!pip install -q efficientnet

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

*# Распакуем картинки*

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

**with** zipfile.ZipFile('../input/sf-dl-car-classification/'+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?

A.Rotate(limit=30,

interpolation=1,

border\_mode=4,

value=**None**,

mask\_value=**None**,

always\_apply=**False**,

p=0.5),

A.ShiftScaleRotate(shift\_limit=0.0625,

scale\_limit=0.01,

interpolation=1,

border\_mode=4,

rotate\_limit=20,

p=.75),

У вас дважды делается поворот: сначала на +-30 градусов, потом еще на +-20 градусов. Получается слишком сильный поворот, который может негативно сказаться на качестве.

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.

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

layer.trainable = **False**

*# ВСТАВКА fine-tuning только для последних слоев*

**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**

Нашел у еще одного студента такой код, вы делали вместе? Но я понял идеи: судя по коду, вы замораживаете *первые 50 и последние 30 слоев*, в чем смысл такого действия? И почему при этом слои батч-нормализации вы оставляете обучаемыми?

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

x = Dropout(0.25)(x)

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

Два скрытых полносвязных слоя в голове могут привести к большему переобучению сети. Больше слоев не значит лучше, особенно если речь идет о полносвязных слоях и небольшом обучающем датасете.

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.

* подобраны переменные (размер картинки, батч, количество эпох)

Подбор означает, что сравнивается несколько вариантов. А у вас я вижу только один вариант. Если вы сравнивали несколько, то было бы неплохо занести результаты в таблицу.

* добавлена Batch Normalization

Но это ведь ничем не особновано. В керасе доступно 20-30 разных слоев для нейронной сети, если их все добавить в голову, станет ли результат лучше?

* В связи с багами kaggle не получилось довести до конца.

Жаль, но вы могли бы написать в канал в Slack, я мог бы вам помочь справиться с багами.

* Буду продолжать в colab. Продолжу работу, т.к. хочу разобраться глубже.

Это здорово! Я год назад был студентом SF, и решал это же соревнование. После сдачи еще месяца два проводил эксперименты и повышал точность, и не жалею, это помогло разобраться в теме.