```
!pip install -U efficientnet
```

Сейчас есть более легкий способ использовать EfficientNet, так как эта сеть есть в модуле keras.applications.

```
# Will unzip the files so that you can see them..
for data_zip in ['train.zip', 'test.zip']:
    with zipfile.ZipFile(DATA_PATH+data_zip,"r") as z:
        z.extractall(PATH)
```

Как вариант, это можно было бы записать проще (работает только в IPython-средах):

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

```
print('Средняя длина картинки', mean(image_size_x_list))
print('Max длина картинки', max(image_size_x_list))
print('Min длина картинки', min(image_size_x_list))
print('Средняя ширина картинки', mean(image_size_y_list))
print('Max ширина картинки', max(image_size_y_list))
print('Min ширина картинки', min(image_size_y_list))
```

В ноутбуке к сожалению стерты выходы ячеек кода.

```
albumentations.Transpose(p=0.5)
```

Эта аугментация транспонирует изображение, то есть «кладет его на бок». Обычно аугментации делают так, чтобы получались реалистичные изображения, как если бы они были собраны из интернета. Положенное на бок изображение автомобиля – нереалистичное, поэтому я бы не рекомендовал так делать.

```
albumentations.ShiftScaleRotate(p=0.5),
albumentations.Rotate(limit=30,
```

У вас дважды делается поворот, плюс еще транспонирование. Все это очень сильные аугментации, которые могут замедлить процесс обучения модели и при этом привести к более низкой точности на сабмите. Аугментации все-таки должны быть умеренными по силе.

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

Почему-то у 90% тех, кто делает этот проект, вставлена эта строчка кода. Но если мы делаем аугментации, почему не использовать RandomCrop вместо CenterCrop?

```
test_generator = train_datagen.flow_from_directory(
```

У вас валидация делается с аугментациями. Так не должно быть. Представьте, что одну модель вы обучили на слабых аугментациях, другую на сильных. Если валидация делается с аугментациями, то модель с сильными аугментациями покажет более низкую точность на валидации. Если же делать валидацию нормально, без аугментаций, то все может быть наоборот. Поскольку валидация влияет на выбор моделей и гиперпараметров, важно делать ее без аугментаций.

Кроме того, было бы полезно делать визуализацию изображений, полученных аугментациями. Вдруг окажется, что в аугментациях ошибка, и изображения поулчились совсем не такие, как надо? Нужно проверять результат аугментаций прежде чем начинать обучение.

```
ModelCheckpoint('model_1_best.hdf5' , monitor = ['val_acc'] , verbose = 1
, mode = 'max')
```

Надо добавить save_best_only=True, иначе модель будет сохраняться на всех эпохах независимо от точности на валидации.

- Пофаинтюним, но немного.

Представьте, что вам на работе дали задание обучить сеть для классификации – и вы показываете как результат своей работы данный ноутбук. В разделе, где обучается модель, есть только код – нет никаких комментариев и даже выводов ячеек кода. Нужно как минимум показать график точности при обучении.

```
def make_callbacks():
```

У вас 4 раза в ноутбуке объявляется одна и та же функция, зачем? Если вы вносите изменения в коллбэки, то передавайте из как параметры в функцию make_callbacks.

Вообще весь код заморозки слоев, компиляции и обучения можно завернуть в функцию – и тогда файн-тюнинг будет просто четырьмя вызовами этой функции, то есть уместится в 4 строки кода.

Если заглянуть в сообщество ods.ai в slack, то там любят использовать смайлик “no-jupyter”. То есть многие не любят jupyter-ноутбуки, и есть на это причины. Jupyter-ноутбуки вызывают привыкание к не очень хорошему code-style, когда вместо написания функции и использования ее несколько раз – просто копируются ячейки с кодом. В итоге код получается большим, плохо читаемым и даже автор кода может в итоге в нем запутаться.

```
pred = np.mean(predictions, axis=0)
```

Было бы еще полезно посмотреть визуально на предсказания (plt.imshow). Так можно увидеть насколько сильно различаются результаты на разных шагах TTA.

Еще можно делать так: сделать сначала одинаковое количество шагов TTA на каждом изображении, а затем делать дополнительные шаги на тех изображениях, в которых результат остался спорным.

- Так же я плохо разобрался с LR. Я так и не понял имеет-ли смысл задавать "расписание" для LR при компиляции модели в оптимайзере, если оно задается в функции callbacks. Я думаю, что проработка этого вопроса помогла бы мне улучшить результат.

Вы могли бы обратиться в slack с этим вопросом во время выполнения проекта. В оптимайзере расписание LR задается для каждого шага (батча), а в callbacks (LearningRateScheduler) – для каждой эпохи. В этом вся разница. Еще в callbacks можно использовать ReduceLROnPlateau.

Но у вас почему-то одновременно используются LearningRateScheduler и ReduceLROnPlateau, так не должно быть, ReduceLROnPlateau в этом случае вероятно не будет оказывать влияния.