> Нотубук для google colab - sf-car-classification-emelyanovan-google-colab.ipynb Ноутбук для kaggle - sf-car-classification-emelyanovan-kaggle.ipynb

Я правильно понял, что они ничем не отличаются кроме способа загрузки данных? Тогда смотрю ноутбук для Colab.

```
# Загружаем обвязку под keras для использования продвинутых библиотек аугментации, например, albuminations
!pip install git+https://github.com/mjkvaak/ImageDataAugmentor -q
```

В ImageDataGenerator есть есть параметр preprocessing_function, где тоже можно выполнить аугментацию через albumentations. Поэтому ImageDataAugmentor, как мне кажется, не особо здесь необходим.

```
# Установим пакет с обученными сетями EfficientNetB3, EfficientNetB5, EfficientNetB6...
!pip install -q efficientnet
```

В новых версиях Keras эти сети есть в модуле keras.applications, и устанавливать дополнительную библиотеку не обязательно. Кстати, можно посмотреть сети с Tensorflow Hub, там есть еще более новые архитектуры, обученные на большом объеме данных.

```
# Устаналиваем конкретное значение random seed для воспроизводимости
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
PYTHONHASHSEED = 0
```

Воспроизводимость будет в генераторе изображений, но в обучении ее не будет, так как во-первых для детерминированной инициализации весов нужно устанавливать seed в tensorflow (tf.random.set_seed), а во-вторых обучение сверточных сетей на GPU недетерминированно, то есть даже с одинаковой инциализацией получатся разные результаты. Но ничего страшного в этом нет, точная воспроизводимость не обязательна.

```
my_datagen_classic = ImageDataGenerator(
    rescale=1. / 255,
```

Для разных сетей нужна разная нормализация. Например, для pip-пакета efficientnet согласно этому примеру из документации, нужна нормализация функцией efficientnet.keras.preprocess_input. Это не то же самое, что деление на 255.

```
rotation_range = 50,
```

Очень большой угол поворота, так сеть может существенно дольше обучаться.

```
alb_aug.OneOf([
        alb_aug.CenterCrop(height=150, width=120),
        alb_aug.CenterCrop(height=150, width=220),
    ],p=0.5),
```

Если делаем аугментации, то почему бы нам не использовать random crop вместо center crop?

```
def print_aug_image(image):
    image = image.astype(np.uint8)
    return AUGMENTATIONS(image=image)['image']
```

Если я правильно помню, albumentations принимает изображения в формате uint8, а ImageDataGenerator возвращает их в формате float32 от 0 до 255. Так что по-моему здесь вы все делаете верно, нужно преобразовывать тип в uint8.

```
my_datagen_albu = ImageDataGenerator(
    preprocessing_function=print_aug_image
)

# Произведем аугментацию данных
train_datagen = ImageDataAugmentor(
    rescale=1./255,
    augment = AUGMENTATIONS,
    validation_split=VAL_SPLIT,
)
```

Когда вы сравниваете albumentations с аугментацией встроенными методами, то с albumentations используете ImageDataGenerator, и все работает корректно. Получается, что ImageDataAugmentor все-таки не нужен.

```
valid_datagen = ImageDataGenerator(
```

Может быть это дело вкуса, но все-таки рекомендую сокращение «val», а не «valid». Слово valid имеет совсем другой смысл.

```
model.add(L.Dense(256, activation='relu', bias_regularizer=l2(1e-
4), activity_regularizer=l2(1e-5)))
model.add(L.BatchNormalization())
```

Почти все добавляют в «голову» эти слои, но пока еще никто из сдающих не смог обосновать, зачем они там нужны. Ведь больше слоев не всегда значит лучше.

```
ReduceLROnPlateau(monitor='val_loss', factor=0.5, min_lr=1e-
6, patience=2, verbose=1)
```

У вас параметр patience равен 2. Представьте, что вы тренируете спортсмена, и прекращаете тренировки если в течение 2 тренировок подряд его результаты не растут. В итоге спортсмен мог бы тренироваться годами, а вы прекратите тренировать его уже через неделю. Но это ведь неправильно: может быть много случайностей, которые влияют на результат в какой-то из дней. Так же и с нейросетями: график точности подвержен случайным колебаниям, поэтому patience лучше делать существенно выше. Количество эпох можно поставить равным 1000 и добавить callback EarlyStopping: обучение остановится тогда, когда точность на валидации перестанет расти.

```
ModelCheckpoint(f'{MODEL}_best.hdf5', monitor=['val_acc'], verbose=1, mode
 ='max')
```

Здесь нужно добавить save_best_only=True, иначе параметры monitor и mode не будут иметь эффекта, и модель будет сохраняться каждую эпоху.

```
# Для начала заморозим веса базовой и обучим только "голову".
# Делаем это для того, чтобы хорошо обученные признаки на Imagenet не зати
рались в самом начале нашего обучения
base_model.trainable = False
```

Спорный вопрос, на практике файн-тюнинг сразу всей сети целиком часто дает более высокую точность.

```
os.popen(f'cp /content/{MODEL}_best.hdf5 "{DRIVE_PATH}{MODEL}_{STEP}_best.hdf5"')
```

Это хороший вариант для скрипта .py, но в ноутбуке можно просто командой:

```
!cp /content/{MODEL}_best.hdf5 {DRIVE_PATH}{MODEL}_{STEP}_best.hdf5
```

Такая команда понимает аргументы в фигурных скобках.

```
label_map = (train_generator.class_indices)
label_map = dict((v,k) for k,v in label_map.items()) #flip k,v
predictions = [label_map[k] for k in predictions]
```

Мне кажется что конкретно на данном датасете смысла в этом действии нет, ведь label_map будет содержать записи такие как 1 -> «1». То есть удалив эти две строки, вы получите тот же результат в файле submission.csv.

```
for i in range(tta_steps):
    preds = model.predict(test_sub_generator, steps=len(test_sub_generator), verbose=1)
    predictions.append(preds)
```

Здесь можно было бы сначала попробовать применить TTA к валидационному датасету и сравнить точность:

1. Без TTA
2. С одной итерацией TTA
3. С N итерациями TTA

Тогда было бы видно, повышает TTA точность или нет, и сколько итераций TTA оптимально делать.

> Добавлена Batch Normalization в архитектуре “головы” модели

Тут надо сравнивать: точность с батч-нормализацией и без нее. Возможно, что без нее было бы не хуже.

> Использована более продвинутая библиотека аугментации изображений albumentations

Библиотека продвинутая, но как показывает практика – слишком много аугментаций часто ни к чему и не улучшают точность. Хороший набор аугментаций есть в слоях Keras (например RandomCrop). Такие аугментации выполняются на GPU с хорошей производительностью.

В целом хороший код, хорошая точность получилась.