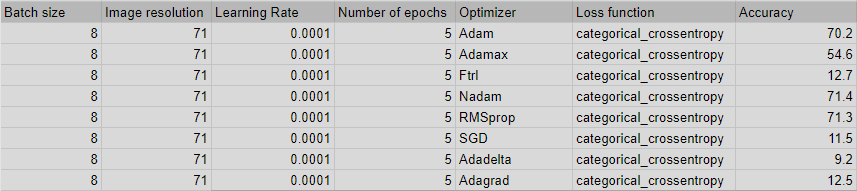
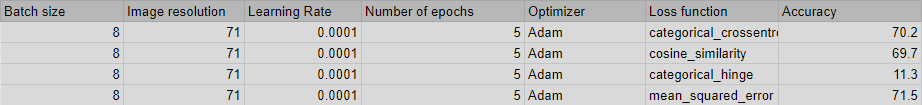
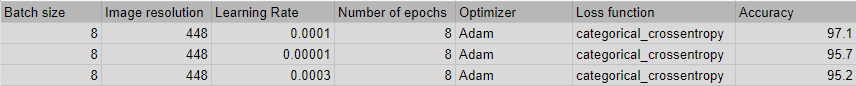
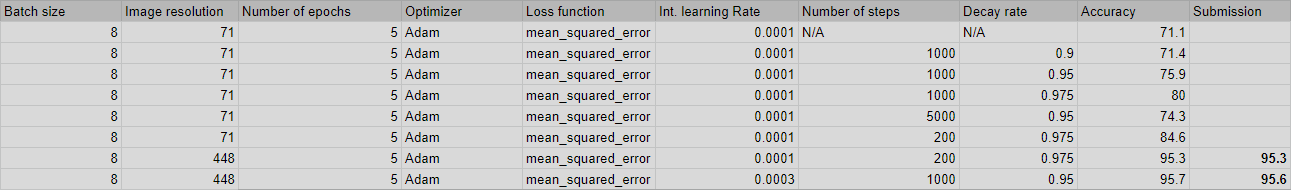
## **Отчет о проделанной работе**

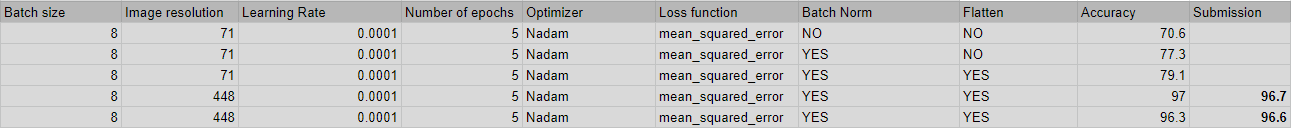
**1. Выбор модели.** Первым делом я попробовал применить одну из последних моделей линейки ImageNet наивно полагая что таким образом удастся повысить результат на submission. Была использована модель EfficientNet-L2 (480 MB) к которой в Kaggle я раздобыл веса (1.9 GB). Модель считала 6 часов и в итоге, к моему удивлению, дала результат в очень скромные 86%. После, я перешел на модель EfficientNet-B7 (66 MB) которая считала побыстрее и давала validation accuracy 95-96%. Тем не менее, работа модели всё равно занимала много времени и было решено попробывать модель EfficientNet-B3 (12 MB). Она считала гораздо быстрее и давала примерно такой же результат как и EfficientNet-B7. Важным моментом оказалось что использование легкой EfficientNet-B3 дало гибкость изменять параметры, задавать такие параметры с которыми EfficientNet-B7 не тянула и выдавала ошибки. Это оказалось ключевым моментом для повышения качества предсказаний. Стоит упомянуть что была попытка использовать модель не из линейки EfficientNet, а именно MaxVit-L. Однако, не получилось сразу наскоком ее запустить и я решил не терять времени разбираясь с ней. В общем не стоит гнаться за моделями с высокими рейтингами ибо они очень тяжелые. Работа приведенная ниже была выполнена с EfficientNet-B3.

**2. Batch size и Image resolution.** Первым важным открытием было batch size не оказывает существенного влияния на validation accuracy. В тоже самое время, небольшой batch size позволил увеличить разрешение картинок (иначе модель не принимала тяжелые данные на вход) что в свою очередь сказалось на качестве результата сильным образом. В таблице ниже приведены результаты экспериментов:  Вот так вот сразу удалось достигнуть весьма достойного результата в 97.1% на submission.

**3. Optimizer.** Не мудрствуя лукаво, я просто перебрал все доступные optimizers. Картинки были уменьшены до 71x71 для того чтобы быстрее считалось. В таблице ниже приведены результаты экспериментов:  Optimizers Nadam и RMSprop дали результаты чуть повыше чем ранее используемый Adam.

**4. Loss function.** Аналогично работе с optimizers, я перебрал несколько loss functions работающими с категориальной validation. В таблице ниже приведены результаты экспериментов:  mean\_squared\_error дала результат получше чем categorical\_crossentropy. Соответственно следующим логичным шагом было объеденить mean\_squared\_error c Nadam и RMSprop optimizers и посчитать на картинках высокого разрешения. В таблице ниже приведены результаты экспериментов: image.png К сожалению на submission улучшения не было а был лишь повторен результат 97.1%

**5. Learning rate.** Сперва я просто попробовал увеличить и уменьшить learning rate. В таблице ниже приведены результаты экспериментов:  Улучшения это не дало. Следующим шагом было применение exponential decay для learning rate где я перебирал пару ключевых параметров на картинках низкого разрешения. Удалось добиться значительного улучшения однако когда я применил exponential decay для картинок высокого разрешения результат был плох. Причина сего сего казуса осталась для меня непонятной. Также стоит отметить что Nadam и RMSprop optimizers не могли быть использованы с exponential decay. В таблице ниже приведены результаты экспериментов:  По-настоящему улучшения удалось достичь через управления learning rate через call back (см. ниже).

**6. Batch normalization и Flatten.** Batch normalization и Flatten были по-очереди добавлены к model head. Как и в случае с learning rate, validation accuracy повысился для картинок низкого разрешения, однако прогресса не было для картинок высокого разрешения. В таблице ниже приведены результаты экспериментов:  В конечной модели Batch normalization и Flatten были отключены.

**7. Callback.** В работе были использованы следующие функции callback: -checkpoint (для сохранения лучшей итерации) -stopping (остановка обучения при отсутствии прогресса) -reduce\_lr (управление learning rate) Последняя функция оказалась весьма действенной и дала улучшение на submission (97.38). В таблице ниже приведен результат эксперимента: image.png

**8. Abumentation.** Была попытка использования более продвинутой Albumentation (см. код ниже) однако результаты были низки. Должно быть я не разобрался с параметрами.

**9. Fine tuning.** Fine tuning был сделан в 4 шага: сперва head, а потом подгружалась base model по 1/3. Соответственно learning rate не управлялся через callback а уменьшался вручную на каждом шаге: 1e-3, 1e-4, 3e-5, 1e-5. Validation accuracy улучшалось соответственно: 93.26, 94.93, 95.84, 96.14 и результат нв сабмишене всего 96.32%. Относительно низкий результат скорее всего получился из-за неоптимального learning rate.

**10. Assembling.** Model assembling был осуществлен комбинацией двух моделей: Xception (95.71% на submission) и EfficientNet-B3 (97.24% на submission). Был сделан простенький перебор коэффициентов: Xception/EfficientNet-B3 = 0.33/0.67, 0.30/0.70 и 0.25/0.75 с результатами на submission 97.41, 97.43 и 97.41% соответственно. Таким образом лучший результат в 97.43 действительно оказался лучше чем каждая модель по отдельности. Assembling работает!

**Финальный аккорд** В попытке прорваться в элиту состязания, разрешение картинок было повышено до 544x544, было использовано больше epochs и Assembling Xception с EfficientNet-B3. Финальный результат на submission следующий:

Xception: 96.48%

EfficientNet-B3: 97.38%

Assembling: 97.438%

По-сути, без изменений.

Место: 34/285