Отчет по БДЗ №1

Федоров Никита

Декабрь 2022

1 Введение

Ссылка на wandb со всеми экспами. Каждый run - это отдельный эксп. В конфиге у run можно найти поле script (или scripts) - там сохранен весь код экспа, запустившего данный run. Таким образом, можно легко в точности воспроизвести любой эксп, просто скопировав и запустив его код (не сохранился код только для нескольких самых первых экспов, но в них все равно ничего интересного нет).

Деление trainval на обучающую и валидационную выборку сначала происходило в отношении 80:20, а потом я изменил его на 90:10.

2 Checkpoint

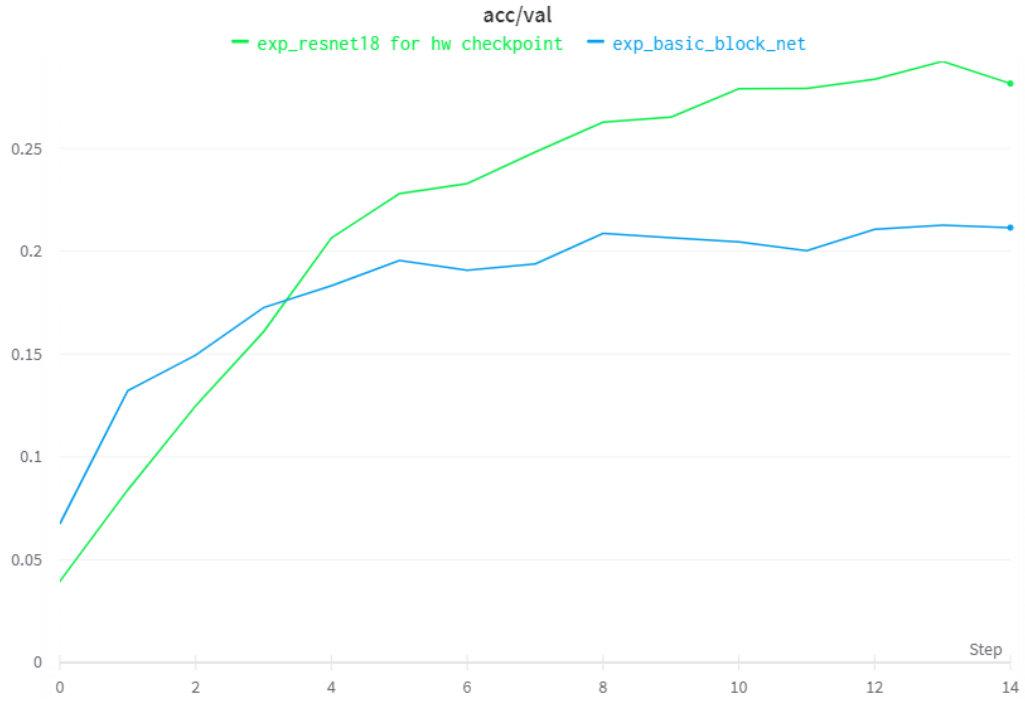
Для чекпоинта я запустил два экспа.

В первом я обучил один BasicBlock из второй маленькой домашки.

Во втором я обучил resnet18, запихивая в него картинки 64×64 и преобразуя его 1000 выходов линейным слоем в 200 выходов для классификации. Это было очень плохой идеей, потому что, во-первых, у моделек из torchvision.models в конструкторе есть параметр num_classes, который можно было приравнять к 200, а во-вторых, на вход им нужно подавать картинки с пространственными размерностями хотя бы 224. К сожалению, последнее я понял только ближе к дедлайну, когда у нас уже отняли датасферу (: Поэтому большинство экспов я проводил с самодельными моделькаами, а качество выше 60% на валидации удалось добиться только в самом конце, когда я стал ресайзить картинки к 224×224 перед тем, как запихивать их в нейронку.

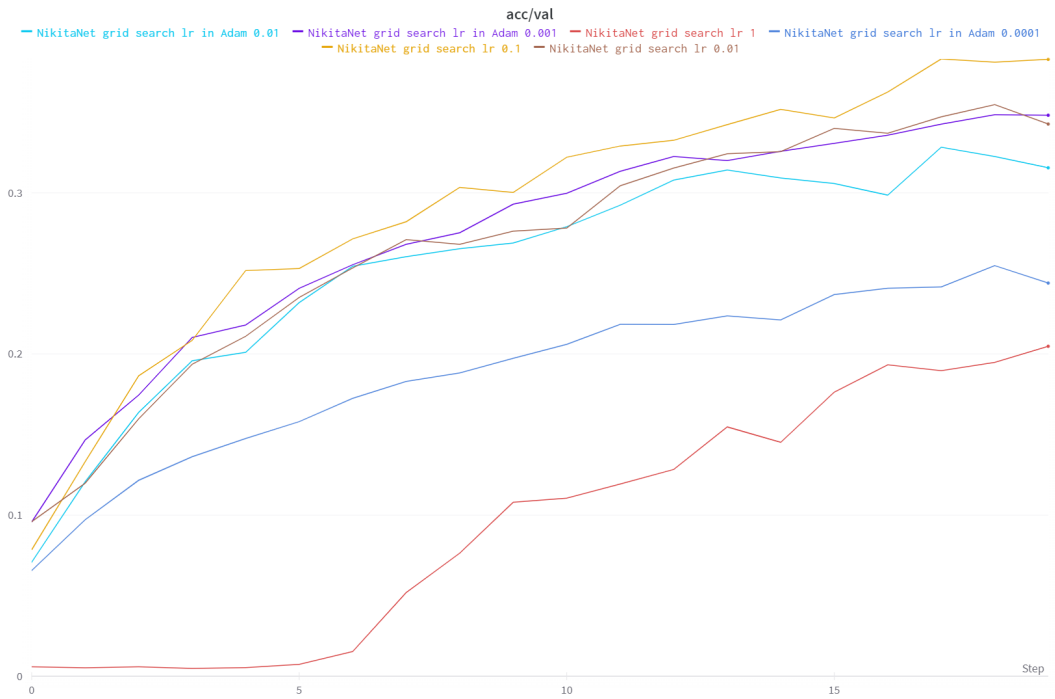
В обоих экспах использовался SGD с lr=0.1 без weight_decay и без scheduler'a. В качестве аугментации использовался только RandomHorizontalFlip.

Здесь и далее на графиках по оси x отложен номер эпохи, по оси y - accuracy на валидации.



3 Собственная архитектура

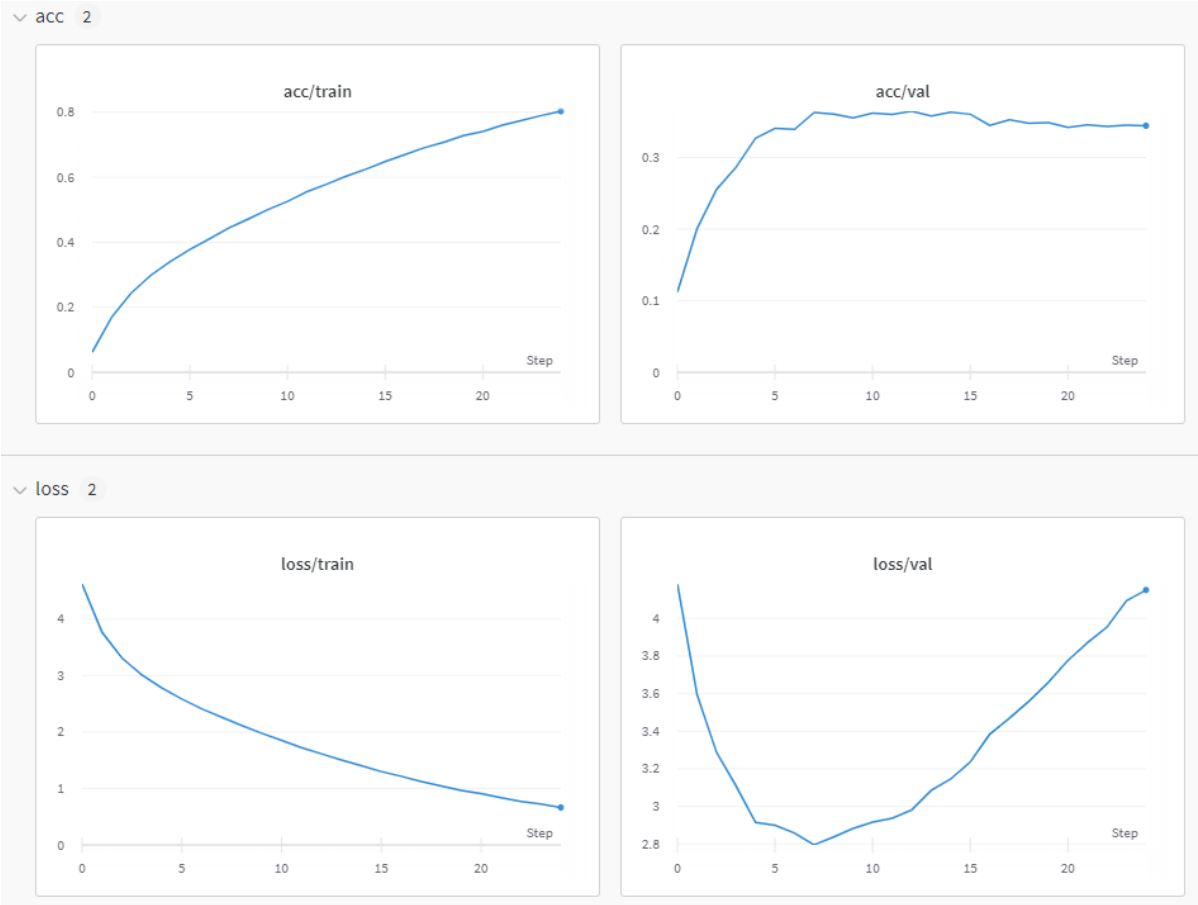
Далее я собрал нейросеть из нескольких BasicBlock'ов, идущих подряд, постепенно увеличивая при этом число каналов. В конце делался GlobalAvgPooling по простанственным размерностям и это подавалось в линейный слой для классификации. Эту нейросеть я пытался учить на SGD и Adam с разными значениями learning rate (на графике в легенде, если не указано, что это Adam, то это SGD)



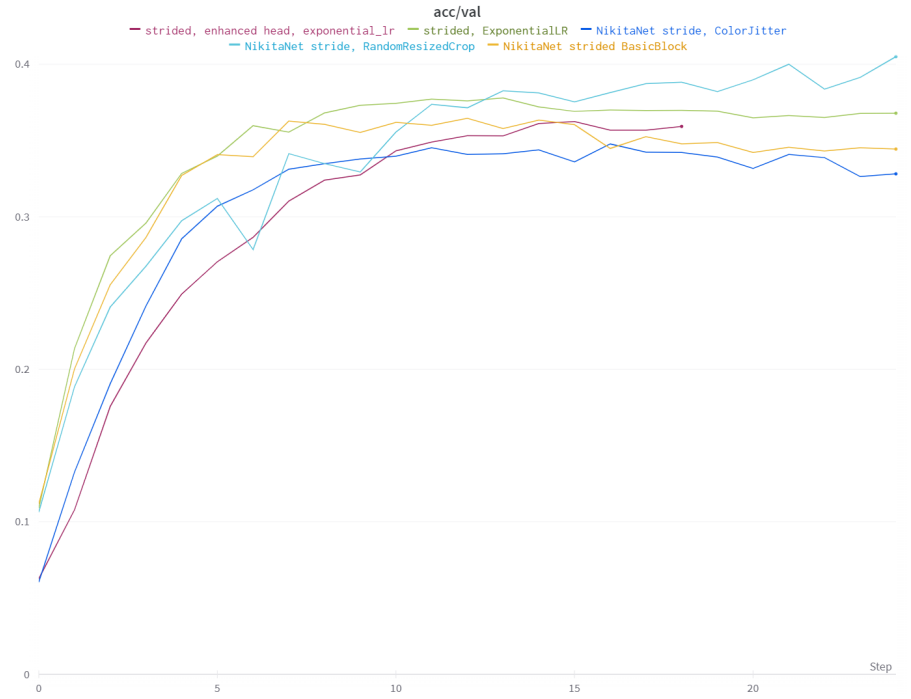
После этого я учил все на SGD с lr=0.1

3.1 Добавление stride=2 в BasicBlock

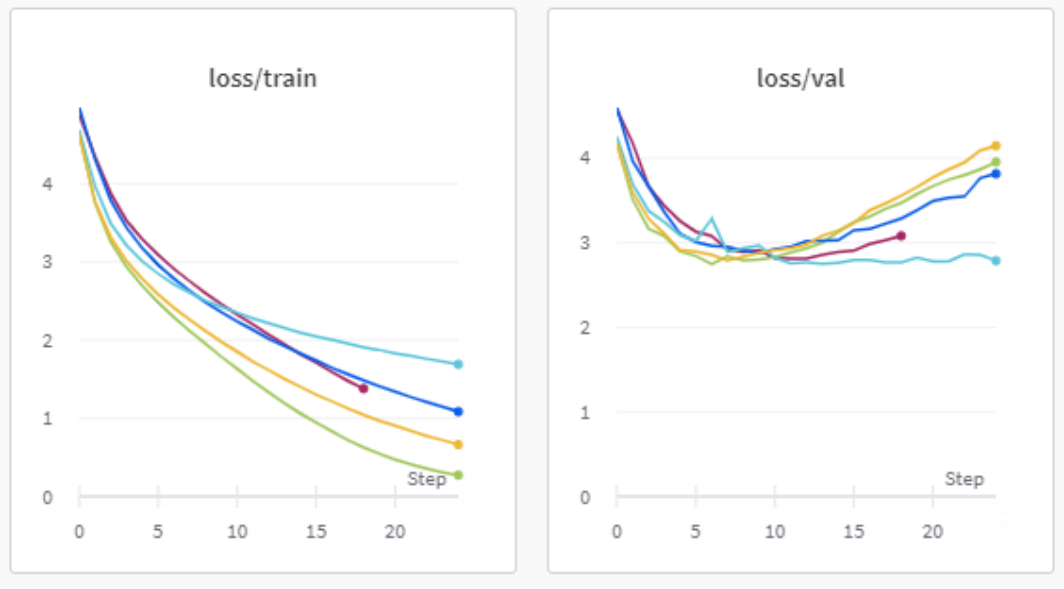
Теперь я добавил в первую свертку каждого BasicBlock'а параметр stride=2. Эксп. Оно дико переобучилось:



Тогда я стал пробовать добавлять аугментации RandomResizedCrop, ColorJitter. Пробовал добавить ExponentialLR scheduler, составить классифицирующую голову из нескольких FC слоев с relu и bn между ними (вместо одного слоя).



Но почти все, кроме ResizedCrop сильно переобучалось:



Отсюда я сделал выводы:

- классифицирующую голову не имеет особого смысла делать более сложную, чем один линейный слой

- можно всегда использовать ExponentialLR с γ порядка 0.95, это скорее всего немного улучшит качество

- нужно больше аугментаций

И потом я добавил больше аугментаций...

3.2 Чередование BasicBlock'ов со stride'ом и без

Структура нейросети:

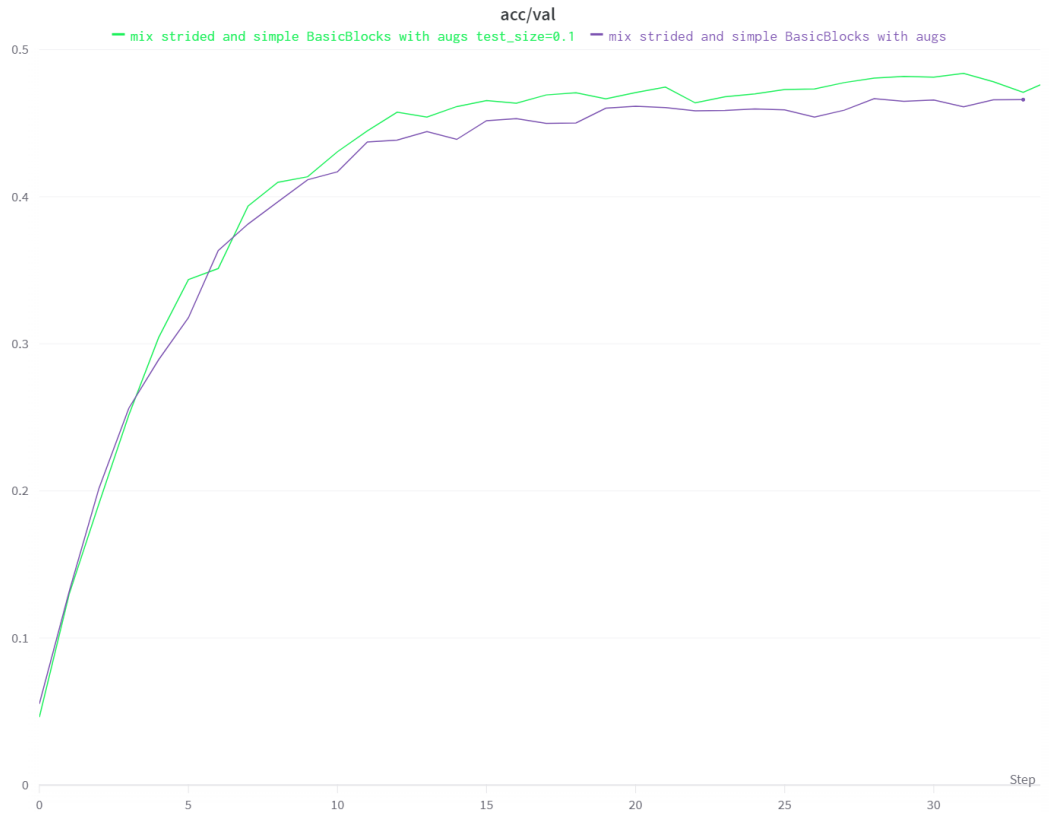
```
self.blocks = nn.Sequential(
    BasicBlock(3, 32),
    StridedBasicBlock(32, 64),
    BasicBlock(64, 64),
    StridedBasicBlock(64, 128),
    BasicBlock(128, 128),
    StridedBasicBlock(128, 254),
    BasicBlock(254, 254),
    StridedBasicBlock(254, 512),
    BasicBlock(512, 512),
)
self.avgpool = nn.AdaptiveAvgPool2d(output_size=1)
self.linear = nn.Linear(512, N_CLASSES)
```

Аргументы блоков - это число входных и выходных каналов соответственно.
Аугментации:

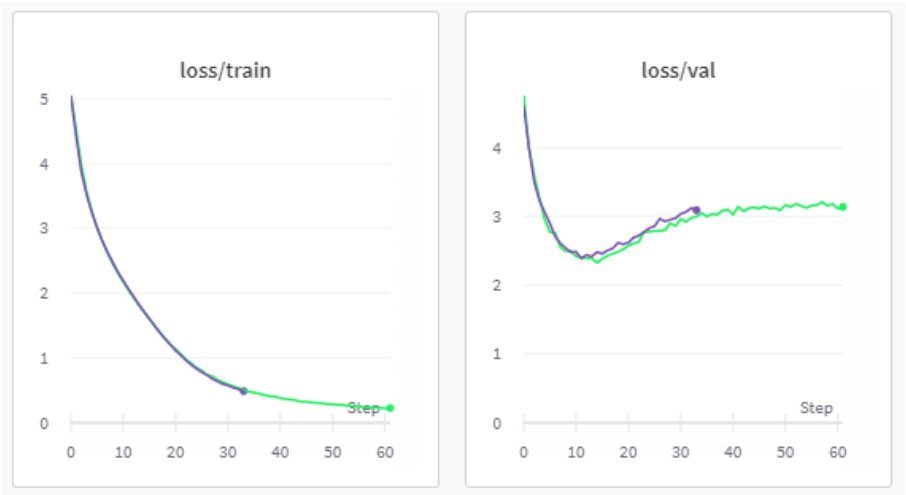
```
train_transform = T.Compose([
    T.RandomChoice([
        T.RandomResizedCrop(64, scale=(0.6, 1.0)),
        T.ColorJitter(brightness=.5, hue=.3),
```

```
            T.Grayscale(num_output_channels=3),
            T.GaussianBlur(kernel_size=3, sigma=(0.1, 1.0)),
            T.RandAugment(),
            T.AugMix()
        ]),
        T.RandomHorizontalFlip(),
        T.ToTensor(),
        T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
```

Ссылка на эксп. Также я попробовал запустить ту же архитектуру при train_size : val_size = 90:10 (до этого отношение было 80:20).



Благодаря архитектуре и аугментациям качество выросло, но переобучение все еще высокое:



И тогда я добавил еще больше аугментаций.....

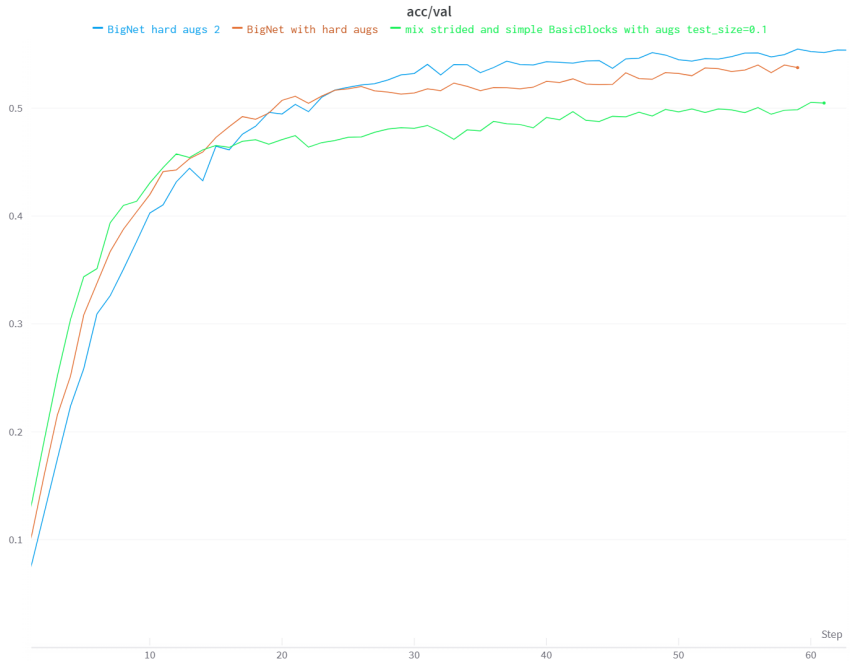
```
train_transform = T.Compose([
    T.RandomChoice([
        T.AutoAugment(),
        T.TrivialAugmentWide(),
        T.RandAugment(num_ops=4),
        T.Compose([
            T.RandomApply([T.RandomResizedCrop(64, scale=(0.6, 1.0))]),
            T.RandomApply([T.RandomChoice([
                T.ColorJitter(brightness=.5, hue=.3),
                T.Grayscale(num_output_channels=3),
            ])], p=0.7),
            T.RandomChoice([
                T.GaussianBlur(kernel_size=3, sigma=(0.1, 1.0)),
                T.AugMix(),
            ]),
        ])
    ]),
    T.RandomHorizontalFlip(),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

И еще больше аугментаций.....

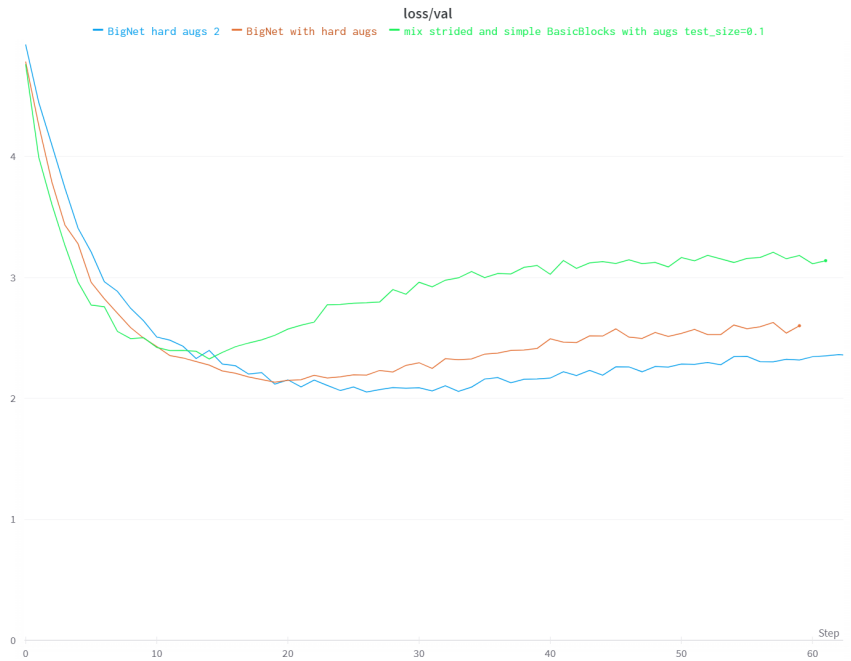
```
train_transform = T.Compose([
    T.RandomChoice([
        T.RandomChoice([
            T.AutoAugment(T.AutoAugmentPolicy.CIFAR10),
            T.AutoAugment(T.AutoAugmentPolicy.IMAGENET),
            T.AutoAugment(T.AutoAugmentPolicy.SVHN),
        ]),

        T.TrivialAugmentWide(),

        T.RandomChoice([
            T.RandAugment(num_ops=5),
            T.RandAugment(num_ops=6),
            T.RandAugment(num_ops=7),
        ]),

        T.Compose([
            T.RandomResizedCrop(64, scale=(0.6, 1.0)),
            T.RandomApply([T.RandomChoice([
                T.ColorJitter(brightness=.5, hue=.3),
                T.Grayscale(num_output_channels=3),
            ])], p=0.7),
            T.RandomChoice([
                T.GaussianBlur(kernel_size=3, sigma=(0.1, 1.0)),
                T.AugMix(),
```

```
            ]),
            T.RandomChoice([
                T.RandomPerspective(),
                T.RandomRotation(30)
            ]),
        ])
    ]),
    T.RandomHorizontalFlip(),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

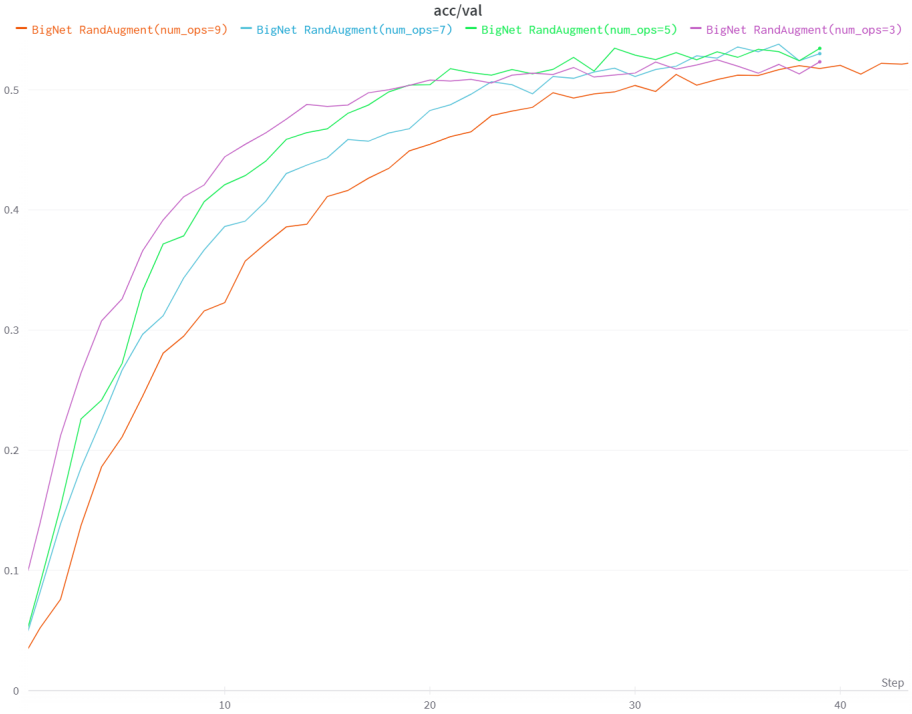
И это помогло



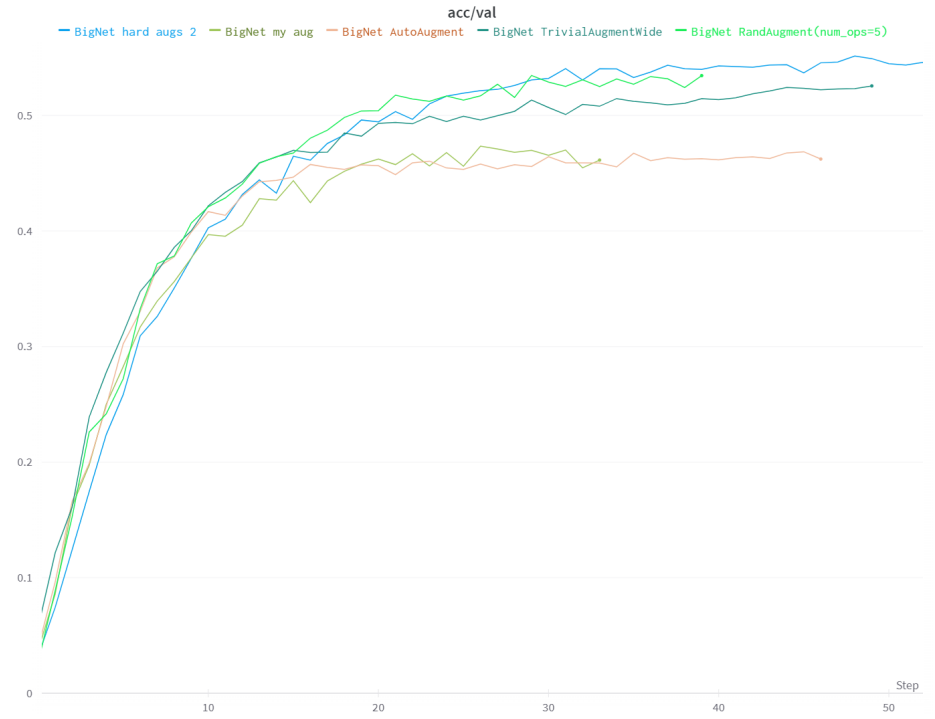
Удалось значительно снизить переобучение:



Также я пробовал использовать все эти аугментации по отдельности:



Лучше всех себя показало num_ops равное 5 и 7.



Отсюда можно сделать вывод, что, чем использовать один из этих вариантов аугментаций, лучше запихать их в RandomChoice и каждый раз выбирать одну из них.

3.3 Окончательная самодельная архитектура

Я также пытался менять архитектуру нейросети, добавляя в конце еще один StridedBasicBlock(512, 512)или StridedBasicBlock(512, 1024), убирая некоторые BasicBlock'и из архитектуры (without

last basic block, without 2 last basic blocks, strided blocks only), добавляя слои Dropout и Dropout2d в разные места (dropout2d, less dropout2d, dropout1d), добавляя weight_decay в оптимайзер. Также пробовал не делать GlobalAvgPool в конце, а делать Flatten карты признаков с пространственной размерностью 4×4 (flatten head). В конце концов лучше всех показала себя архитектура без последнего BasicBlock'а:

```
self.blocks = nn.Sequential(
    BasicBlock(3, 32),
    StridedBasicBlock(32, 64),
    BasicBlock(64, 64),
    StridedBasicBlock(64, 128),
    BasicBlock(128, 128),
    StridedBasicBlock(128, 254),
    BasicBlock(254, 254),
    StridedBasicBlock(254, 512)
)
self.avgpool = nn.AdaptiveAvgPool2d(output_size=1)
self.linear = nn.Linear(512, N_CLASSES)
```

Эксп: without last basic block

Таким образом, моя архитектура смогла добиться отметки в 59-60% на валидации (если учить 200+ эпох).

4 Resize к 224 на 224 + mobilenet_v2

Потом я спёр код Ильдуса с семинара. Но сначала я заменил его трансформации картинок:

```
train_transform = T.Compose([
    T.RandomResizedCrop(224, scale=(0.5, 1.0)),
    T.RandomHorizontalFlip(),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

test_transform = T.Compose([
    T.Resize(256),
    T.CenterCrop(224),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

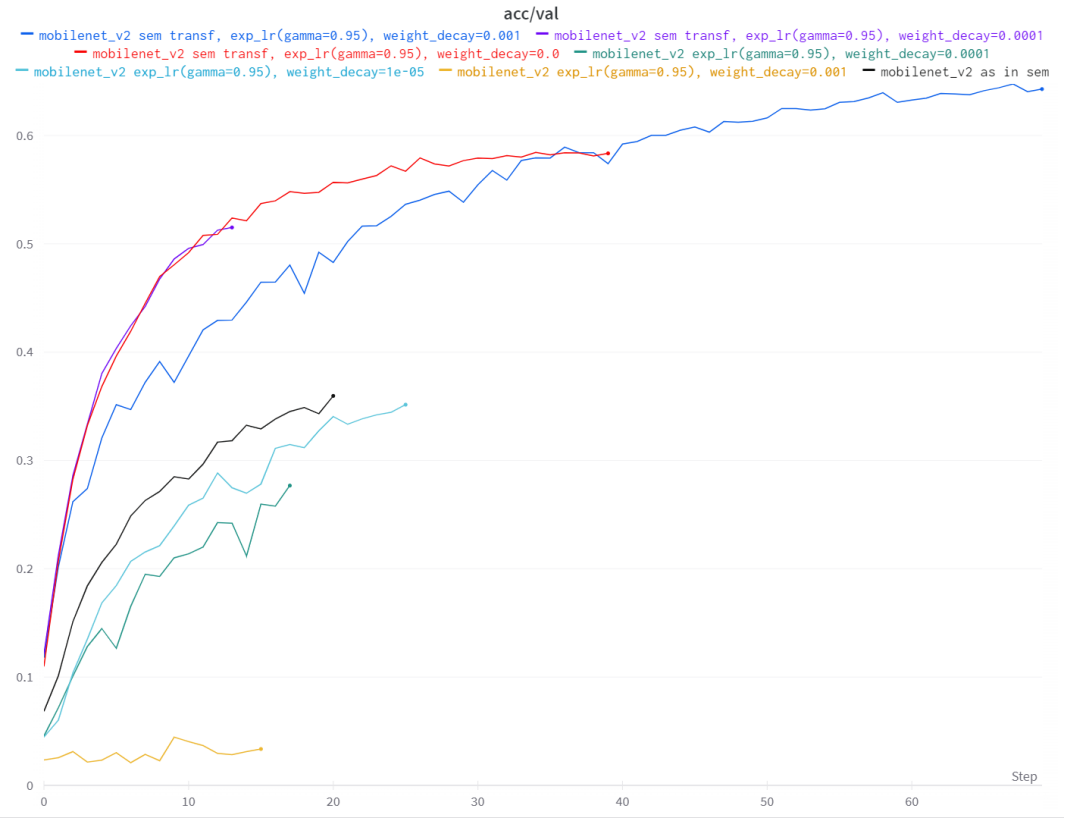
на свои:

```
train_transform = T.Compose([
    T.RandomResizedCrop(64, scale=(0.5, 1.0)),
    T.RandomHorizontalFlip(),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

test_transform = T.Compose([
```

```
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

поскольку мне показалось странным ресайзить картинки к размеру 224. Потом я наконец догадался, что это важно (об этом даже написано в доке torchvision), и использовал трансформации Ильдуса.

В итоге получилось (экспы, начинающиеся на "mobilenet_v2 sem transf"используют трансформации Ильдуса):



Лучший из этих экспов показал качество более 64% на валидации. Его я использовал в качестве финальной сдачи.

5 Заключение

Подавая в модель картинки размером 64×64, получить на моделях из torchvision.models качество выше 40-45% у меня не удалось (что теперь уже для меня неудивительно).

Для таких маленьких картинок мне удалось соорудить свою собственную архитектуру нейросети, которая выбивает качество 59-60% на валидации.

Если же подавать на вход моделям из torchvision.models картинки с пространственной размерностью не менее 224, то получается значительно превзойти данный уровень. Конечно, хотелось бы поставить побольше экспов, понимая это. Но у меня уже закончилась квота на кагле, поэтому оставляю решение в таком виде.