```
for data_zip in ['train.zip', 'test.zip']:
    with zipfile.ZipFile("/content/drive/MyDrive/Ford_vs_Ferrari/" \
                +data_zip,"r") as z:
        z.extractall(PATH)
```

Это можно было бы записать проще (работает только в IPython-средах):

```
!unzip ../input/train.zip -d {PATH}
!unzip ../input/test.zip -d {PATH}
```

```
ModelCheckpoint('best_model.hdf5' , monitor = ['val_accuracy'] , verbose =
 1   , mode = 'max')
```

Здесь надо добавить save_best_only=True, иначе модель будет сохраняться каждую эпоху независимо от точности.

```
model.fit_generator(
```

Это устаревший метод, сейчас он эквивалентен model.fit.

```
model.save('../working/model_last.hdf5')
model.load_weights('best_model.hdf5')
```

Это одна и та же модель, потому что в ModelCheckpoint вы не указали save_best_only=True.

- Добавим новые аугментации и посмотрим как это повлияет на качество

Здесь есть две методологические проблемы. Во-первых, сильные аугментации могут усложнять задачу обучения, поэтому обучение может идти медленнее, но достигать в итоге лучшей точности. То есть 5 эпох может быть недостаточно для сравнения. Во-вторых, процесс инициализации и обучения случаен, поэтому даже с одними и теми же аугментациями вы будете получать каждый раз немного разную точность. Может быть разница в результатах на вашем эксперименте (68.5% и 71%) – обусловлена просто случайностью.

```
rescale=1. / 255
```

Для сети Xception лучше пользоваться функцией keras.applications.xception.preprocess_input вместо деления значений пикселей на 255. Тогда файн-тюнинг будет идти с той же нормализацией изображений, что и обучение сети на ImageNet, что может в итоге повысить точность.

- Вот, это пока лучший вариант с аугментациями! надо на его примере сделать submit 71.63%

Хорошо было бы свести все результаты и то, как они получены, в таблицу. У вас очень много кода дублируется, из-за чего размер ноутбука растет, и вы сами можете в итоге запутаться в своем ноутбуке, а тем более тот, кто проверяет. Было бы удобно создать функцию, которая принимает на вход набор аугментаций, создает генераторы, создает и обучает сеть, и просто запустить эту функцию несколько раз.

- Пробуем с аугментациями с лучших работ с kaggle

Скажу по секрету, действительно хороших работ в соревновании Car classification в разделе «ноутбуки» нет. Все просто пробуют что-то случайное, нет ни одной работы (насколько я знаю), где бы обстоятельно и надежно сравнивались разные подходы. Поэтому если хотите научиться обучать сверточные сети, не смотрите ноутбуки в данном соревновании, а откройте какое-то публичное соревнование на Kaggle по задаче классификации изображений и смотрите ноутбуки там.

- VAL_SPLIT = 0.05, дает лучший результат, будем в дальнейшем использовать его!

Понижая val_split вы увеличиваете кол-во обучающих изображений и уменьшаете кол-во валидационных. Конечно чем больше обучающих изображений, тем выше среднестатистически будет точность. Но чем меньше кол-во валидационных изображений, тем менее надежной будет оценка точности.

Вообще если вы собираетесь и дальше проводить эксперименты (а я вижу что следом у вас идет эксперимент по сравнению learning rate и т. д.), то val_split лучше увеличивать. Так оценка точности будет более надежной. И так понятно, что чем больше обучающих изображений – тем выше точность. То есть гиперпараметры стоит подбирать, например, с val_split=0.5, а обучать для сабмита с val_split=0.05 или даже val_split=0.

- Теперь поработаем с learning_rate

Снова много дублирования кода. Вместо того, чтобы копипастить код по 10 раз, можно создать функцию, которая принимает гиперпараметры, создает, обучает и оценивает модель. Так объем ноутбука был бы раз в 5 меньше, и читаемость была бы выше.

- Получилось, что learning_rate = 1e-3 ухудшил модель: accuracy = 65.18%

По одному эксперименту нельзя с точностью такое утверждать. Влияет не только стартовый learning rate, но и динамика его изменения. Если вы решили начать с lr=1e-3, то после нескольких эпох уменьшайте его до 1e-4 и далее. Такая стратегия называется ступенчатым затуханием, или piecewise-constant decay.

- Learning rate =1e-5 так же сильно ухудшил accuracy

Это ожидаемо: с таким маленьким lr модель просто не успела обучиться.

- Результат использования EfficientNetB3 очень низкий Accuracy всего 12.11%. Не буду использовать эту сеть для дальнейших экспериментов

При такой точности очевидно, что где-то в эксперименте ошибка. Во-первых EfficientNet принимает изображения со значениями пикселей от 0 до 255, а у вас делается деление на 255, из-за чего значения пикселей становятся от 0 до 1. Вообще EfficientNet – очень хорошая модель, мне удавалось обучить ее до точности 98.2% на сабмите.

При обучении EfficientNet в течение 5 эпох у ва наблюдается ситуация, что точность на трейне растет (35%), а на валидации расти не торопится (13%). Причина низкой точности на трейне вероятно в том, что вы неправильно нормализуете входные данные. Причина сильного различия между трейном и валидацией скорее всего в слоях батч-нормализации, они медленно «разогреваются» прежде чем начать хорошо работать. Причину этого можно понять, изучив принцип работы этих слоев. Кроме того существует такая проблема как «batch-normalization train-test dicsrepancy», можете поискать в интернете где про нее подробно написано.

- Из экспериментов видно, что выбор нейронной сети под конкретную задачу имеет значение и оказывает серьезное влияние на результат.

Конечно выбор сети имеет значение, но из экспериментов этого не видно. Прежде всего их нужно корректно поставить (делать корректную нормализацию входных данных). Также нужно обучать больше эпох и желательно несколько раз, чтобы результат был надежнее. И даже этого может быть недостаточно. Размер изображения и learning rate – это гиперпараметры, и разные сети могут быть по-разному чувствительны к этим гиперпараметрам. Например, условная сеть А дает точность 95% при размере 200x200 и 90% при размере 100x100, а сеть В - точность 95% при размере 200x200 и 50% при размере 100x100.

Ну и в целом все сети из keras.applications уже не являются SOTA, есть более новые и эффективные сети, например такая: https://tfhub.dev/google/collections/efficientnet_v2/1

- Попробуем реализовать батч-нормализацию ради эксперимента, посмотрим как как повлияет на метрику

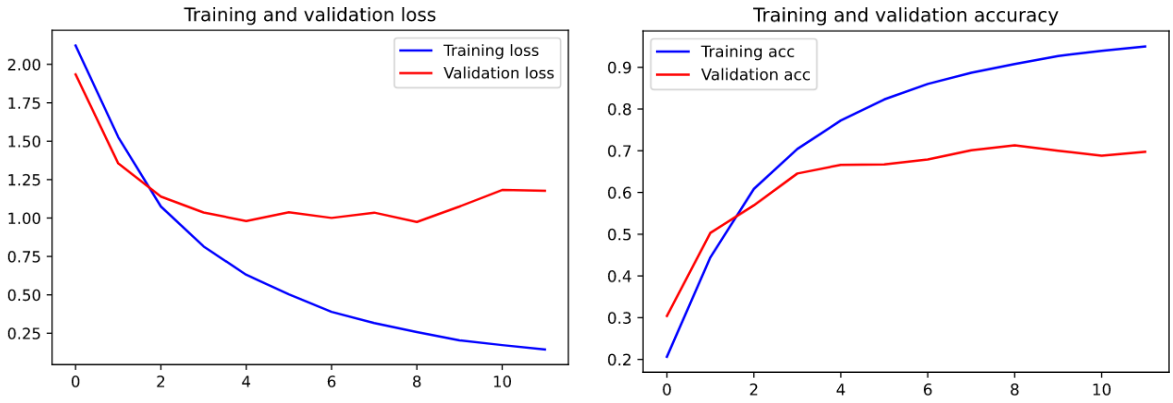
```
test_generator = train_datagen.flow_from_directory(
    PATH+'train/',
```

Здесь у вас почему-то валидация делается с аугментациями, а ранее делалась без аугментаций. Аугментации оказывают сильное влияние на точность при валидации. Поэтому сравнение точности в этом эксперименте с предыдущими экспериментами будет некорректным.

- К тому же была получена информация, что в голову батч-нормализацию не принято добавлять.

Да, не принято. Но в DL вообще много подобных «верований» и во всем нужно сомневаться и проводить эксперименты. Хорошо, что вы их проводите. Но с другой стороны в экспериментах по-моему у вас ошибки, о чем я писал выше (слишком маленький val_split, неверная нормализация, валидация с аугментациями, недостаточное кол-во эпох и экспериментов чтобы сделать надежные выводы).

- Что и требовалось доказать! Эпохи после наступления момента переобучения только отнимают машинное время и ухудшают результат



Графики выглядят убедительно, но либо это просто случайная флуктуация точности, либо где-то ошибка в эксперименте. Точно могу сказать, что на этом датасете сети можно обучать десятки эпох без переобучения. Конечно график точности на валидации в ходе обучения случайно колеблется вверх-вниз, но в целом все же имеет тенденцию к росту. Скорее всего то, что у вас – просто случайное колебание.

Кстати, возможно что ваша сеть в самом деле переобучается, но это из-за наличия скрытого полносвязного слоя:

```
Dense(256, activation='relu')(x)
```

Убрав его вы сможете обучать сеть гораздо дольше без переобучения.

Еще возможно, что вы все-таки выбрали слишком низкий learning rate (1e-4). С низким learning rate модель сильнее переобучается. Для задач computer vision оптимален обычно lr в районе 1e-3 (есть кстати способ нахождения оптимального стартового lr, но он тоже имеет свои проблемы). Затем learning rate в ходе обучения уменьшают, обычно ступенчато. Для этого можно использовать callback ReduceLROnPlateau.

- использование finetuning в данной ситуации не принес пользы, мы получили Accuracy: 59.99%, что ниже Accuracy: 71.63%, которую выдает модель без постепенной разморозки слоев.

Вообще под термином файн-тюнинг понимается дообучение сети под задачу любым способом (с постепенной разморозкой слоев или сразу всю).

```
A.OneOf([
        A.CenterCrop(height=224, width=200),
        A.CenterCrop(height=200, width=224)],
```

Если мы делаем аугментации, то почему не использовать RandomCrop?

Делая много аугментаций старайтесь проверять как они сказываются на производительности процесса обучения. Замерьте время эпохи с аугментациями и без них. Возможно есть какая-то одна «тяжелая» аугментация, которая в 2 раза замедляет весь процесс без существенного влияния на результат.

```
# сделаем несколько предсказаний одной и той же картинки
# усредним эти предсказания
# сохраним в сабмит
test_sub_generator.reset()
predictions_tta = []
for _ in range(EPOCHS):
    predictions_tta.append(model.predict(test_sub_generator, verbose=1))
    test_sub_generator.reset()
```

Вы делаете предсказания на test_sub_generator, в котором *не делаются аугментации*. А значит все итерации цикла TTA дают в точности одинаковые предсказания. Я бы посоветовал визуализировать предсказания (predictions) с помощью plt.imshow чтобы увидеть, дают ли разные шаги TTA разные предсказания или одни и те же. Скорее всего вы бы увидели, что одни и те же.

- Примененние TTA не улучшило мой результат(

Причина как раз описана выше.

- Подаем на вход сети картинки размером 320x320 и получаем улучшение метрики! Предположу что в результате подачи в сеть изображений размером 512x512 получим результат еще лучше чем в эксперименте с изображениями 320x320

Да, увы, многие из современных CNN очень зависят от масштаба изображений. Чем больше размер изображений – тем выше точность.

Вероятно причина сильной зависимости точности от масштаба в том, что сети при принятии решения ориентируются на текстуры и отдельные участки, а не на общую форму объекта. Например об этом можно почитать здесь.