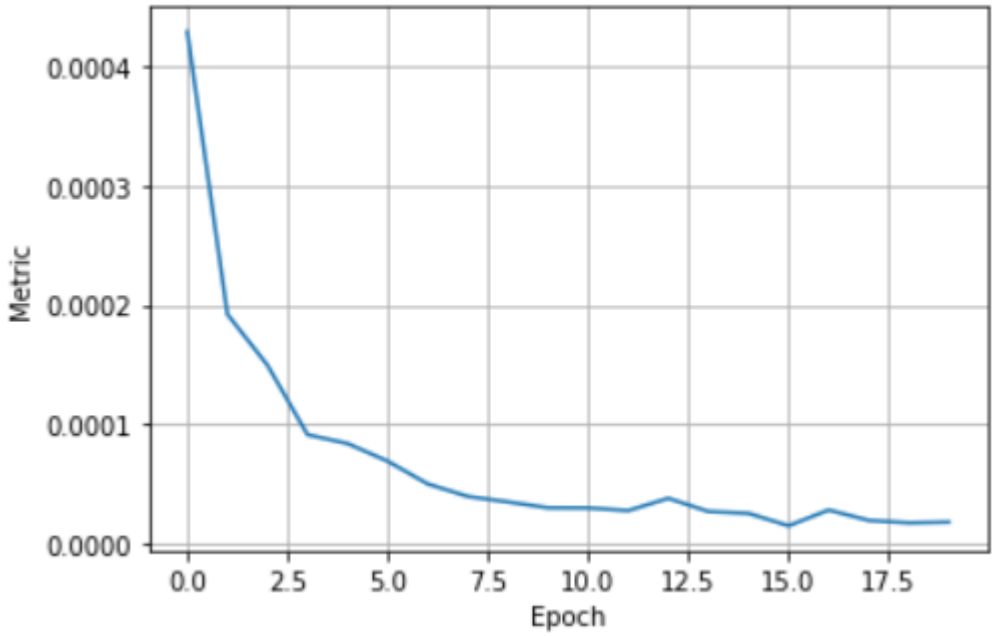
KWS. Отчет

В этом отчете я постараюсь рассказать все, что я попробовал, в хронологическом порядке. Самые важные графики я прикреплю, но подробнее можно ознакомиться в ноутбуке – я старался писать текст и там.

Бейзлайн

Бейзлайн был взят с семинара почти без изменений. Единственное – я зафиксировал везде seed для воспроизводимости (также я сохранил модель и train/val сплит на всякий случай). Модель обучилась лучше (1.5e-5) обозначенного порога (5e-5), в чате писали про большой разброс (видимо seed(3407) действительно is all you need). Также отмечу, что я взял лучшую с т.з. метрики на валидации модель в процессе обучения (а не последнюю), но разница между ними не существенна. (Кажется, ограничений на то, как обучать бейзлайн модель не было, главное получить 5e-5)



Streaming

Далее я принялся реализовывать streaming интерфейс для модели (т.е. пофреймовое предсказание). Для этого нужно завести буфер для нескольких величин: для предыдущих фреймов (в количестве receptive field свертки), для скрытого состояния GRU и для выходов GRU (в количестве

window_size, задаваемом пользователем) – для подсчета внимания, но об этом ниже.

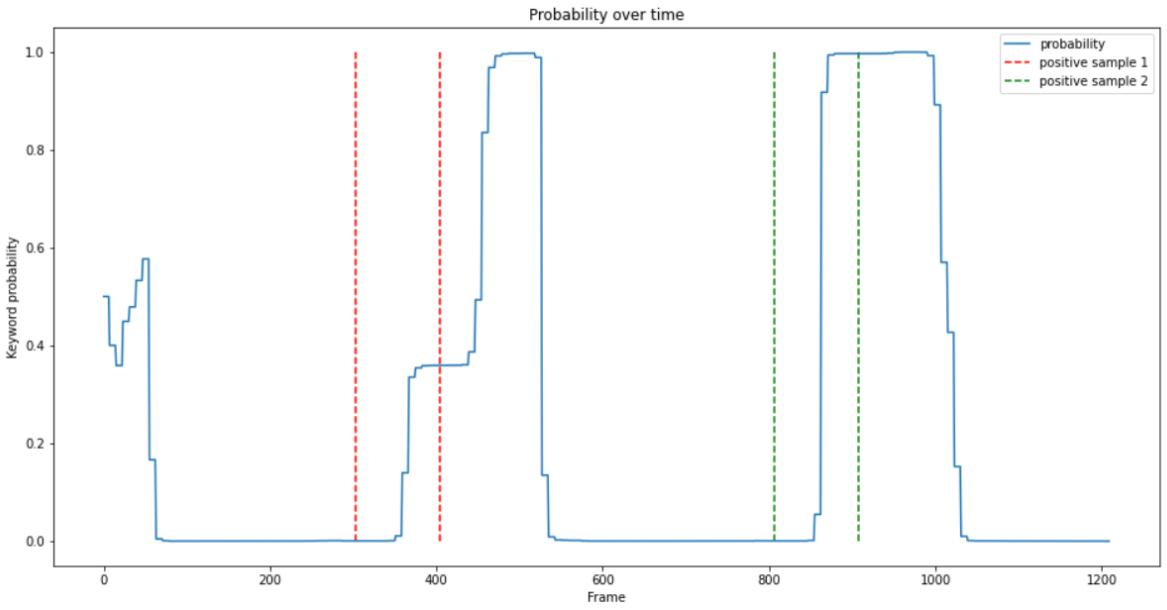
Предсказывать при каждом новом фрейме может быть слишком расточительно + сеть обучалась с большим страйдом, так что при обработке пофреймово (читай с единичным страйдом) можем получить смещенные данные (может пострадать GRU, которая, возможно, уловила с какой периодичностью забывать прошлое). Для гибкости я добавил параметр streaming_step_size, который делает так, чтобы все пересчитывалось с другой периодичностью (например, можно указать параметр равный страйду свертки, чтобы все было как в обучении).

Из-за того, что пересчет может делаться не каждый фрейм, то есть предсказание не будет меняться, я добавил кэш для предсказания.

Наконец, я добавил более умный пересчет внимания: вместо того, чтобы каждый раз пересчитывать скрытый слой для всех элементов и весь софтмакс, я храню экспоненты энергии и их сумму (знаменатель софтмакса), чтобы при новом подсчете считать энергию, ее экспоненту для одного нового элемента, и пересчитать сумму, вычев и добавив по одному элементу.

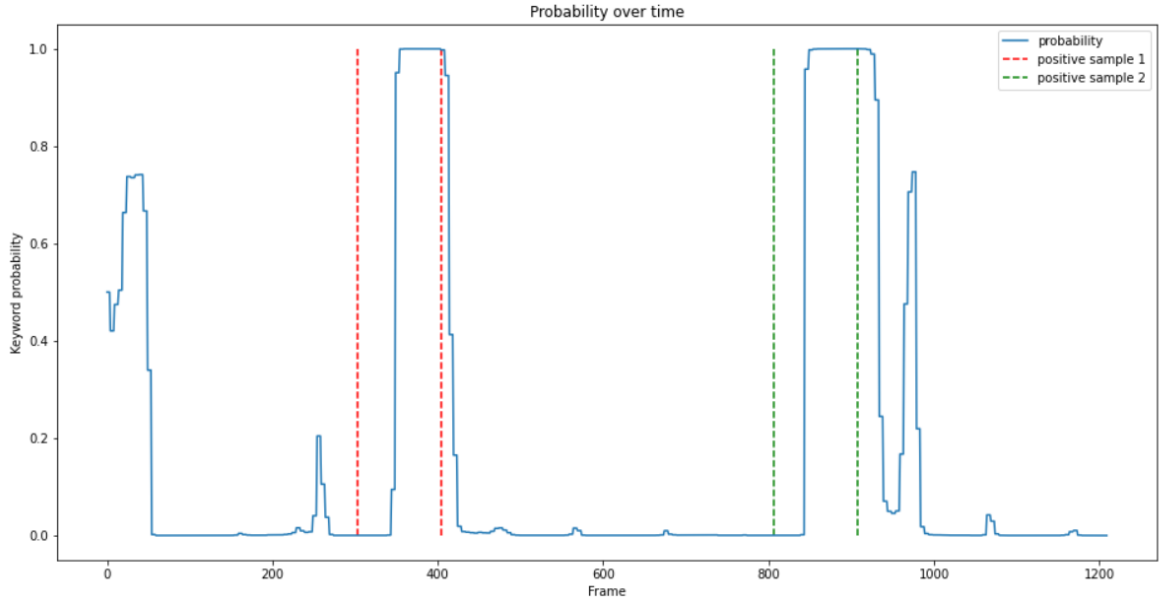
Для иллюстрации inference'а я провел пару экспериментов:

1. Возьмем 10 записей, в которых нет ключевого слова, и 2, в которой есть, склеим их (но поместим положительные в разные места), и понаблюдаем за вероятностью положительного ответа, выдаваемой моделью. Параметры: window_size=20, streaming_step_size=8.



В самом начале модель трясет, потому что контекстное окно некоторое время заполняется (полностью оно заполнится за 20 * 8 = 160 фреймов, но для адекватного качества требуется меньше). Точно по этой же причине модель еще некоторое время выдает высокую вероятность, даже когда слово уже закончилось, и низкую – когда только началось. Если это важно, то это можно обойти 2мя путями: либо вручную сбрасывать состояние после триггера (например, триггер при N фреймов с высокой вероятностью), либо уменьшить параметры выше, тогда контекст будет быстрее обновляться (но так можно потерять в качестве).

2. Попробуем второй путь, уменьшим параметры window_size=8, streaming_step_size=5.

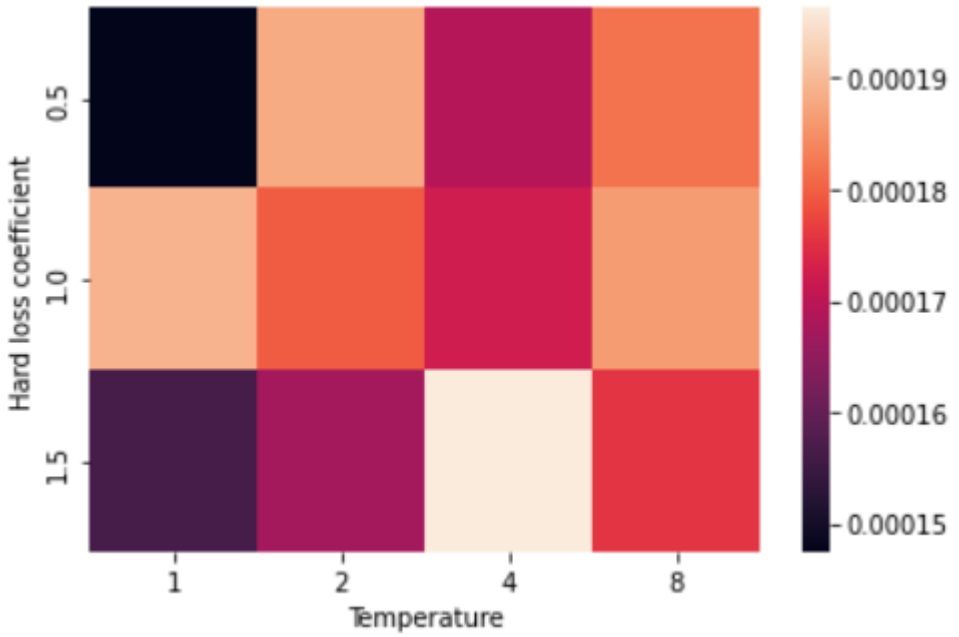


Мы видим, что задержка уменьшилась и распознавание (особенно первой записи) стало лучше, но график стал менее гладким – появились какие-то ложные пики (маленькие и большой), так что нужно аккуратнее подбирать порог.

Дистилляция

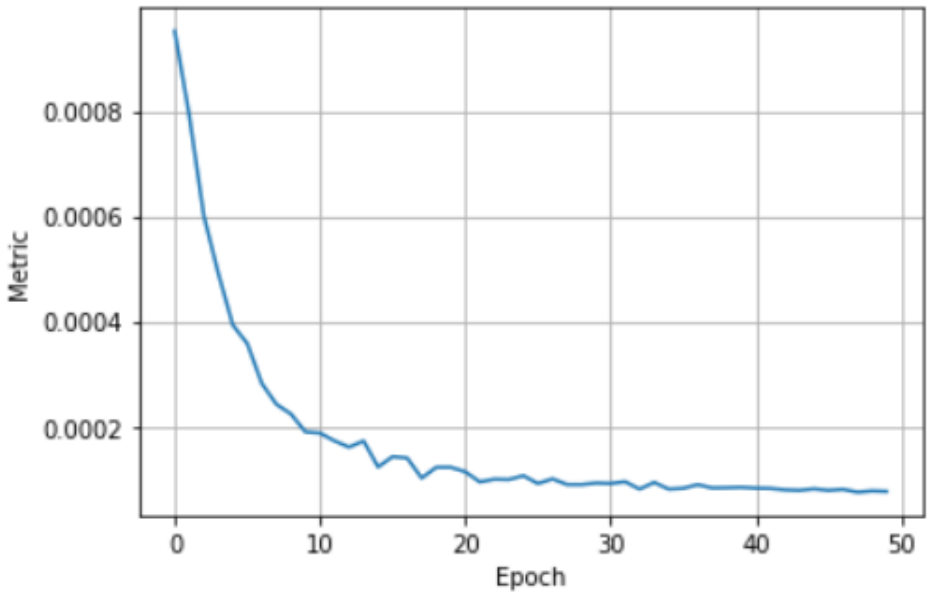
Следующим этапом я принялся оптимизировать модель по памяти и MACs. Начал я с дистилляции. Я использовал бейзлайн модель в качестве учителя, в качестве функции потерь я использовал стандартные для дистилляции кросс-энтропии между студентом и учителем с температурой T (soft-loss) и между студентом и вырожденным истинным распределением с единичной

температурой (hard-loss). Дополнительно я взвешивал hard-loss коэффициентом.
На первом этапе я не знал какие параметры температуры и коэффициента использовать, поэтому я запустил маленький grid-search на 4 часа, чтобы хотя бы прикинуть. Получил такую картину:



При этом я использовал очень маленького студента (так как grid search занимает много времени): `hidden_size=20, cnn_out_channels=3, kernel_size=(9, 20), stride=(4, 10)`. Здесь я в основном уменьшал MACs, поскольку квантизацией в дальнейшем, я гарантированно уменьшил бы размер, а про скорость непонятно (MACs точно не уменьшатся). Я также менял параметры свертки, потому что увеличенный страйд уменьшает количество применений свертки, но нужно компенсировать ее увеличенным ядром. Так я уменьшаю MACs ценой небольшого увеличения количества параметров.
Далее я с ним же запустил процесс обучения с лучшими параметрами выше, но уже подольше (50 эпох) и со scheduler'ом. Однако 2 лучших запуска не дали нужных результатов (качество порядка 1e-4), но потом я нашел статью в материалах семинара, в которой говорится, что для MNIST и маленьких моделей хорошо работают T от 2.5 до 4. При T=4 лучший результат у меня – при коэффициенте 0.5, его я и взял. Получилось чуть лучше (7.5e-5), но все равно не достаточно. Поэтому я принялся

увеличивать модель.

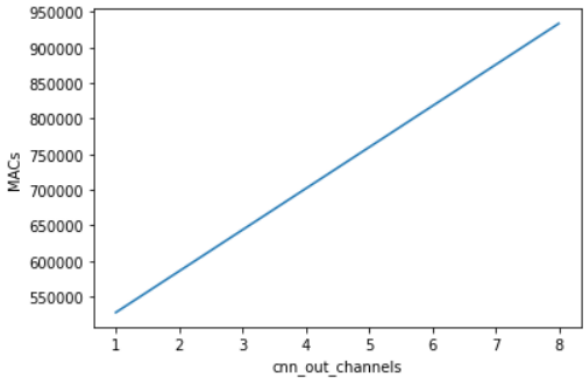


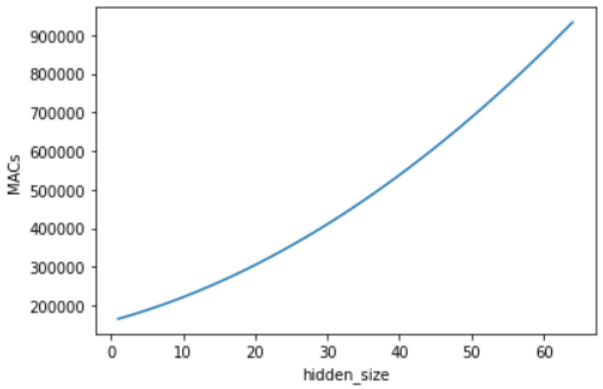

```
END OF EPOCH 50/50
```

```
1 torch.save(student_03.state_dict(), "student_03.pt")
2 print(f"Student_03 best au-fa-fr = {student_03_au}")
```

```
Student_03 best au-fa-fr = 7.454063683579143e-05
```

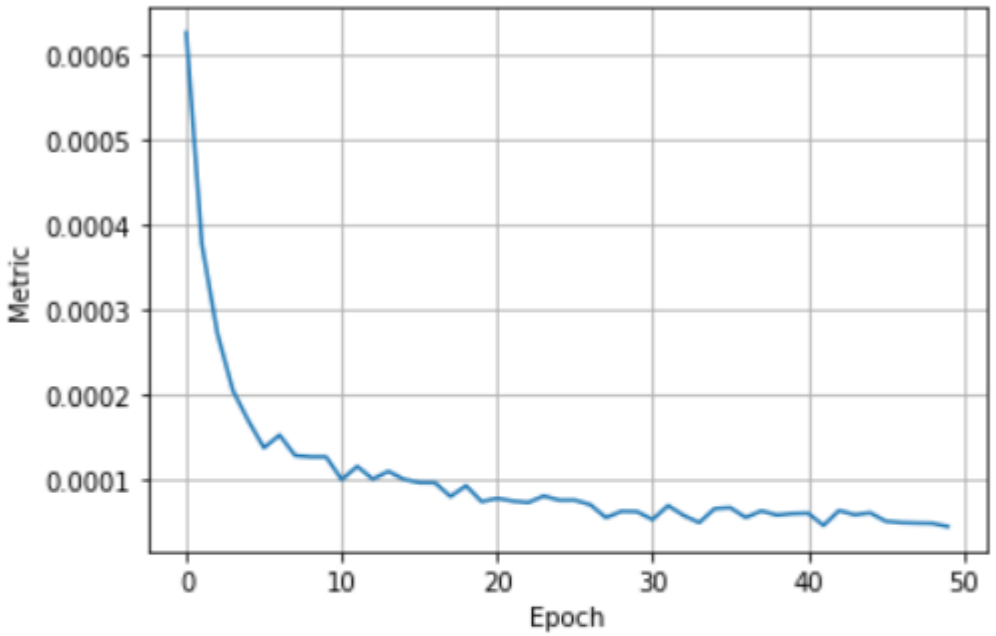
Но перед этим было интересно узнать, как влияют параметры hidden_size и cnn_out_channels на MACs, поэтому я создавал модели и оценивал MACs (для этого их не нужно обучать), получил такое:





Видно, что от hidden_size зависимость квадратичная, поэтому логично уменьшать ее и не так критично увеличивать cnn_out_channels. При этом можно еще провести манипуляцией с параметрами свертки, чтобы еще как-то поменять суммарный MACs.

Я попробовал увеличить количество каналов до 4, но это также не дало нужного результата. Далее я увеличил hidden_size до 32, но к моему удивлению, это тоже не помогло (хотя я также пробовал отдельно менять только этот параметр). Смотря на графики, я подумал, что использование низкого LR скорее мешает, потому что график качества сразу выходит на плато, поэтому отказался от scheduler'a. И это действительно помогло! Качество стало 4.5e-5, что подходит под условие. Такая модель уменьшила MACs в ~5 раз и память в ~5.5 раза.



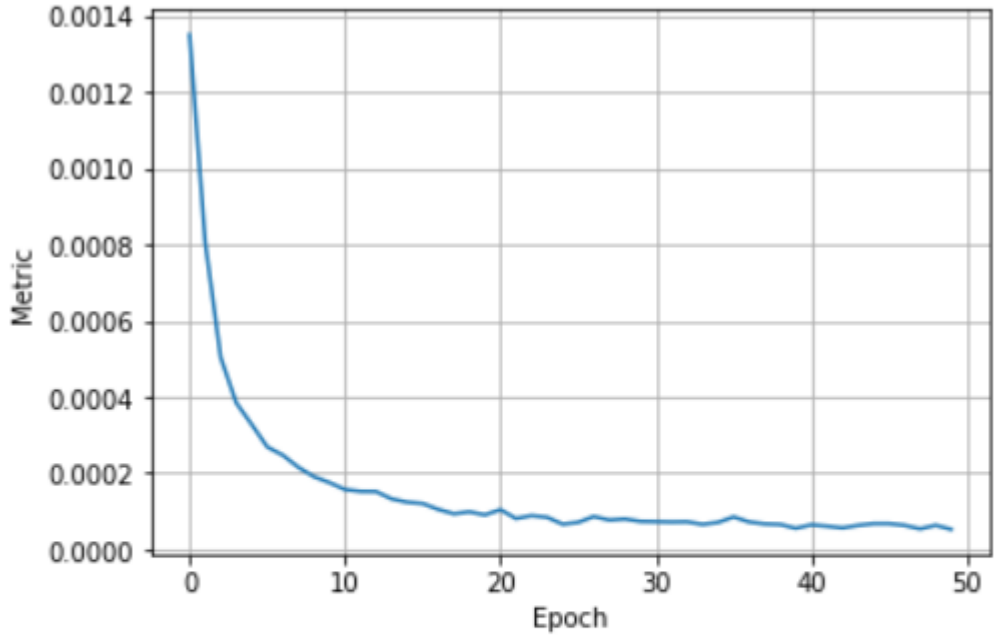

```
END OF EPOCH 50/50
```

```
  1 student_06_au
```

```
4.4523872502749175e-05
```

Учитывая успех от убирания scheduler'a и довольно большой gap до порога (5e-5 * 1.1), я подумал, что можно еще уменьшить модель. Для этого я снова уменьшил параметр hidden_size (до 20), так как он сильнее влияет на

размер и MACs. Получилось, но на тоненького (5.2e-5).




```
END OF EPOCH 50/50
```

```
 1 student_08_au
```

```
5.286053647357623e-05
```

В итоге дистилляция уменьшила мне потребление памяти в ~10 раз, ускорила в ~8.5 раза

```
Student_08
Memory ratio: 10.110951629108655
MACs ratio: 8.465460347390941
Params ratio: 10.110951629108655
```

Квантизация

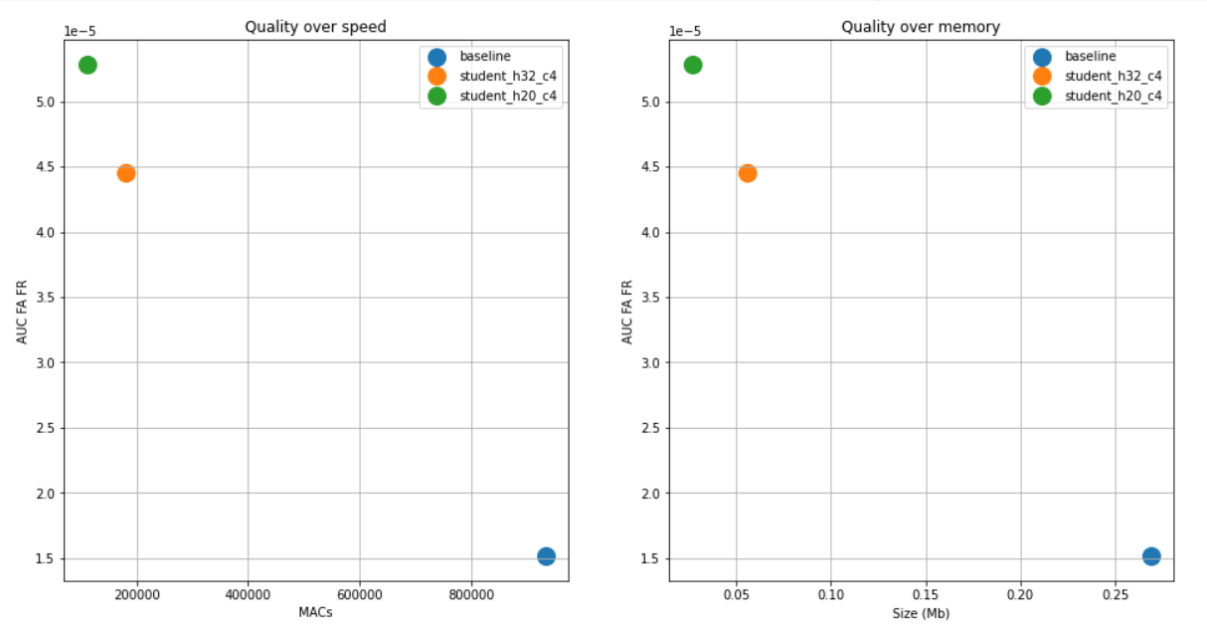
Я захотел попробовать динамическую квантизацию поверх всех моделей (3) с необходимым качеством. Возникла проблема с оценкой моделей, так как thop для них не работал, да и оценивание размера тоже. С первым особых проблем нет, потому что мы знаем, что MACs квантизованной модели совпадает с неквантизованной, то есть можно сравнить только время локально. Второе я починил вручную оценивая через

неквантизованную модель (например, если мы переводим из float32 в qint8, то уменьшаем размер нужных слоев в 4 раза).

Я провел необходимые манипуляции и увидел, что качество просело на порядок. Я немного не понял, почему качество так сильно ухудшилось, хотя однокурсники говорили, что ровно такой метод квантизации хорошо работал. Я предполагаю, что дело в версии торча (к слову квантизованные модели у меня не запускались на GPU, пришлось их обсчитывать на CPU, а в таком случае сравнивать модели на GPU и CPU бессмысленно).

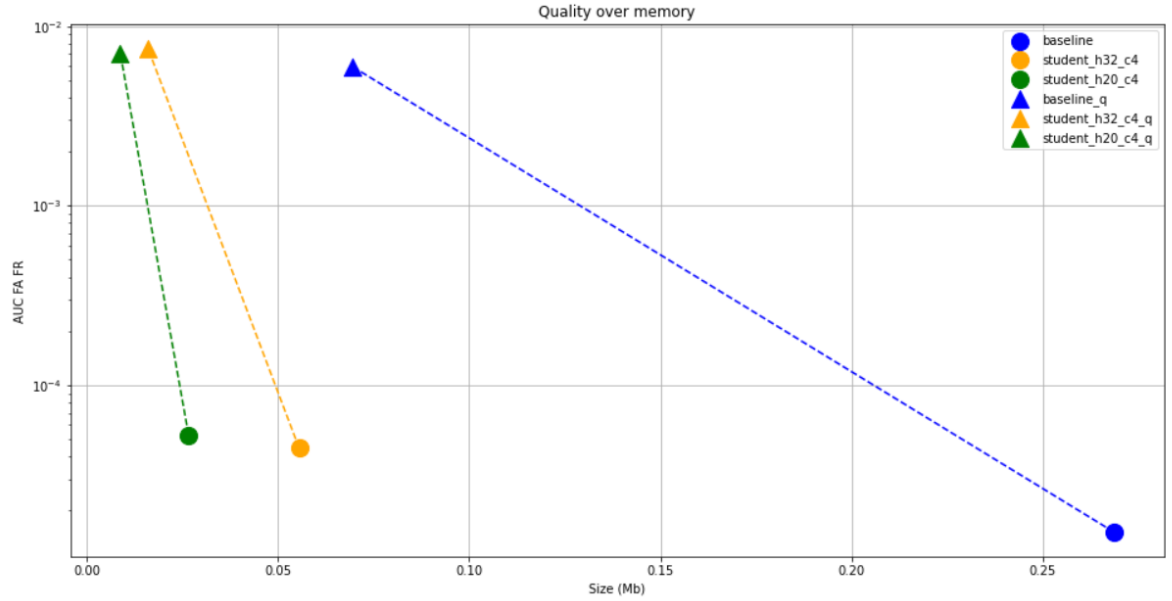
Я также попробовал float16, но там аналогично все плохо:(Я также хотел посмотреть в сторону Quantization Aware Training, но за час не успел.

Но нужно сравнить модельки, для начала я визуально сравнил неквантизованные модели:



Здесь вполне логичный трейдофф между качеством и скоростью, а также качеством и размером модели. Т.е. чем меньше модель, тем ниже качество.

Также можно посмотреть на то как меняется качество при квантизации, хотя в случае сломанного торча, это довольно сомнительно:



Здесь нужно обратить внимание на логарифмическую шкалу по качеству. Квантизованные модели отмечены треугольниками. Видно, что для квантизованных моделей выполнены почти те же соотношения, однако интересно, что средняя модель (желтая) по качеству уступает маленькой (зеленой) в квантизованном варианте.

Выводы

Итого, я обучил baseline модель, реализовал для нее стриминг и получил меньшую модель, которая в 8.5 раз быстрее и в 10 раз меньше по размеру, чем начальная модель, при этом оставаясь адекватного качества. Такой результат я получил только благодаря дистилляции, про которую довольно много читал, их очень много (попробовать все виды дистилляции = смерть), но я справился и с одной путем порядка 10 попыток и грид серча. Кроме того, мне удалось немного пощупать квантизацию и почитать про нее, но к сожалению она у меня не завелась. Наконец, про прунинг я успел только прочитать, но попробовать мне не удалось.