!pip install -q efficientnet

Эта библиотека уже не актуальна, поскольку EfficientNet есть в модуле keras.applications. Но конечно можно ее использовать, ошибки здесь нет.

* Распределение классов довольно равномерное, не требует доработок

Что понимается под доработками? А если бы было неравномерное, вы бы стили балансировать классы? Вообще если целевой метрикой является accuracy (а не F-мера например), то необходимость балансировать классы сомнительна. Повысит ли это метрику? Поизучал научные работы по этой теме, но ни в одной из работ по этому вопросу ничего не нашел. Везде говорят только что accuracy использовать не надо, но это уже другой вопрос. Я думаю что даже при балансировке классов accuracy не повысится.

for data\_zip **in** ['train.zip', 'test.zip']:

with zipfile.ZipFile("../input/"+data\_zip,"r") as z:

z.extractall(PATH)

Это можно было бы записать проще (работает только в IPython-средах):

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

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

A.OneOf([

A.CenterCrop(height=224, width=200),

A.CenterCrop(height=200, width=224)],

Если мы делаем аугментации, почему не использовать RandomCrop?

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.

*# кстати, ты заметил, что для сабмишена мы используем другой источник для генератора flow\_from\_dataframe?*

*# Как ты думаешь, почему?*

Звучит как притча о китайском мудреце:)

x = Dense(256, activation='relu')(x)

x = Dropout(0.25)(x)

x = Dense(256, activation='relu')(x) *#моя добавка*

Больше слоев не означает лучше. Если сделать в голове сети много полносвязных слоев, то она быстро переобучится.

for layer **in** base\_model.layers [:50]:

layer.trainable = False

for layer **in** base\_model.layers[-30:]:

*# батч норм должен настраивать свои параметры для новых данных!*

*а иначе фиксируем слой!*

if **not** isinstance(layer, BatchNormalization):

layer.trainable = False

Не понял идеи. Судя по коду, вы замораживаете *первые 50 и последние 30 слоев*, в чем смысл такого действия? И почему при этом слои бат-нормализации вы оставляете обучаемыми?

checkpoint = ModelCheckpoint('best\_model.hdf5' , monitor = ['val\_accuracy'] , verbose = 1 , mode = 'max')

Надо добавить save\_best\_only=True, иначе модель будет сохраняться на всех эпохах независимо от точности на валидации.

model.fit\_generator(

model.predict\_generator(

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

*# Сначала замораживаем все слои кроме новой "головы"*

*# Потом, когда мы научили последние слои (голову) под новую задачу, можно разморозить все слои и пройтись маленьким лернинг рейтом*

Да, так часто делают, но ваш код этому не соответствует. И к тому же любые утверждения в DL спорны. Ко всему стоит относиться с сомнением. Может быть обучая всю модель без постепенной разморозки слоев мы получим лучший результат? Тут надо либо проводить собственные эксприменты, либо читать научные статьи.

model.save('../working/model\_last.hdf5')

model.load\_weights('best\_model.hdf5')

Это одна и та же модель, т. к. вы не указали save\_best\_only=True