```
! pip install git+https://github.com/mjkvaak/ImageDataAugmentor
```

В ImageDataGenerator есть параметр preprocessing_function, где тоже можно выполнить аугментацию через albumentations, поэтому в целом нет необходимости ставить дополнительную библиотеку ради этого. Хотя, с другой стороны, в ImageDataAugmentor есть полезный метод show_data.

```
A.OneOf([
        A.CenterCrop(height=224, width=200),
        A.CenterCrop(height=200, width=224)],
        p=0.5),
```

Если мы делаем случайные аугментации, то почему не использовать RandomCrop?

```
train_gen = ImageDataAugmentor(rescale=1./255,
```

В разных моделях нужна разная нормализация, но почти нигда она не равна делению на 255. Например, для Xception требуется нормализация tf.keras.applications.xception.preprocess_input – это не то же самое, что деление на 255. Правильная нормализация (то есть такая же, какая была при предобучении) может немного улучшить точность, хотя это не принципиальный момент.

```
test_datagen = train_gen.flow_from_directory(PATH+'train',
```

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, особенно если вы сравниваете разные модели и значения гиперпараметров.

Пример того, как делать валидацию без аугментаций, был приведен в бейзлайне.

```
!pip install -q efficientnet
import efficientnet.keras as efn
```

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

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

Эти слои в голову обычно не добавляют, но вы можете проверить – вдруг они повысят точность?

```
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.

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.25,
                              patience=2,
                              min_lr=0.0000001,
                              verbose=1,
                              mode='auto')
```

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. Остановку при этом можно делать по earlyStopping, как у вас и сделано.

Количество эпох можно было сделать больше, что улучшило бы точность. Хотя, с другой стороны, вы обучаете на большом размере изображений 512x512, и обучение может идти долго.

```
tta_steps = 10
predictions = []

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

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.