```
!pip install -q 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("/kaggle/input/sf-dl-car-classification/"+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}

```
AUGMENTATIONS = albumentations.Compose([ albumentations.Transpose(p=0.5), albumentations.Flip(p=0.5),
albumentations.OneOf([ albumentations.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3),
albumentations.RandomBrightnessContrast(brightness_limit=0.1, contrast_limit=0.1) ],p=1),
albumentations.GaussianBlur(p=0.05), albumentations.HueSaturationValue(p=0.5), albumentations.RGBShift(p=0.5), ])
```

dataloaders ¶

```
train_datagen = ImageDataAugmentor( rescale=1./255, augment=AUGMENTATIONS, preprocess_input=None)

train_generator = train_datagen.flow_from_directory( PATH+'train/', target_size=(IMG_SIZE, IMG_SIZE),
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=True, subset='training')

val_datagen = ImageDataAugmentor(rescale=1./255)

test_generator = train_datagen.flow_from_directory( PATH+'train/', target_size=(IMG_SIZE, IMG_SIZE),
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=True, subset='validation')

test_sub_generator = val_datagen.flow_from_dataframe( dataframe=sample_submission, directory=PATH+'test_upload/',
x_col="Id", y_col=None, shuffle=False, class_mode=None, target_size=(IMG_SIZE, IMG_SIZE), batch_size=BATCH_SIZE,)
```

```
In [17]:
train_datagen = ImageDataGenerator(
    rescale=1. / 255
```

Какой-то странный блок кода, вставленный в виде текста. Если это лишнее, то лучше это убрать.

```
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.

```
Transpose()
```

Зачем вы делаете транспонирование изображения? Обычно аугментации делают так, чтобы получались реалистичные изображения, как если бы они были собраны из интернета. Положенное на бок изображение автомобиля – нереалистичное, поэтому не стоит так делать. Вероятно точность за сабмите улучшилась бы, если убрать эту аугментацию.

```
ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.2, rotate_limit=45, p=0.2),
        OneOf([
            OpticalDistortion(p=0.3),
            GridDistortion(p=0.1),
            IAAPiecewiseAffine(p=0.3),
        ], p=0.2),
        OneOf([
            CLAHE(clip_limit=2),
            IAASharpen(),
            IAAEmboss(),
            RandomBrightnessContrast(),
        ], p=0.3),
        HueSaturationValue(p=0.3),
```

Я бы посоветовал убрать этот копипаст с официальной документации по albumentations и сделать более простые аугментации. Достаточно будет поворота, масштабирования, изменения яркости, контраста, цветов. Можно также добавить cutout.

Приведенные выше аугментации (такие как OpticalDistortion и прочие) могут быть сильно затратными по времени. Попробуйте сравнить время одной эпохи с этими аугментациями и без них – скорее всего без них будет в разы быстрее.

```
train_generator = train_datagen.flow_from_directory(
```

Почему-то у вас в ноутбуке дважды создаются генераторы (UPD: посмотрел далее – даже не дважды, а 4 раза). Если вам зачем-то нужно создавать их несколько раз – то сделайте функцию для их создания и вызывайте ее много раз, так код будет более лаконичным.

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.

```
model.fit_generator(
model.evaluate_generator(
```

Это устаревшие методы, сейчас они эквивалентны fit и predict, а в следующих версиях keras могут быть удалены.

В целом не совсем понятна структура ноутбука. В первой части ноутбука вы создате модель, обучаете ее всего 2 эпохи и делаете сабмит. Видимо это делается в качестве теста перед дальнейшим, более длительным обучением. Но стоило все-таки как-то прокомментировать, намного легче разобраться в ноутбуке, снабженном комментариями.

Представьте, что вам на работе дали задание обучить сеть для классификации – и вы показываете как результат своей работы данный ноутбук, без комментариев и с множеством дублирований в коде (например, по нескольку раз создаются генераторы). Как оценят такую работу? Я бы посоветовал делать код более лаконичным, создавая функции и используя их вместо копирования участков кода, и добавляя комментарии, желательно подробные.