Отчет о проделанной работе в большом домашнем задании по Глубинному Обучению:

Выполнил:

студент группы БПМИ211 Черномордин Родион Романович

Москва 2023

1 Предисловие

В моем отчете я буду идти вместе с тобой по моему блокноту. Он получился довольно большим, так как я потратил на все бдз кучу времени, а добился примерно ничего. Моя финальная модель будет всегда находиться в самом низу блокнота, точнее код ее запуска. К сожалению, я во время решения нажал на кнопку в кагле, чтобы ноутбук перезапустился, но это же удалило все графики в начале. Некоторые попытки я обрываю во время обучения, так как не вижу в них смысла, поэтому висит у некоторых красное полотно, чтобы сохранить их графики. Все решения я проводил на выборке, где размер тестовой части равен 0.5.

2 Решение

2.1 Baseline

Опустим момент, где я ищу все библиотеки и так далее. Код для блоков класса ImageDataset и для функций тренировки и валидации модели я взял с семинара по dl (код находится в открытом доступе в гитхаб). Это было до Нового Года, и я решил не изобретать велосипед и взял за baseline модель resnet18 - не слишком глубокая, хорошая сверточная нейронка. Важный момент - я ее не предобучал! Использовал SGD (так как это база), цикличный lr (так как в одном из дз получил, что он лучше всех работает, а в какой-то статье нашел этому давно логическое обоснование, то есть на подкорке лежало, что он классный), ну и в качестве функции потерь - кросс энтропию. В качестве аугментации я взял только рандомным разворот по горизонтали. Тут важный момент, я перерыл кучу интернет ресурсов про агументацию и нашел вот такое чудо: люди берут один набор данных с трансформером без аугментации и складывают с таким же набором, но с аугментации. Звучит логично, мы делаем больше данных, а шанс того, что к нам в батч (размером 64) из 10000 картинок попадут две одинаковые крайне мал. Но я не просто так взял и поставил это, я провел небольшой эксперимент, поперебирал комбинации шедулеров и аугментаций:

- SGD lr=0.1 m=0.9 ----> 0.19802
- SGD(lr=0.1 m=0.9) + CyclicLR(0.01, 0.1) ----> 0.20004
- SGD(lr=0.1 m=0.9) + LambdaLR ----> 0.1452
- SGD(lr=0.1 m=0.9) + CosineAnnealingLR ----> 0.1994
- SGD(lr=0.1 m=0.9) + CyclicLR(0.01, 0.1) + RandomHorizFlip ----> 0.23708
- SGD(lr=0.1 m=0.9) + CyclicLR(0.01, 0.1) + RandomHorizFlip + RandomGray ----> 0.21022
- SGD(lr=0.1 m=0.9) + CyclicLR(0.01, 0.1) + C(RandomHorizFlip, RandomGray) ----> 0.2136
- SGD(lr=0.1 m=0.9) + CyclicLR(0.01, 0.1) + RandomHorizFlip + RandomPerspect ----> 0.22146

2.2 Первый вариант

Эту часть я быстро перескажу, так как большим успехом она не пользовалась. Я за основу взял часть resnet. Но у resnet была очень большая проблема - быстро переобучался. С этим мог бы помочь dropout, поэтому я вспомнил, что такое есть в Alexnet, которая себя хорошо показала на маленьких датасетах. В классе SecondVrNN ее и реализовал. По седьмую версию включительно я игрался с ее составляющим: добавлял больше слоев, менял параметры, добавил skip connection. Все это приводило к улучшению результата.
Дальше пошли уже графики, ура!
Я решил добавить больше аугментаций, но я не экспериментировал, а просто взял все те, что мне понравились: RandomVerticalFlip, ElasticTransform, RandomGrayscale, RandomInvert, RandomEqualize, RandomHorizontalFlip. До девятой версии я игрался со всем, чем можно. Даже инициализировал веса по умному. Ничего не придавало сильного буста. Потом я продолжил играться со слоями до 13 версии, пробовал также поменять sgd на adam (но это было супер ошибкой), увеличить количество эпох, что тоже не приводило к какому-то успеху. В любом случае моя валидационная выборка выходила на "плато"и не могла от туда выбраться.

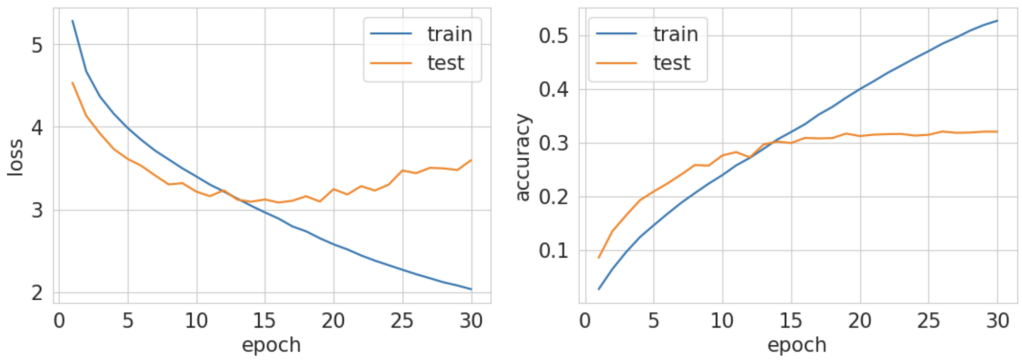

Рис. 2.1: Типичный график моих метрик для моделей

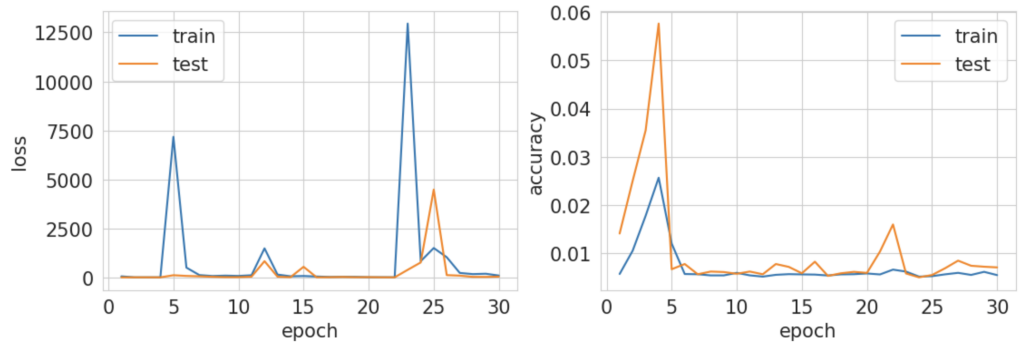

Рис. 2.2: Adam

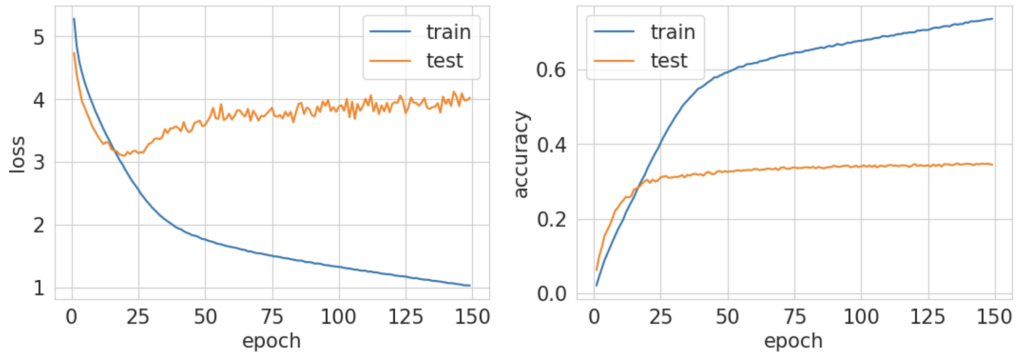

Рис. 2.3: Больше эпох

2.3 Второе дыхание

Я зашел в тупик, ничего у меня не работало, а куда развивать я не видел. Но вспомнил, что начинал я вообще с resnet, так может им и продолжить? Просто немного переписать код, чтобы он получился (структуру resnet я взял с их официальной гит странички со всеми библиотеками). Так и родилась моя финальная версия - 14-ая. Но просто резнет нет смысла использовать (помним про переобучение), что нам может помочь ее исправить - dropout. Поэтому я решил в блоки (класс RsBlock) в конце добавить Dropout, у которых в самом начале

вероятность стояла примерно 0.25. Размеры ядер у блоков я решил сохранить, а padding поставить, чтобы наши conv не меняли размер картинки (ну и так 40x40, куда еще меньше делать). В resnet существует 4 группы блоков: с выходом 64 каналов, с выходом 128 каналов, 256 и 512. Также посмотрим на начальный блок слоев: первым идет свертка, которая своими параметрами сжимает нашу картинку - заменим stride на 1, чтобы сохранилось изображение 40x40, а вот с размером ядра (и паддингом для него) я позже поиграюсь, но сейчас поставлю размер ядра 3, а padding=1. В конце идет maxpool - вещь, которая сожмет нашу картинку, тоже выкидываем (я помню, что проводил опыт, что будет если оставить, но код я потерял, но с ним хуже очев). Моя мотивация заключалась в том, что у нас и так супер мелкие изображения, поэтому нужно убирать все, что влияет на размер картинки (не считая последнего avgpool), так как сам resnet заточен на картинки 224x224 (как я помню), где уместна свертка с уменьшением размера.

Дальше я решил поэкспериментировать все же с scheduler. Я выбирал между двумя: Cos и еще одним, чье название забыл, но его смысл в том, что он помогает бороться с обучением на "плато так как замораживает lr, когда на него выходит (насколько я помню). Первым решил попробовать ко-синусный, поставив начальный высокий lr = 0.5 (логика была в том, что пусть он прыгнет далеко, а потом уже будем сходиться, чем мелкими шажками куда-то идти). И результат обрадовал - с заменой на косинус мой acc на val выборке вырос на 0.05 примерно

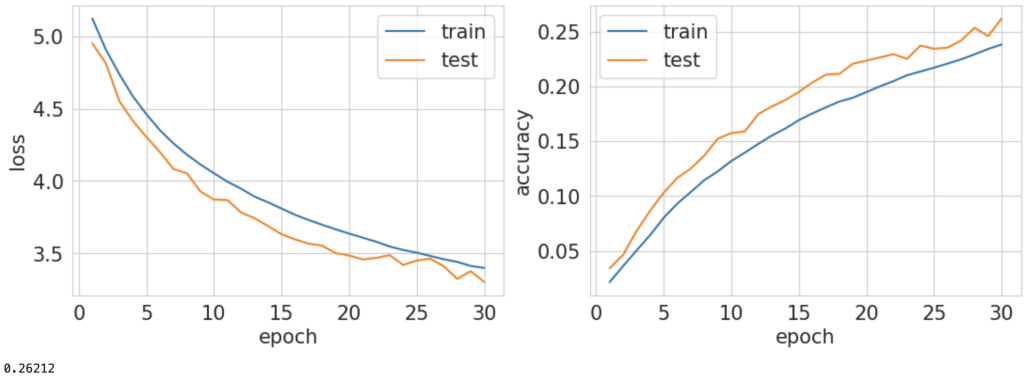

Рис. 2.4: CyclicLR

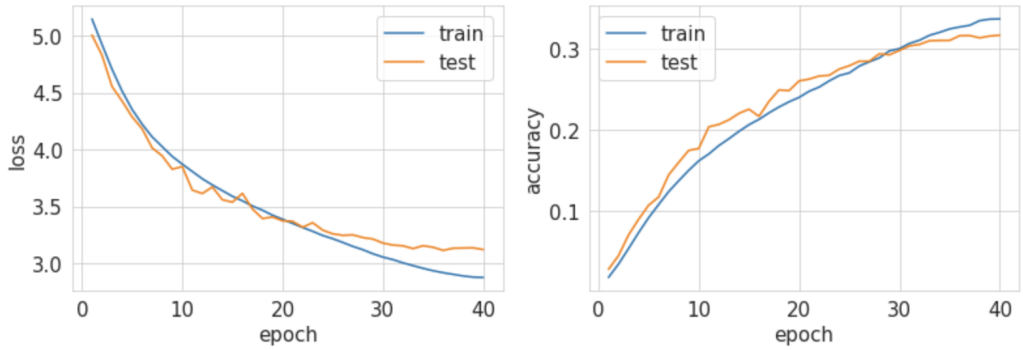

Рис. 2.5: CosineAnnealingLR

Все равно есть чувство, что я делаю со своим гениальным разбиением данных какую-то неправильную вещь. Поэтому сделаю по стандартному - train dataset состоит из одного набора данных с тарнсформером на аугментации (до этого, напомню, было два: с трансформером и без). Результаты стали хуже относительно тех, что были до этого:

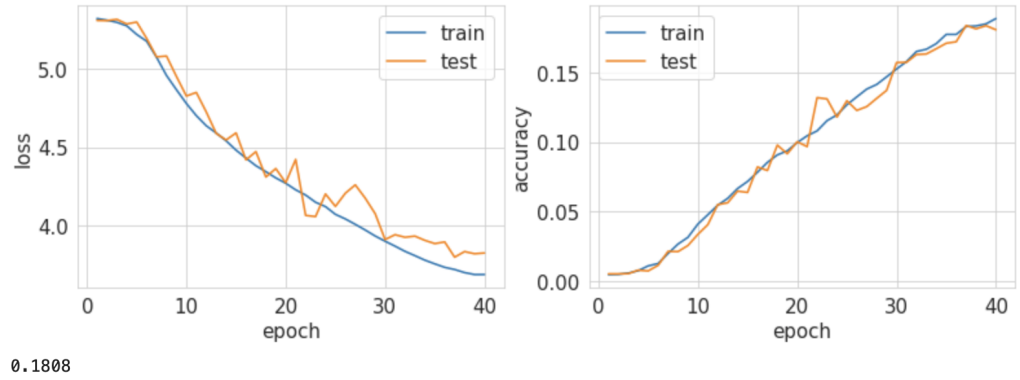

Рис. 2.6: Новое решение с 3 блоками

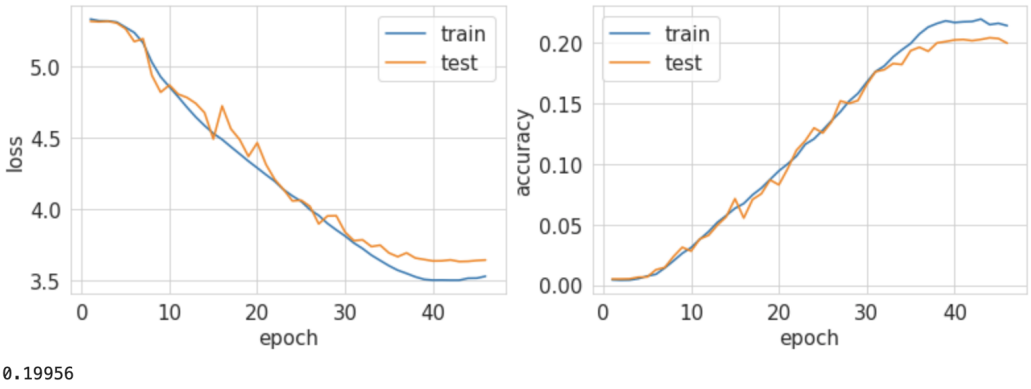

Рис. 2.7: Новое решение с 4 блоками

Можно заметить, что при всех 4 группах блоков у меня результат немного повышается, но я считаю, что ресурсы, потраченные на это, не оправдывают такой небольшой прирост, поэтому оставлю 3 группы (с 2 группами оценка была оказалась хуже).

Итак, я сейчас использую 3 группы по два блока внутри них (в коде их количество не изменилось), в каждом блоке есть по одному Dropout с вероятностью 0.25. Очевидно, что карта может так лечь, что я всю картинку могу в них потерять, поэтому уменьшу вероятность их до 0.1. Результат обрадовал:

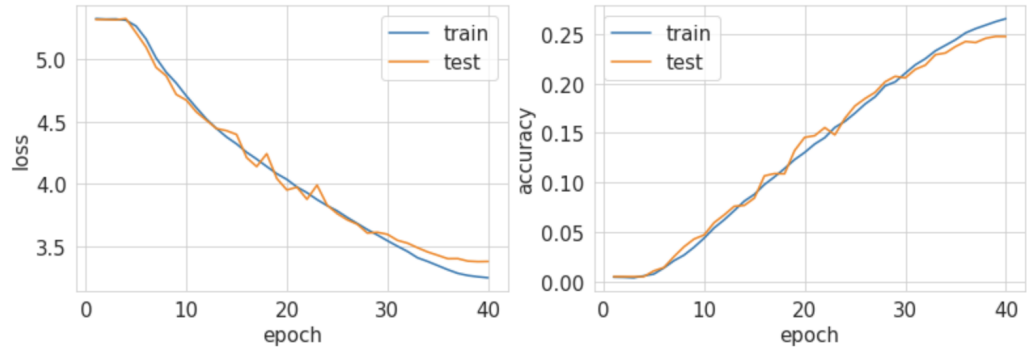

Рис. 2.8: Новое решение с 4 блоками

Сейчас я начинаю упираться в свой потолок, переобучение избежал по максимуму. Нужно решить проблему того, что моя модель плохо просто описывает задачу - нужно сделать ее слегка глубже (помним, что 4 группы по 2 блока ну слишком долго). Попробую немного изменить архитектуру - вынесу дропауты из блоков, поставлю их после групп блоков, увеличу конечный линейный слой (класс FifteenVrNN), здесь получилось просто все ужасно, поэтому брошу это идею:

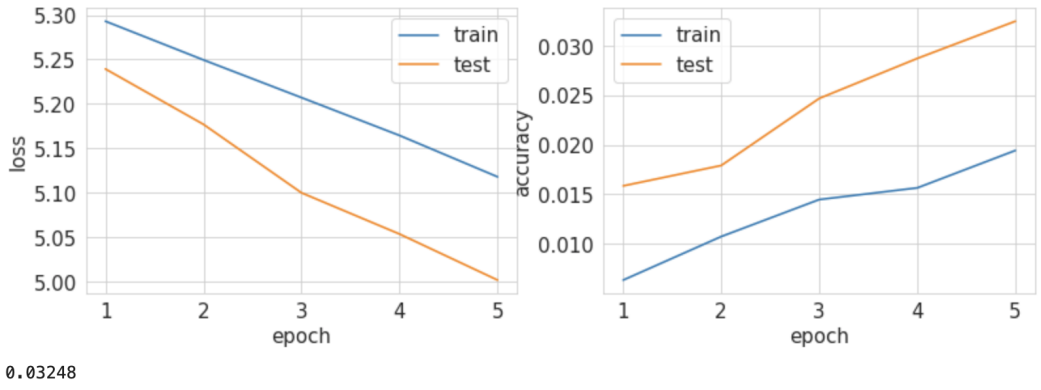

Рис. 2.9: 15 версия модели

Возвращаемся к 14 версии. Все еще цель слегка увеличить свою модель. Тогда мне пришло решение - увеличить размер ядра до 5 (paddning = 2) у самого первого линейного слоя. Это дало свои супер плоды, но линия валидации начала сильней отдаляться от train, то есть мы начали стремиться к переобучению (хоть результат и стал лучше).

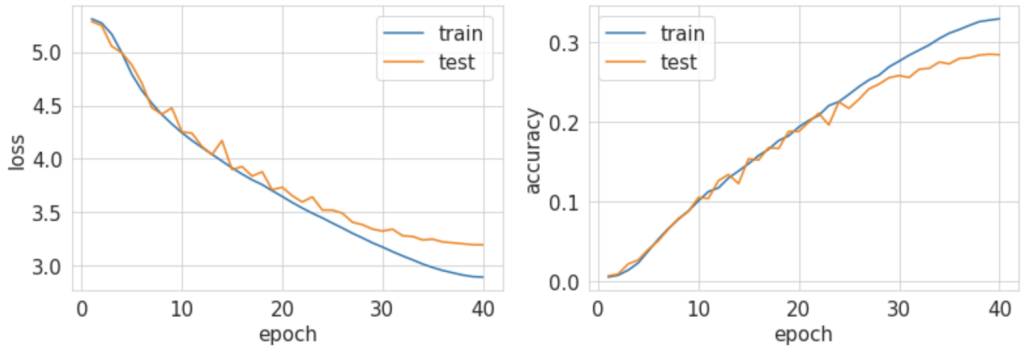

Рис. 2.10: Модель с увеличенным первым линейным слоем

Ну давайте попробую слегка поднять вероятность Dropout везде:

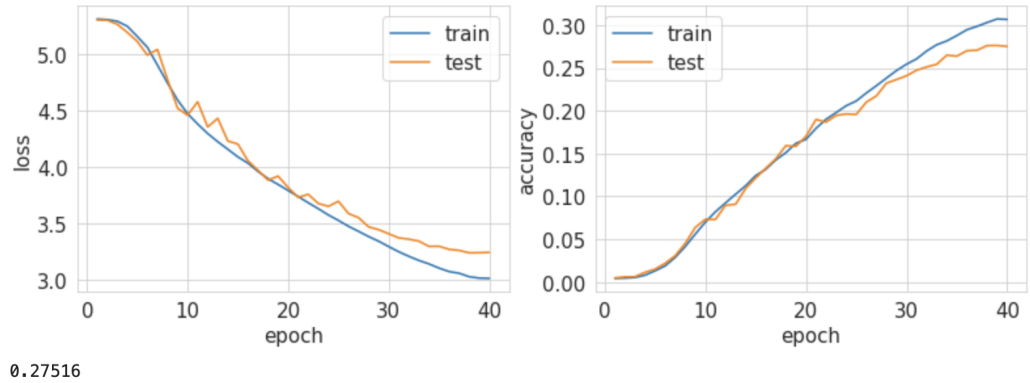

Рис. 2.11: 14 версия с поднятым p Dropout

Больше я не знаю куда можно увеличить модель и сохранить ее логику со здравым временем исполнения. Перебираю параметры, которые установил ранее. Сейчас мой начальным lr = 0.5, что не похоже на стандартные значения (0.001 или 0.0001), давайте тогда посмотрим, что получится с ними. Установим начальный lr = 0.0001. Результат убил:

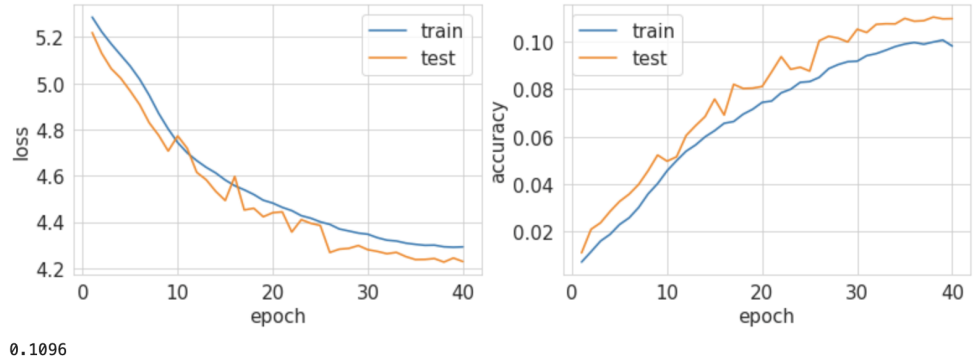

Рис. 2.12: lr = 0.0001

Посмотрим, что получится, если увеличим размер эпох до 100. Пик модель достигла на 40 эпохе (больше не будем это трогать):

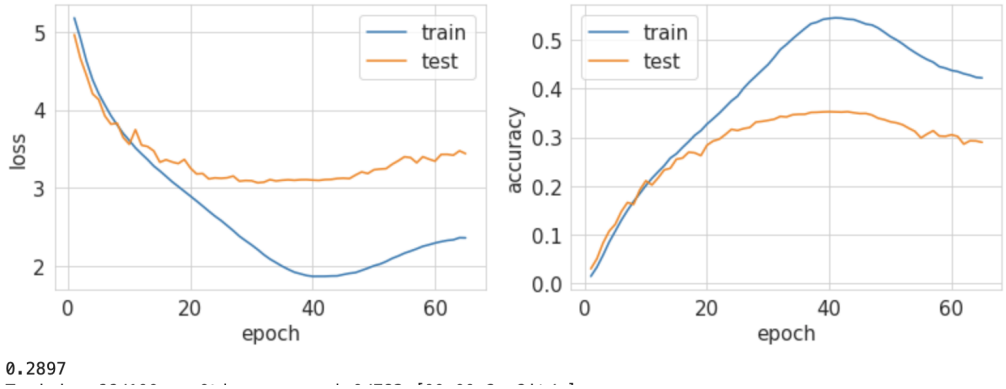

Рис. 2.13: 100 epochs

Помним, что над аугментацией я долго не сидел, поэтому стоит сейчас что-то с ней сделать. Я сомневаюсь (внутренне чувство), что изменение цветовое как-то помогает моей модели, скорее уместны flip и прочие прокрутки. Поэтому я решил исследовать новый набор аугментаций (опять на шару стреляю, так как времени уже совсем не осталось, а одна модель 3 часа обучается пимерно): RandomVerticalFlip, RandomRotation, RandomResizedCrop, RandomHorizontalFlip. Но результаты только стали хуже:

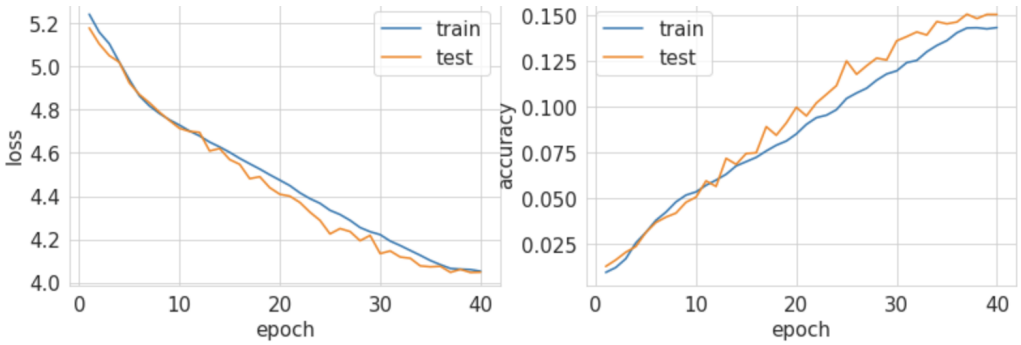

Рис. 2.14: Новый набор преобразований

Возвращаемся к старому набору аугментаций и начинаем их перебирать жадным алгоритмом. Без RandomInvert я смог очень сильно улучшить свои показания, без ElasticTransform и RandomEqualize результат тоже стал лучше. А вот без RandomGrayscale, RandomHorizontalFlip и RandomVerticalFlip резлультат наоборот стал хуже (тут прошу мне поверить, не сохранил блок с их кодом):

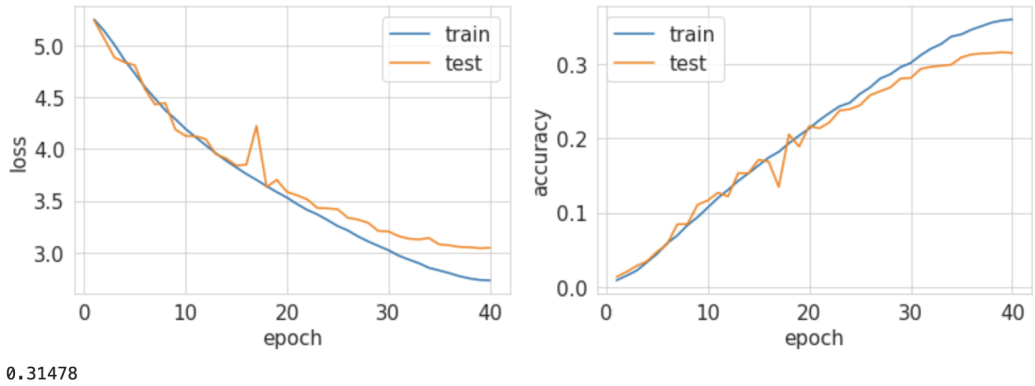

Рис. 2.15: Убрал RandomInvert

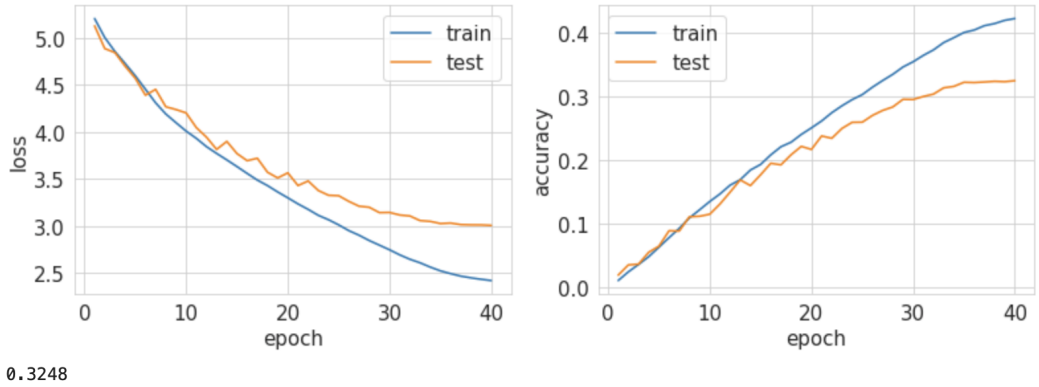

Рис. 2.16: Убрал RandomEqualize

Видим, что переобучение стало еще сильней выделяться, но модель стала лучше описывать нашу задачу. А самый верный справиться с любым переобучением (как вариант)

- l2 рег. Добавим его со значением weight_decay = 0.0001, и это окажется лучшим моим вариантом, что я добивался.

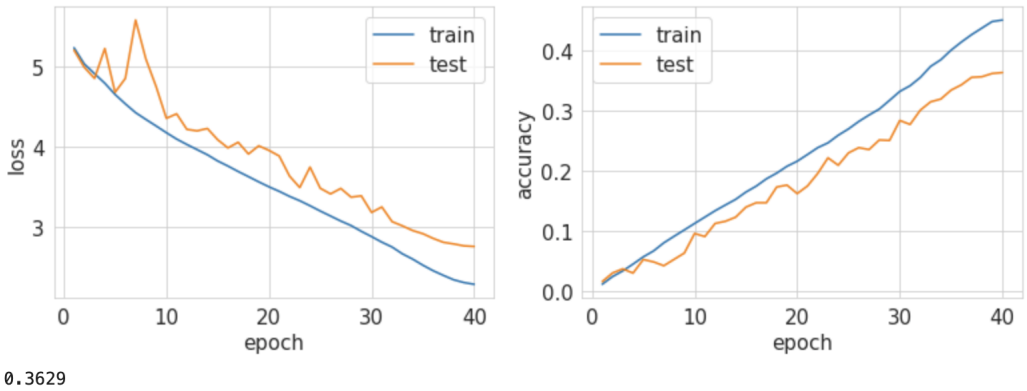

Рис. 2.17: l2

2.4 Финиш

Посмотрим на итоговый вариант, результаты которого я и заслал. Я сделал все то же самое, только сделал test_size = 0.1. И у меня получилась страшная картинка, где явно видно, что модель переобучилась. Я немного поломал голову и поймал гениальную мысль: я использую 3 аугментации, вероятности которых равны 0.5, 0.5 и 0.1. Получается, мат ожидание доли, которая не будет задета аугментацией (а она меня интересует довольно сильно) равна всего лишь 0.225! То есть я из оригинальных данных могу использовать лишь малую часть, поэтому модель зацикливается на аугментациях, что и дает мне ужасный результат. Логичным решением было бы уменьшить вероятности, но тут я понял, почему некоторые использовали метод, при котором делали одну часть без аугментаций, а другую с ними. Я так и сделал, что дало мне восхитительный результат и 0.43 на public test! Тут же по сути у меня мат ожидание части, которая осталась "чистой"равно 0.6125. Что довольно приятно. На этом и закончилась моя история с этим бдз!

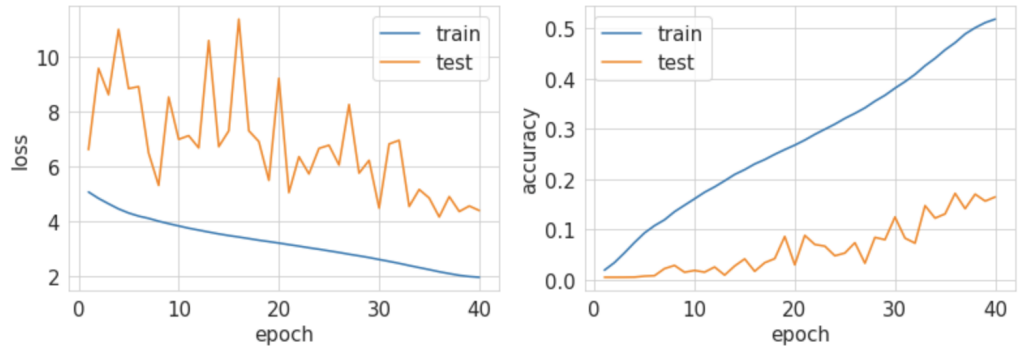

Рис. 2.18: Первый вариант

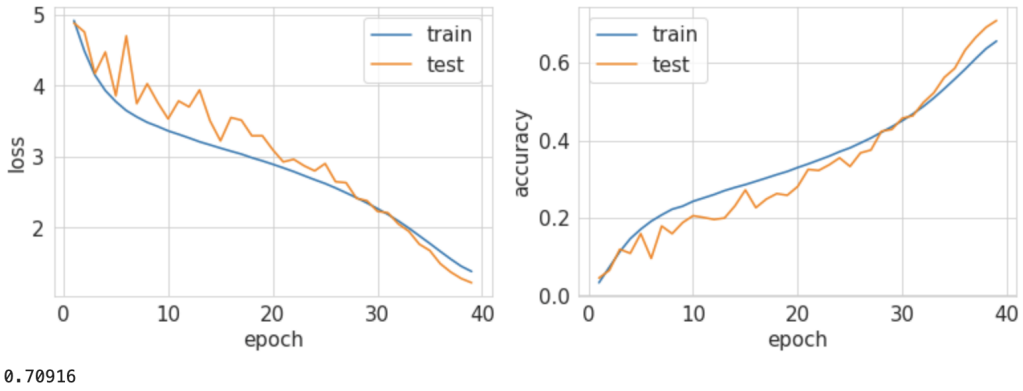

Рис. 2.19: Итоговый результат

3 Вывод

Сейчас на часах 23:46 21 января, я потратил все бесплатные ресурсы кагла и коллаба, пару тысяч для датасферы, так как мой железный монстр не тянул такие вычисления на своем гпу. Я преисполнился, потрогал все на практике, понял лучше принцип действия. Изменил бы я свой подход к этому дз, если бы вернулся в прошлое? Однозначно да, так как видно, сколько я неверных шагов допускал. Немного обидно, конечно, что так долго пыхтел и выбил на 5 оценку скор, но что поделаешь. Если после прочтения останутся вопросы к моему решение - смело пиши, пожалуйста, буду стараться пояснять. Код старался писать структурировано и чисто. Спасибо, пока!