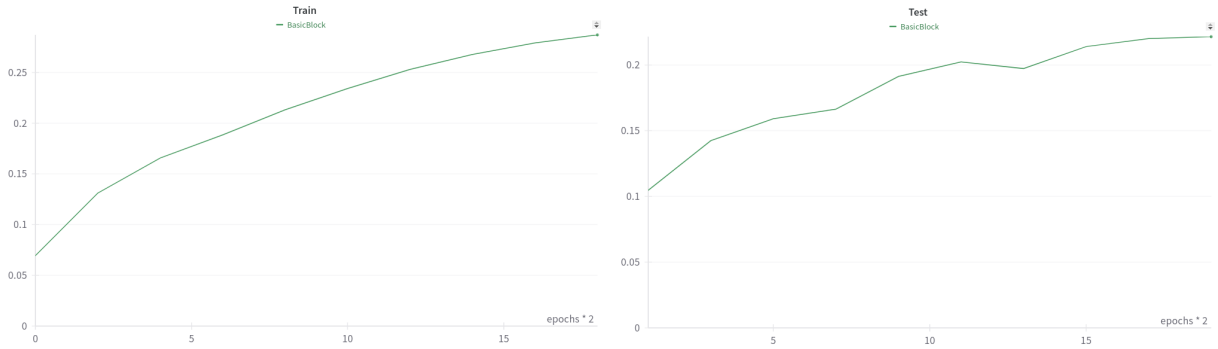
Отчет по большому домашнему заданию
Глубинное Обучение 1

Панфилов Борис
ФКН ВШЭ

21 января 2024 г.

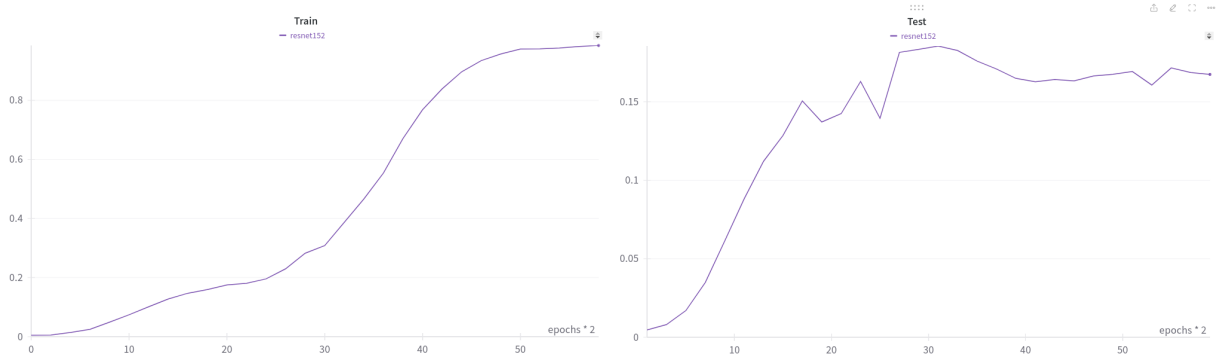
1 Начало

В самом начале этого задания нужно было выбить accuracy хотя бы 10%. Для этого я воспользовался арихтектурой из второй маленькой домашки. Это был один блок резнета.



На графике видно, что качество на тесте по итогу 20 эпох находится в районе 23%, что меня полностью устроило.

Далее первое, что я решил попробовать - запустить большую модель на много эпох, ведь чем сложнее модель, тем больше шансов у нее выучить все закономерности, которые можно выделить на основе тренировочной выборки. Ну а вдруг. В качестве большой моделей я взял ResNet152 и запустил обучение на 30 эпох.

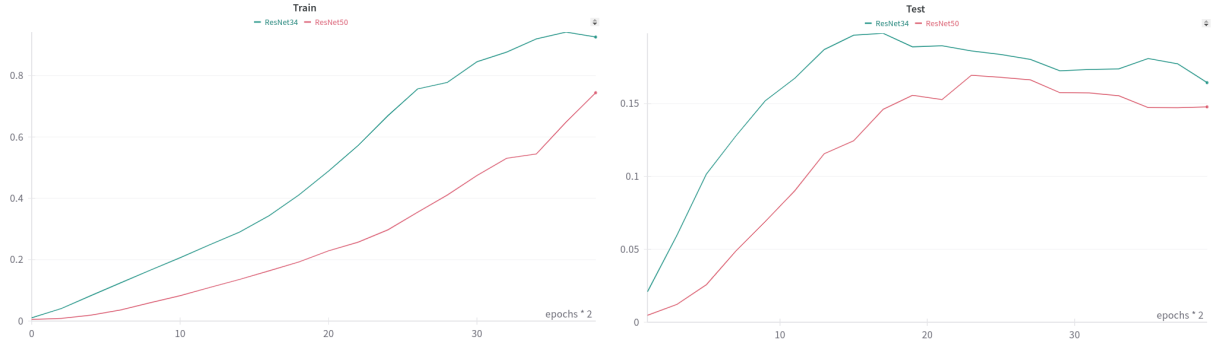


Анализ:

1. Модель очень сильно переобучилась.
2. Качество не поднималось выше 20%.

Поэтому далее я перешел к экспериментам с более маленькими моделями. Мой взгляд упал на ResNet34 и ResNet50.

Для начала я запустил их в максимально ванильном варианте - без расписания для lr и аугментаций на 20 эпох. Зеленый - ResNet34, красный - ResNet50.

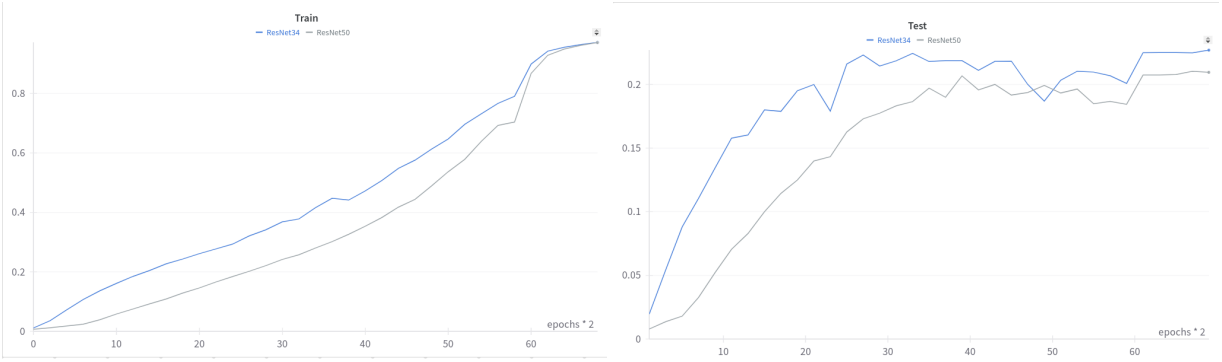


Анализ:

1. Обе модели переобучились, потому что качество на тесте достигло пика и начало уменьшаться.

2. Качество не поднималось выше 20%.

Далее я решил увеличить количество эпох до 35 и добавить расписание для lr. В качестве расписания я поставил уменьшение lr в 10 раз после 30 эпох. Голубой - ResNet34, серый - ResNet50.

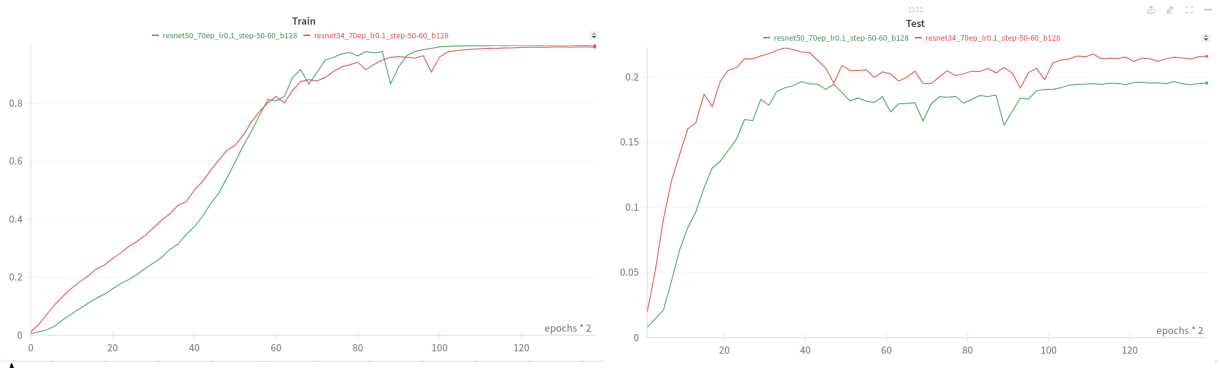


Анализ:

1. Качество улучшилось. По окончании обучения оно составляет где-то 23%

2. Тем не менее переобучение никуда не делось.

Я по жизни максималист, поэтому решил еще увеличить количество эпох и посмотреть что получиться. Итак, увеличил количество эпох до 70 и поставил уменьшение lr в 10 раз после 50 и 60 эпох. Красный - ResNet34, зеленый - ResNet50.



Анализ:

1. Видно, что качество упало в сравнении с предыдущими результатами. Теперь акураси на валидации в районе 20%.

2. А вот на трейне качество подскочило до 1, то есть переобучение стало только более явным.

2 Самописные модели

В силу того, что мы работаем с картинками всего 40*40, а по дефолту резнеты предназначены для картинок 224*224 приходит в голову идея, что нужно немного подкрутить архитектуру и станет лучше.

§2.1 MyResNets

Первое изменение, которое я сделал - убрал слой MaxPooling'а после первой свертки. Он точно лишний в самом начале пайплана нейросети, потому что уменьшает картинку в 2 раза. Это много, учитывая, что у нас всего 40*40 пикселей на входе.

Плюс, по опыту предыдущих запусков было понятно, что нужно начинать использовать какие-нибудь средства регуляризации. Поэтому я начал эксперементировать с разными аугментациями. В качестве расписания для lr я решил брать косинусоидное. С ним все нормально училось. Только почему-то я выставлял initial_lr=0.5, что сыграло со мной злую шутку, рассказ об этом будет далее. Теперь к результатам обучений.

Описание запуска:

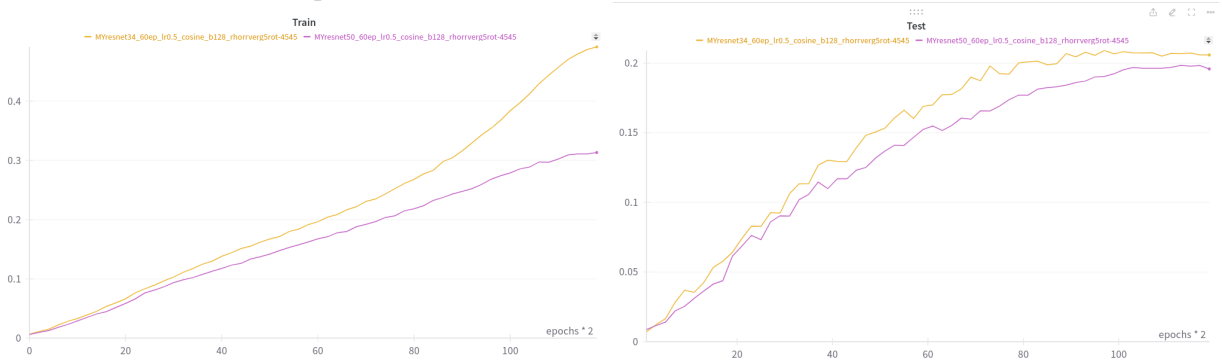
1. augmentations: жесткие(флипы, блюр с ядром 5 и повороты на 45 градусов)

2. lr: 0.5

3. scheduler: CosineAnnealingLR

4. batch size: 128

Желтый - ResNet34, фиолетовый - ResNet50.



Анализ:

1. ResNet50 перестал переобучаться, тем не менее на плато выходит при качестве в 20 процентов, чего маловато.

2. ResNet34 по прежнему переобучается, либо его обобщающей способности не хватает.

Далее я решил сделать аугментации немного попроще: уменьшил угол, на который могут поворачиваться картинки до 15, уменьшил ядро блюра.
Описание запуска:

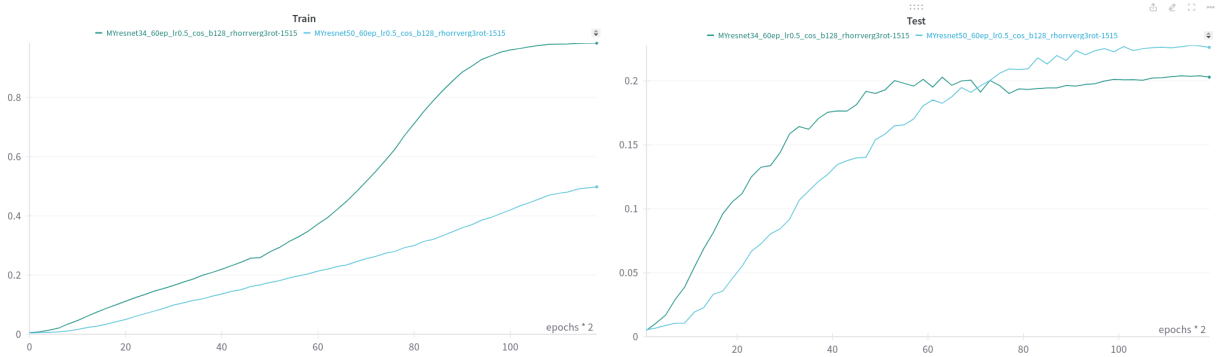
1. augmentations: средние(флипы, блюр с ядром 3 и повороты на 15 градусов)

2. lr: 0.5

3. scheduler: CosineAnnealingLR

4. batch size: 128

Зеленый - ResNet34, голубой - ResNet50.



Анализ:

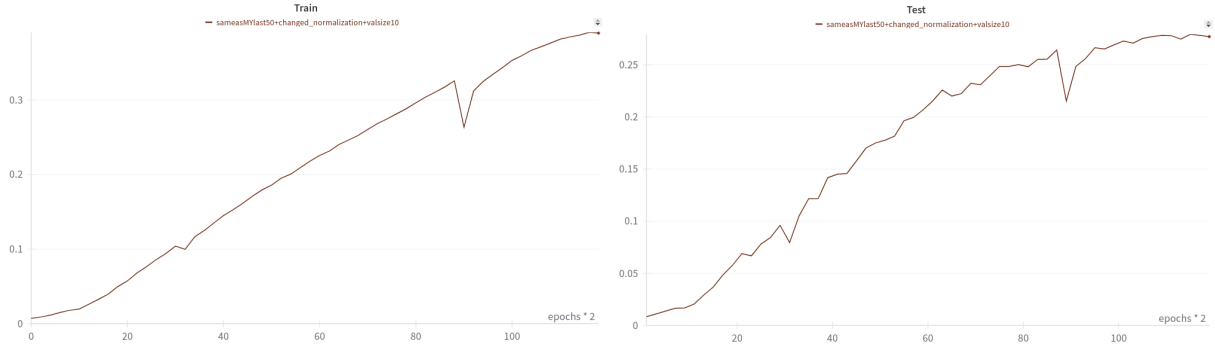
1. ResNet34 переобучился еще больше, что логично, потому что мы ослабили регуляризацию.

2. ResNet50 начал учиться лучше и показал качество в 25%, что пока что является наилучшим результатом.

Далее я решил работать с ResNet50, потому что качество на нем получалось лучшее. Заметил, что нормализую картинки используя v2.Normalize((0.5, 0.5, 0.5),

(0.5, 0.5, 0.5)), хотя есть насчитанные на ImageNet'е статистики, которые являются более точными. С этого момента я начал использовать их, вот так выглядит итоговая нормализация: v2.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)).
Описание запуска

1. augmentations: средние(флипы, блюр с ядром 3 и повороты на 15 градусов)
2. lr: 0.5
3. scheduler: CosineAnnealingLR
4. batch size: 128

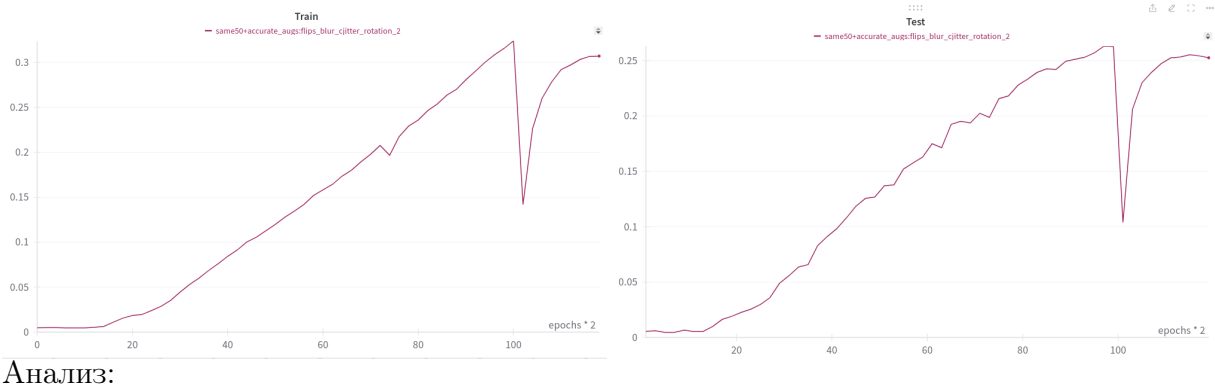


Анализ:

1. Видим, что качество еще улучшилось и достигло 28 процентов.
2. Появился странный провал после 44 эпохи

Далее я решил попробовать другие аугментации, состоящие из флипов, поворотов, блюра и color jitter'а. Тут я уменьшил вероятность первых 4 преобразований, но добавил color jitter для усложнения.
Описание запуска

1. augmentations: средние(флипы, блюр с ядром 3 и повороты на 15 градусов, color jitter)
2. lr: 0.5
3. scheduler: CosineAnnealingLR
4. batch size: 128

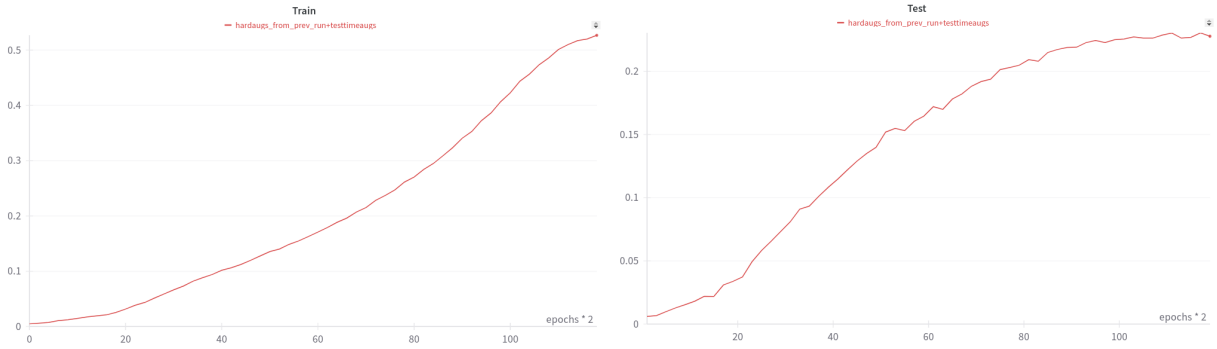


Анализ:

1. Качество стало похуже.

2. Появилась просто гиганская яма после 50 эпохи.

Попробовал добавить аугментации на моменте теста, но качество только ухудшилось.



§2.2 ReceptiveResNet50

Поскольку все изменения, которые я пробовал выше не давали хорошего прироста к качеству на тесте, я решил пойти почитать статьи. Гуглил решение задачи классификации на датасете Tiny ImageNet, потому что по своей структуре он очень похож на датасет, который использовался у нас в соревновании. Подцепил из одной из них следующую идею: для маленьких картинок важно увеличивать рецептивное поле модели. У исходного ResNet50 рецептивное поле одного пикселя, то есть то насколько соседних пикселей он смотрит, равнялось 46. А у MyResNet50 всего 39, так как я убрал макспулинг. Несмотря на то, что у нас картинки всего 40*40, и в принципе рецептивного поля вроде хватает, мне показалось полезным все таки увеличить его.

Для увеличения рецептивного поля я добавил слой макспулинга после 3 блока. Он уменьшает картинку в 2 раза, поэтому я решил убрать страйды из каждого первого слоя каждого из блоков, которые тоже уменьшали картинку. Теперь рецептивное поле для каждого пикселя получилось равно 72.

Для объективности экспериментов я на всякий случай продолжал их запускать параллельно с 2-мя разными наборами аугментаций. Я их называю v1augs и v2augs. Первый набор я накидал на рандоме, а второй попытался выстроить на основе выработанной интуиции. Тем не менее почти всегда у меня получалось примерно одинаковое качество используя v1augs и v2augs, поэтому, из-за отсутствия интуиции и удовлетворения этими наборами, других я придумывать не стал. Слева код для v1augs, справа для v2augs.

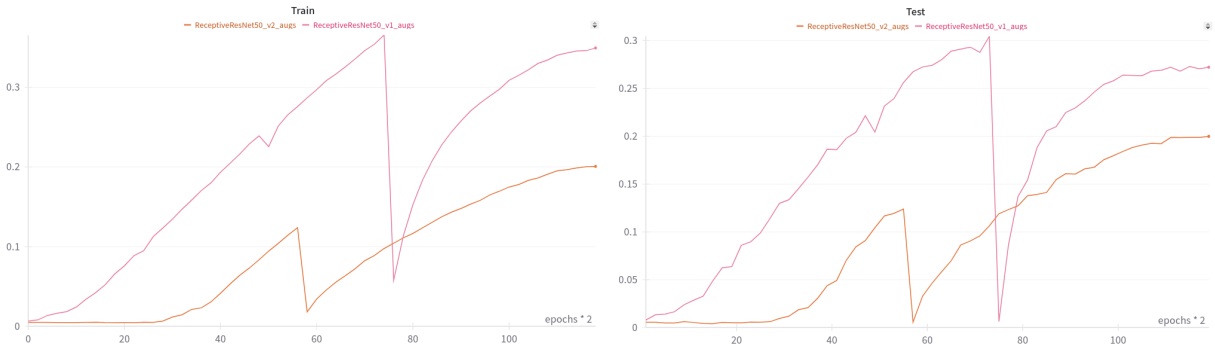
```
train_transform = v2.Compose(
        [v2.RandomHorizontalFlip(),
        v2.RandomVerticalFlip(),
        v2.GaussianBlur(3),
        v2.RandomRotation((-15, 15)),
        v2.ToTensor(),
        v2.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])
val_test_transform = v2.Compose(
        [v2.ToTensor(),
        v2.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])
```

```
train_transform = v2.Compose(
        [v2.RandomHorizontalFlip(p=0.3),
        v2.RandomVerticalFlip(p=0.3),
        v2.RandomApply(torch.nn.ModuleList([v2.GaussianBlur(3),]), p=0.2),
        v2.RandomApply(torch.nn.ModuleList([v2.RandomRotation((-15, 15)),]), p=0.2),
        v2.RandomApply(torch.nn.ModuleList([v2.ColorJitter(brightness=[0, 1], contrast=[0, 0.5]),]), p=0.2),
        v2.ToTensor(),
        v2.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])
val_test_transform = v2.Compose(
        [v2.ToTensor(),
        v2.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])
```

Теперь посмотрим на результаты обучения:
Описание запуска:

1. augmentations: v1augs - розовый, v2augs - оранжевый

2. lr: 0.5
3. scheduler: CosineAnnealingLR
4. batch size: 128



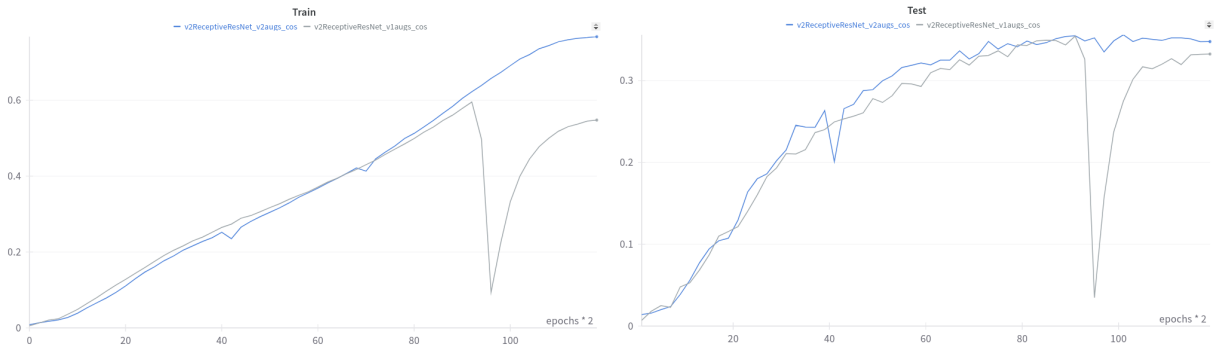
Анализ:

1. Никуда не исчезают странные гиганские ямы.
2. В моменте качесвто модели доходило до 30 процентов. Это классный прорыв.

§2.3 v2ReceptiveResNet50

Увеличение рецептивного поля давало плоды, поэтому я решил увеличить его еще, добавив еще один макспулинг, на этот раз после второго блока.
Описание запуска:

1. augmentations: v1augs - серый, v2augs - голубой.
2. lr: 0.5
3. scheduler: CosineAnnealingLR
4. batch size: 128

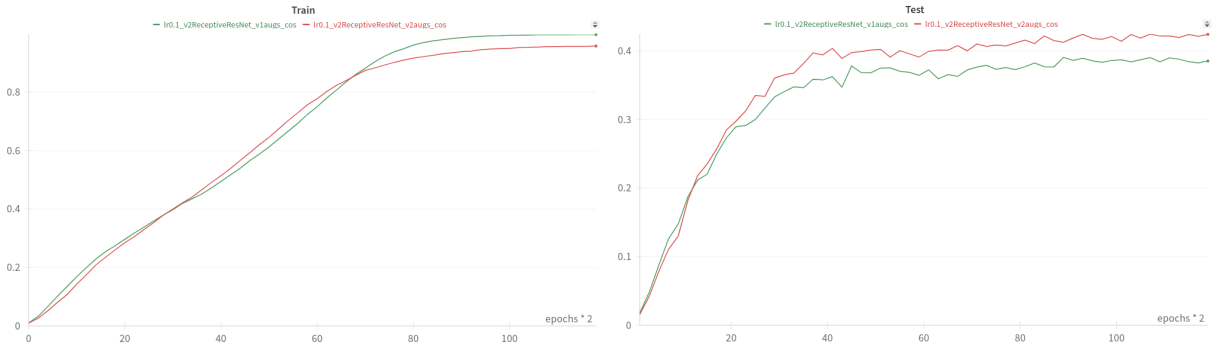


Анализ:

1. Качество ощутимо улучшилось и теперь достигло 36 процентов на валидации.
2. На запуске с v1augs опять рпоизошло что-то странное.

Происходила, какая-то ересь, написал семеру, потому что у самого совсем не было идей из-за чего подобные спецэффекты провала могут наблюдаться. Первой гипотезой было, что у меня стояло циклическое расписание, а не просто косинусоидное. Второй гипотезой были взрывы градиентов из-за большого lr. Перепроверил, что расписание не циклическое.А вот lr=0.5 и правда меня подводил. Попробовал уменьшить:
Описание запуска:

1. augmentations: v1augs - зеленый, v2augs - красный
2. lr: 0.1
3. scheduler: CosineAnnealingLR
4. batch size: 128

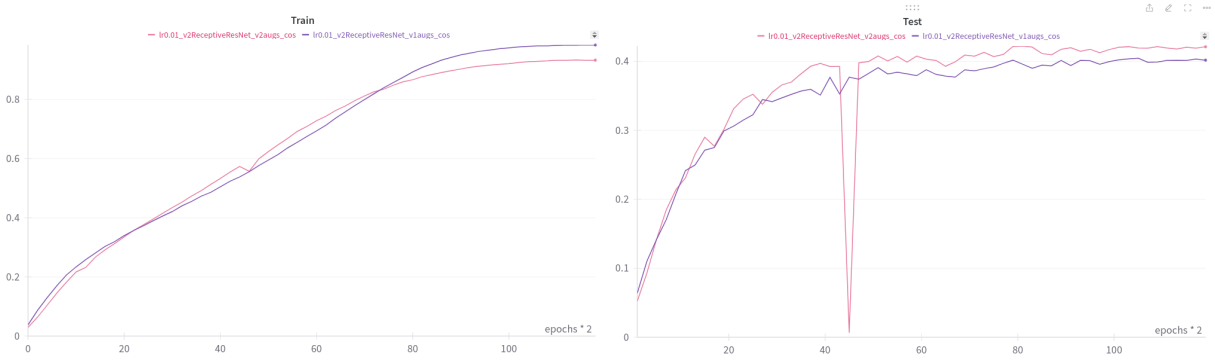


Анализ:

1. Непонятные спецэффекты пропали.
2. Удалось пробить качество в 40 процентов на валидации!

Попробовал еще уменьшить lr.
Описание запуска:

1. augmentations: v1augs - фиолетовый, v2augs - розовый
2. lr: 0.01
3. scheduler: CosineAnnealingLR
4. batch size: 128

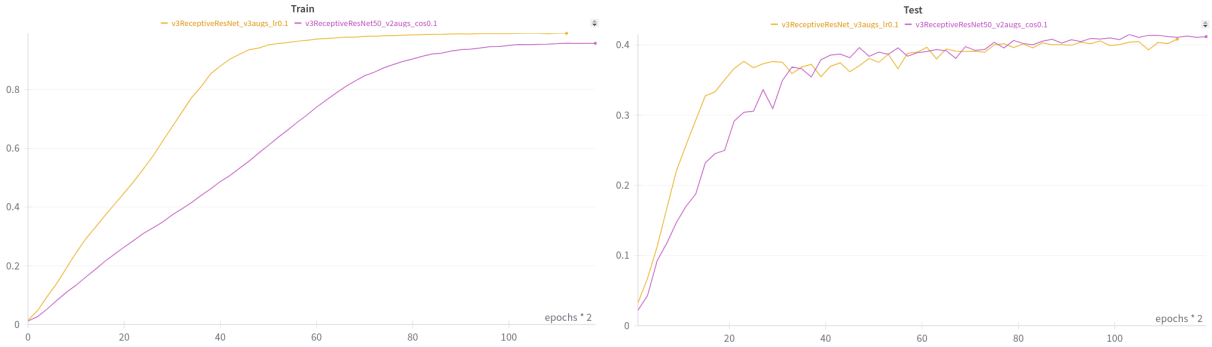


Анализ:

1. Качество получилось столь же хорошее, что и при предыдущем запуске.
2. Опять появился непонятный мне до сих пор провал.

§2.4 v3ReceptiveResNet50

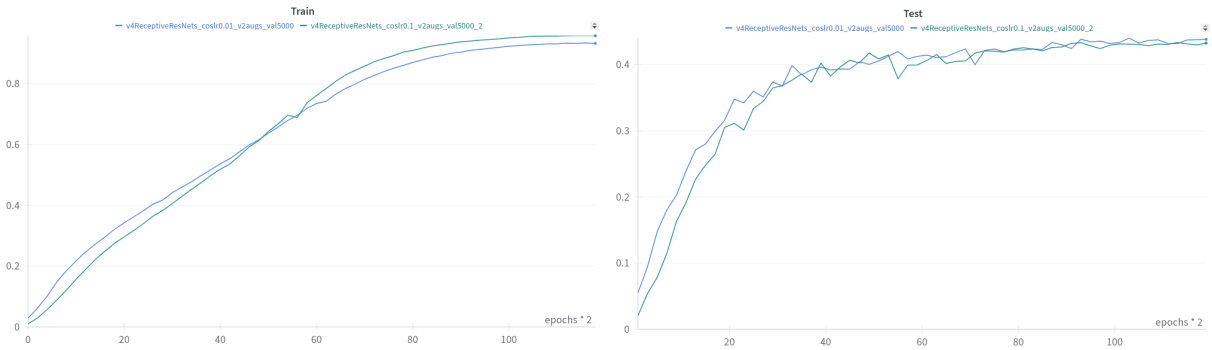
Тут попробовал изменить одну из сверток внутри каждого BottleNeck блока с 1*1 на 3*3. Но результат от этого не изменился, поэтому это изменение я удалил.



§2.5 v4ReceptiveResNet50

Тут я вспомнил, что первым же слоем в моей модели стоит свертка 7*7 со страйдом 2, а значит она уменьшает картинку в 2 раза прям сразу. А это плохо, поэтому сделал страйд равный 1 и смог выбить 42.4% accuracy на тесте благодаря этому изменению.
Описание запуска:

1. augmentations: v2augs
2. lr: 0.1 - зеленый, 0.01 - синий
3. scheduler: CosineAnnealingLR
4. batch size: 128



Такое вот получилось приключение. Суммарно было затрачено около 10 дней gpu времени, зато сколько всего интересного удалось узнать и попробовать!