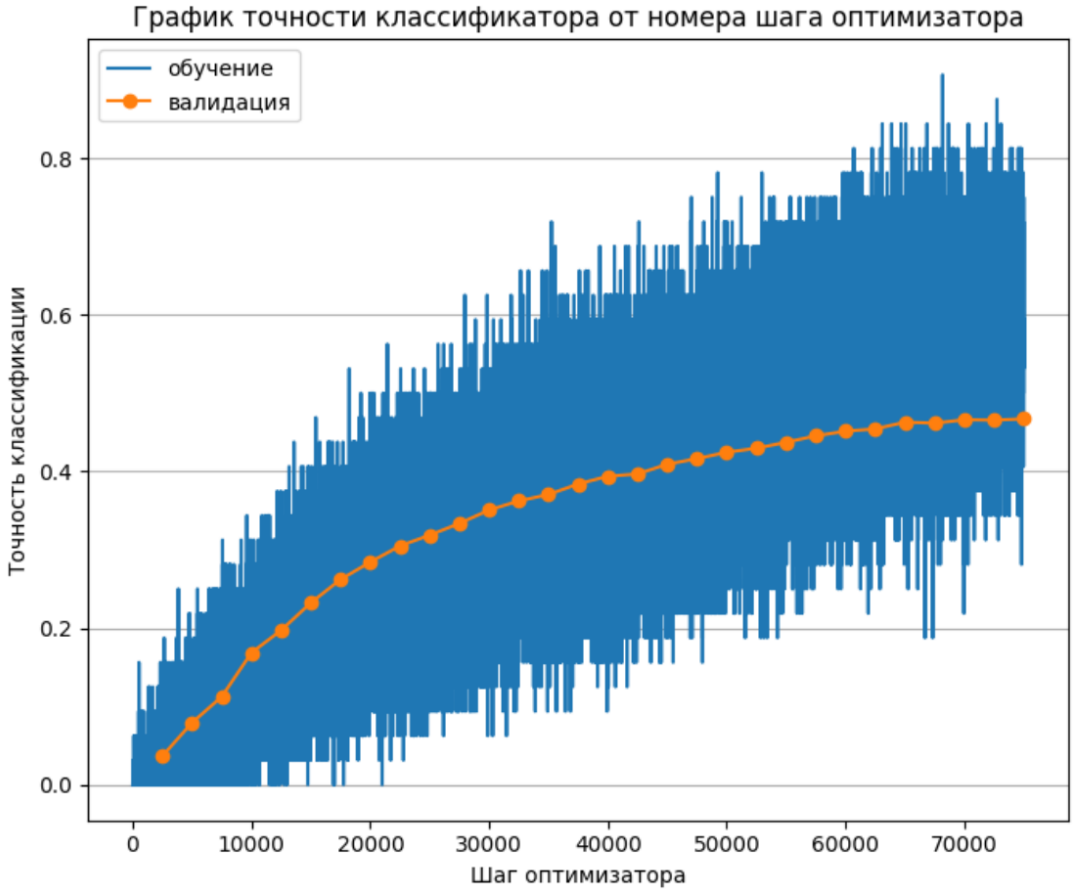
Переобучение уменьшилось, но качество на валидации тоже. Попробуем зафиксировать sigma=0.6

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
  - RandomApply([RandomAffine(degrees=10)], p=0.6),
  - RandomApply([GaussianBlur(3, 0.6)], p=0.3)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 30)
- dropout = 0.6
- 30 эпох

Результаты

- Train: 58.2%
- Val: 46.7%
- Test: -



Качество выросло по сравнению с предыдущим запуском!

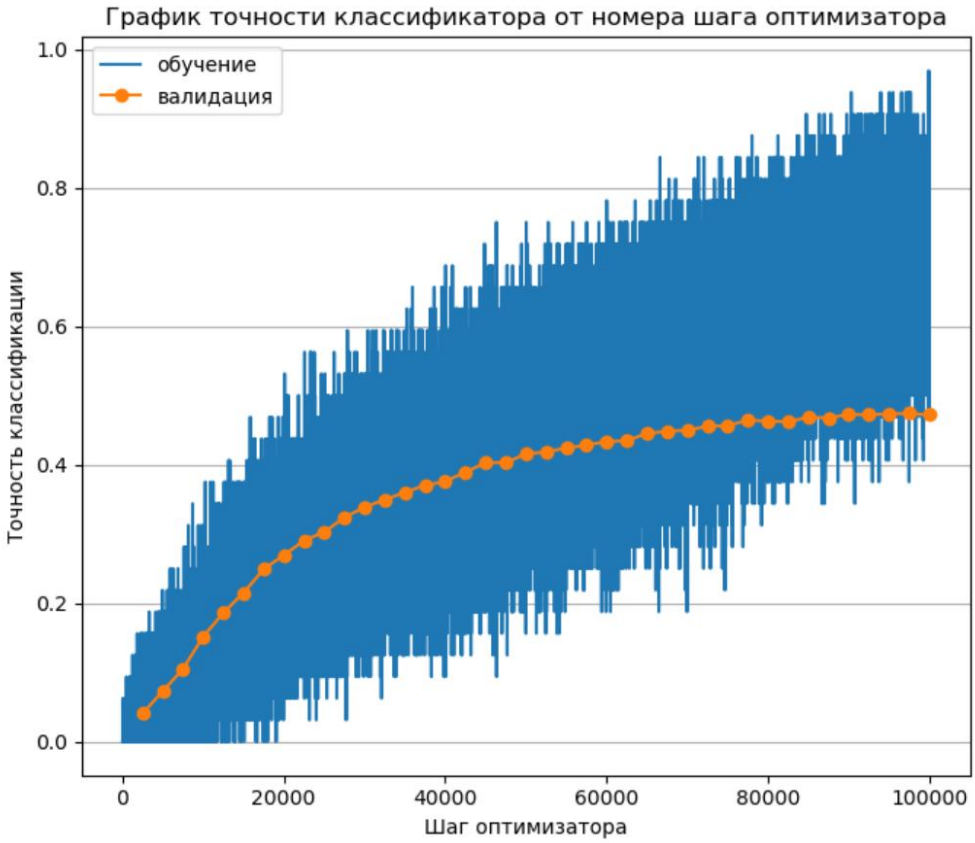
Попробую поставить 40 эпох

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
  - RandomApply([RandomAffine(degrees=10)], p=0.6),
  - RandomApply([GaussianBlur(3, 0.6)], p=0.3)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 40)
- dropout = 0.6
- 40 эпох

Результаты

- Train: 69.7%
- Val: 47.2%
- Test: 44.96%



Качество на тесте выросло, но это потому, что я в test-time augmentation прогонял картинку 10 раз, а не 5

Как мы видим, аугментация сделала только хуже и ничего не смогло ей помочь сделать что-то лучше

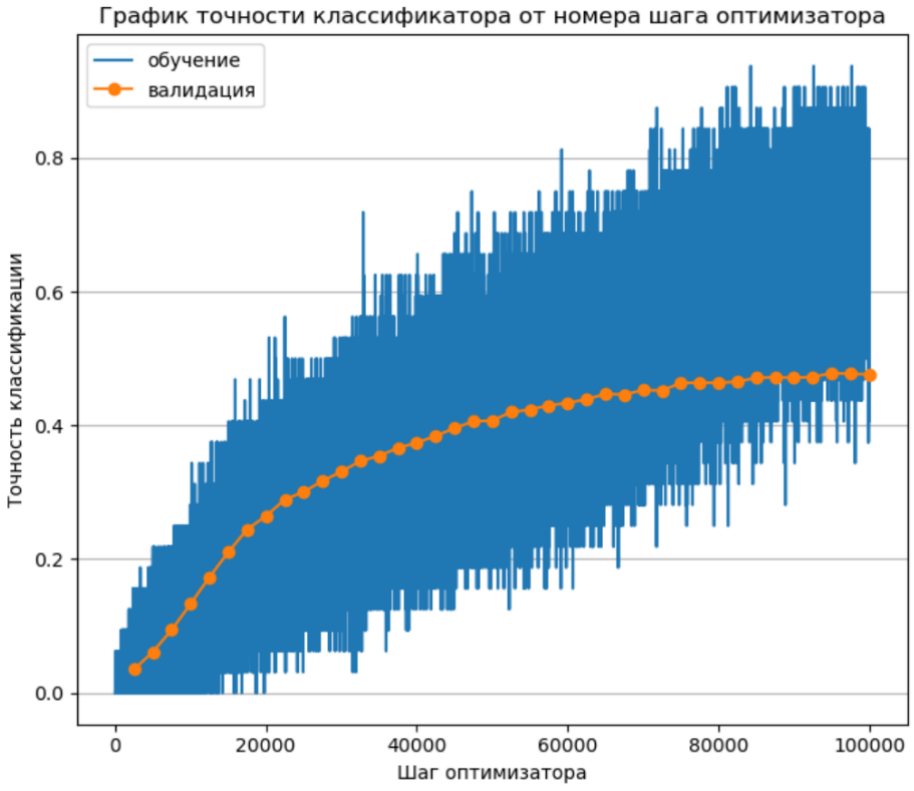
Попробуем запустить на 40 эпох поставив dropout=0.7

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.6)
  - RandomApply([RandomAffine(degrees=10)], p=0.6)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 40)
- dropout = 0.7
- 40 эпох

Результаты

- Train: 67.8%
- Val: 47.7%
- Test: 45.74%



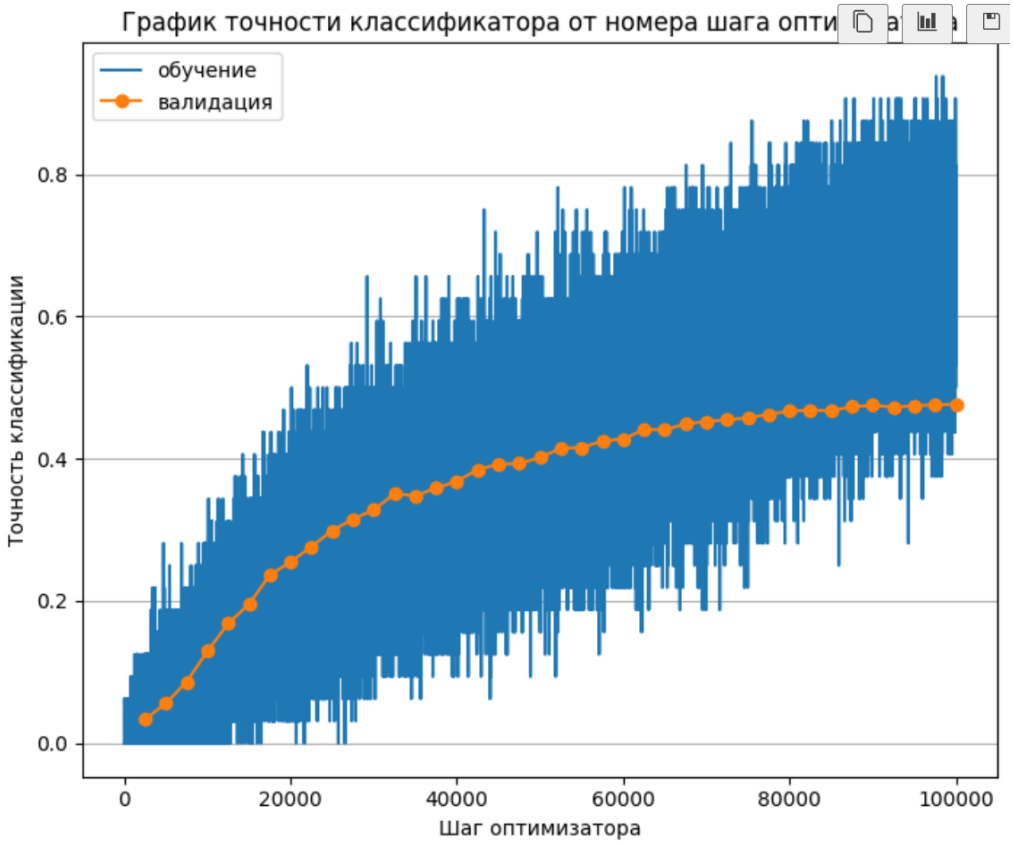
Попробуем увеличить вероятность применения аугментаций до 0.8, это должно помочь уменьшить переобучение

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.8)
  - RandomApply([RandomAffine(degrees=10)], p=0.8)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 40)
- dropout = 0.7
- 40 эпох

Результаты

- Train: 65.8%
- Val: 47.7%
- Test: -



Как мы видим на валидации качество осталось неизменным, а вот переобучение чуть упало - победа

Попробуем использовать другой шедулер. Меня заинтересовал ReduceLROnPlateau. Как только accuracy на валидации выходит на плато - будем уменьшать LR в 10 раз

В голове идея звучало классно, но на деле оказалась фигней: я провел несколько запусков и в каждом шедулер очень быстро по сути занулял lr, что не давало модели обучаться

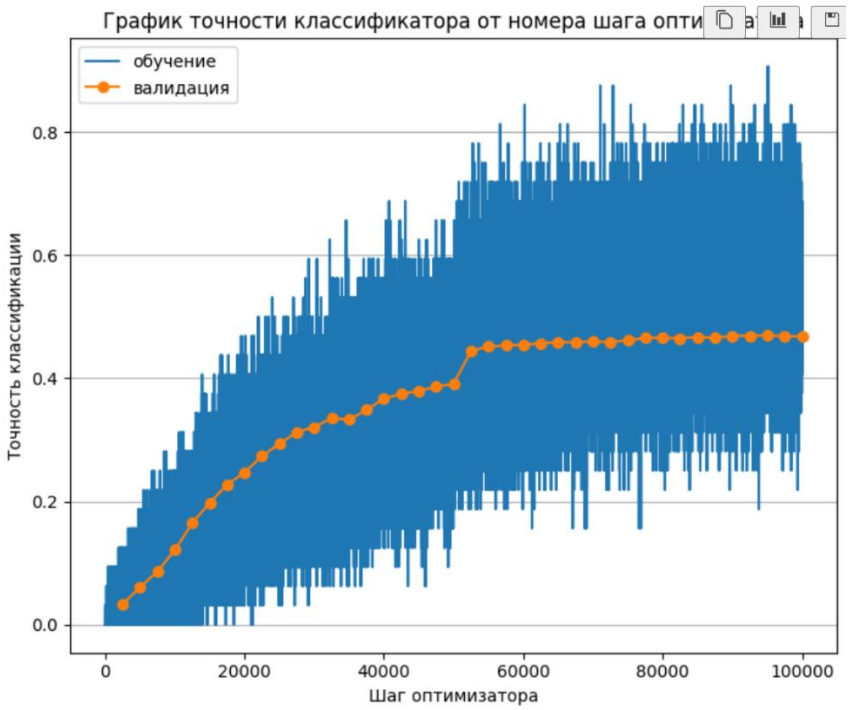
Я решил дать еще один шанс MultiStepLR

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.8)
  - RandomApply([RandomAffine(degrees=10)], p=0.8)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- MultiStepLR([20, 30])
- dropout = 0.7
- 40 эпох

Результаты

- Train: 54.3%
- Val: 46.8%
- Test: -



Все же нет, плохая была это идея

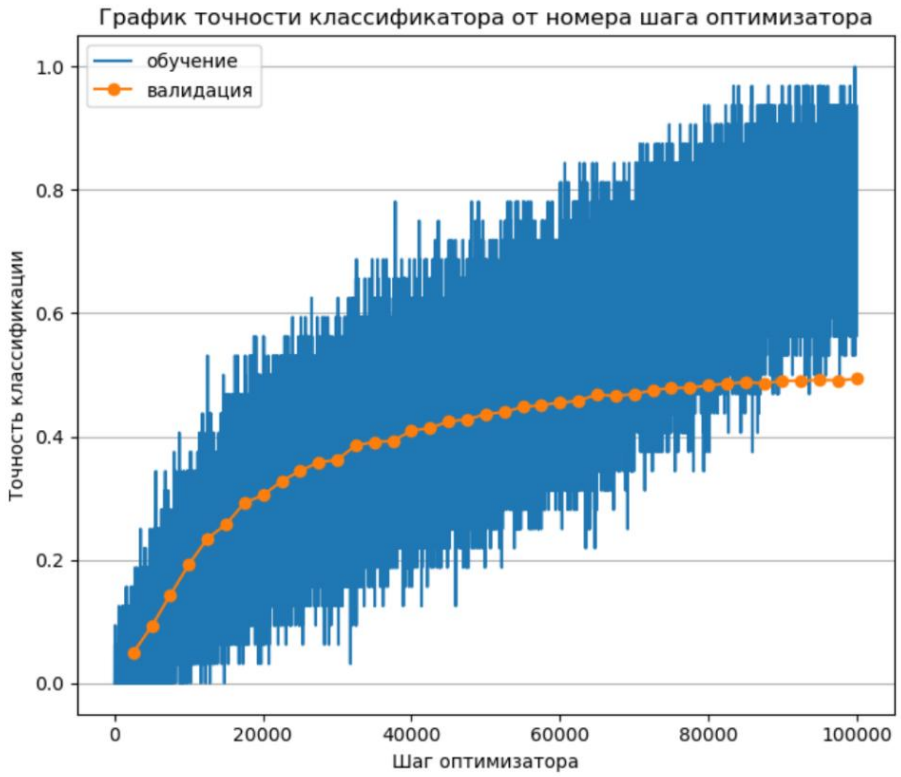
Пока читал статьи я наткнулся на ShakeDrop. Я решил попробовать поставить его куда-то в центр сети и посмотреть, что получится

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.8)
  - RandomApply([RandomAffine(degrees=10)], p=0.8)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- CosineAnnealingLR(T_max = 40)
- ShakeDrop(0.5)
- dropout = 0.4
- 40 эпох

Результаты

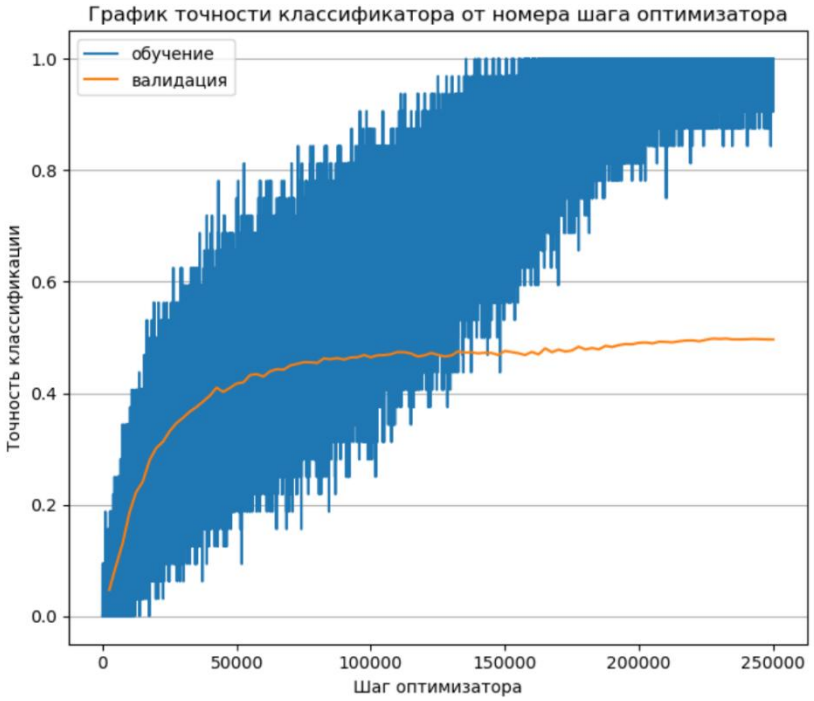
- Train: 76.1%
- Val: 49.4%
- Test: -



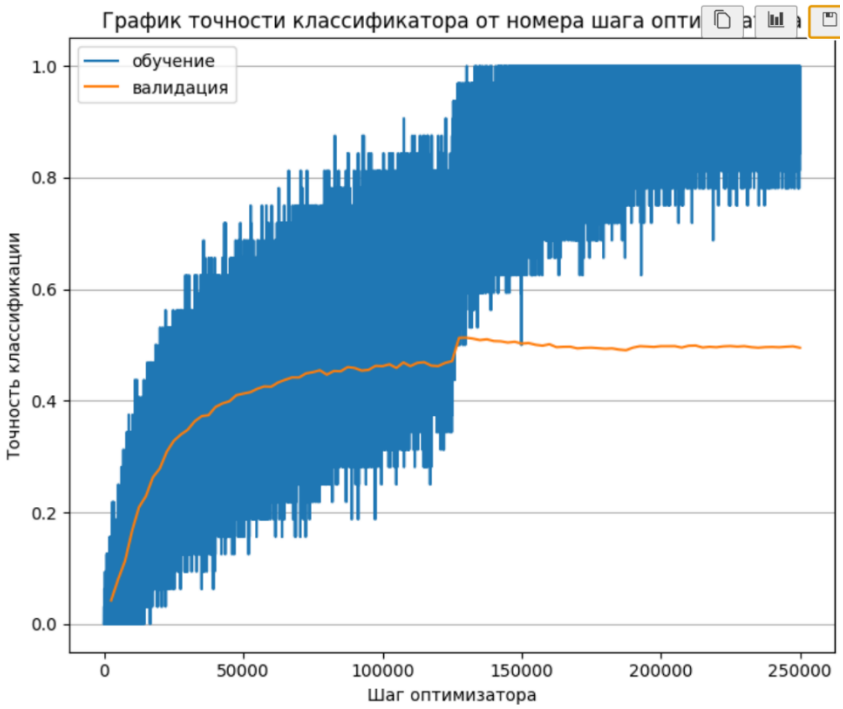
Качество на валидации выросло!

Попробуем увеличить количество эпох до 100 и сравнить 2 шедулера - CosineAnnealingLR(T_max = 100) и MultiStepLR([50,75])

CosineAnnealingLR:



MultiStepLR (Test: 47.06%):



Оба шедулера рано или поздно выходят на плато по качеству на валидации, и дико переобучаются после 50 эпох

Однако середине обучения лучше всех себя показал MultiStepLR. Он выдал качество на валидации 51,3% на 52 эпохе

```
100%|██████████| 2500/2500 [06:50<00:00,  6.08it/s]
100%|██████████| 625/625 [00:25<00:00, 24.41it/s]
Epoch 52 | lr=0.001
 train loss: 0.8210720419883728, train acc: 0.7749750018119812
 val loss: 2.3251969814300537, val acc: 0.5131999850273132
```

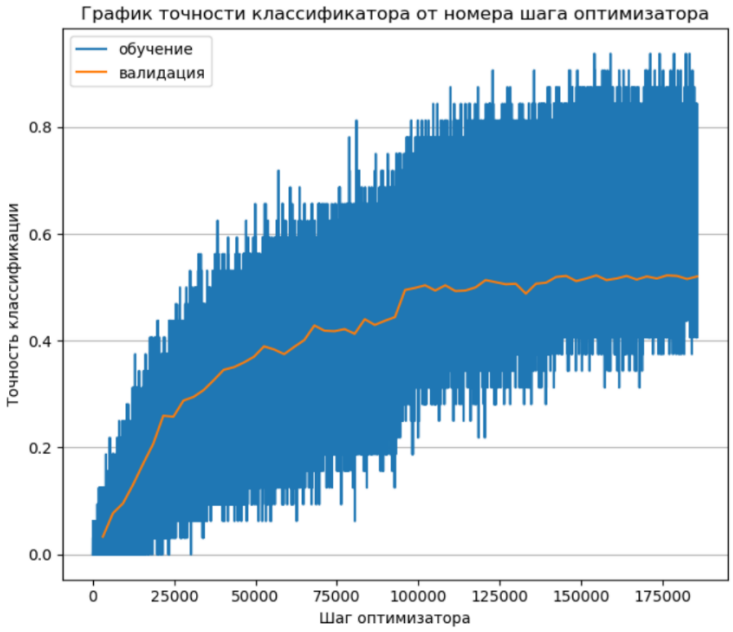
Оставим MultiStepLR и попробуем уменьшить количество эпох и увеличить dropout

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.8)
  - RandomApply([RandomAffine(degrees=10)], p=0.8)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- MultiStepLR([30, 45])
- ShakeDrop(0.5)
- dropout = 0.8
- 60 эпох

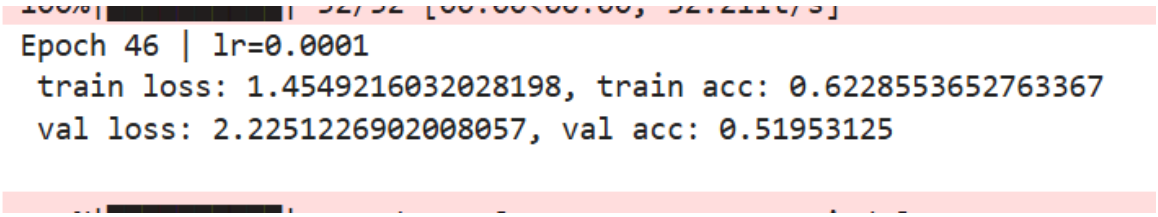
Результаты

- Train: 65.2%
- Val: 52%
- Test: 48.4%



На разных этапах обучения я сохранял веса модели, что позволило мне протестировать не только финальный вариант

Например, вот 46 эпоха обучения:



Я обнаружил, что на ней качество на тесте составляет 49.06%

Это лучший результат, который у меня был. Получаем итог:

My efficientnet_v2_s

- Аугментации:
  - RandomHorizontalFlip(0.5)
  - RandomApply([RandomAffine(degrees=0, translate=(1/8,1/8))], p=0.8)
  - RandomApply([RandomAffine(degrees=10)], p=0.8)
- SGD(lr=0.01, momentum=0.9, weight_decay=1e-4, nesterov=True)
- batch_size = 32
- MultiStepLR([30, 45])
- ShakeDrop(0.5)
- dropout = 0.8
- 46 эпох

Результаты

- Train: 65.2%
- Val: 52%
- Test: 49.06%