> Подгрузим отдельный класс, который поддерживает albumentations

```
!pip install git+https://github.com/mjkvaak/ImageDataAugmentor
from ImageDataAugmentor.image_data_augmentor import *
```

В ImageDataGenerator есть параметр preprocessing_function, где тоже можно выполнить аугментацию через albumentations, поэтому в целом нет необходимости ставить дополнительную библиотеку ради этого.

```
AUGMENTATIONS_TEST_CLEAR = A.Compose([
    A.ToFloat(max_value = 255)
    ])
```

По идее, это можно убрать, если здесь не делаются аугментации. Перевод во float обычно выполняется автоматически.

```
train_datagen_hard = ImageDataAugmentor(validation_split=0.2,
augment=AUGMENTATIONS_TEST_HARD, seed=RANDOM_SEED)
test_datagen_hard = ImageDataAugmentor(augment=AUGMENTATIONS_TRAIN,
seed=RANDOM_SEED)
```

Получается несоответствие между именами: в train hard используются одни аугментации, а в test hard другие, хотя название схожее.

```
!pip install efficientnet
import efficientnet.tfkeras as efc
```

Библиотека efficientnet была актуальна года два назад, но с тех пор EfficientNet появился в модуле keras.applications, так что устанавливать доп. библиотеку не обязательно.

Можно было еще посмотреть более новые модели из tensorflow hub.

```
base_model = efc.EfficientNetB6(weights='imagenet', #Предобучение модели на
ImageNet, используем FineTuning
                                include_top=False,
                                input_shape = input_shape)
```

При обучении EfficientNetB6 у вас не делается нормализация изображений. Это было бы правильно, если бы использовалась EfficientNet из keras.applications, но EfficientNetB6 из pip-пакета efficientnet требует нормализации, как показано в примере.

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

Здесь нужно добавить параметр save_best_only=True, иначе будет сохраняться каждую эпоху.

Если после изменения BATCH_SIZE или IMG_SIZE вам нужно пересоздать генераторы, то нет необходимости копировать весь код, можно создать функцию для создания генераторов и вызывать ее дважды. Если вам нужног создать 8 генераторов, то функция может возвращать словарь имя-генератор.

```
model_2 = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu', bias_regularizer=l2(1e-
4),activity_regularizer=l2(1e-5)),
```

```
    BatchNormalization(),
    Dropout(0.25),
    Dense(CLASS_NUM, activation='softmax')
])
```

Зачем после увеличения IMG_SIZE до 512 заново создавать голову для модели?

```
predictions_1 = model_2.predict_generator(
```

Legacy-функция predict_generator эквивалентна predict.

> добавлена TTA (Test Time Augmentation) - работала для меня очень непоследовательно. Может быть, я не совсем понимаю, нужно дополнительно исследовать, к каких случаях нейросеть ошибается в предсказаниях - улучшило счет на сабмите только в одном случае, в остальных - стабильное падение

Здесь я бы посоветовал сделать TTA на валидации и сравнить точность без TTA и точность с TTA, в зависимости от кол-ва шагов. Если в TTA выполняются сильные аугментации, тогда нужно больше шагов.

> настройка параметров регуляризации полносвязного слоя нейронной сети - использовалась, полезность оценить не удалось - но позволяет избежать "переобучения" сети, что было видно в процессе обучения - с использованными аугментациями и за счет регуляризации сеть переобучалась только при очень длительных процессах обучения в поздней стадии

Не уверен, что параметры bias_regularizer=l2(1e-4),activity_regularizer=l2(1e-5) что-то меняют в лучшую сторону. Вы сравнивали их наличие и отсутствие?

> добавлена Batch Normalization в архитектуре “головы” модели

Добавлен, но зачем?

> использована SOTA архитектура сетей - Xception, EfficientNetB6 - в частности, как удобная альтернатива обучению сети с нуля

Все-таки это уже не SOTA, есть более новые модели, в частности из Tensorflow Hub.