print('Распаковываем картинки')

**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}

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.

Указываем, что сеть не обучается.

base\_model.trainable = **False**

Иногда говорят, что сеть при файн-тюнинге нужно размораживать постепенно, чтобы «хорошо обученные признаки на Imagenet не затирались в самом начале нашего обучения». Но нейронные сети несколько сложнее, чем часто может показаться. Подобные рассуждения выглядят логично, но часто оказываются неверны на практике. Я пробовал сравнивать послойный файн-тюнинг и файн-тюнинг сразу целиком, и второй вариант давал лучшую точность. Но конечно я провел недостаточно много экспериментов чтобы утверждать это наверняка. Пишу это к тому, что в DL надо во всем сомневаться.

То же самое насчет архитектуры головы. Почти все учасники данного соревнования добавляют в голову полносвязный слой и batch-norm, как и у вас:

model.add(L.GlobalAveragePooling2D())

model.add(L.Dense(256, activation='relu'))

model.add(L.BatchNormalization())

model.add(L.Dropout(0.25))

model.add(L.Dense(CLASS\_NUM, activation='softmax'))

Никто не проверяет – а дает ли это прирост точности по сравнению с головой, состоящей только из GlobalMaxPool2D и выходного слоя? Болше слоев не значит лучше. Если посмотреть на научные работы по сверточным сетям, то там практически всегда делают голову только из GlobalMaxPool2D и выходного слоя. Значит это хорошо работает при обучении на ImageNet. Но с другой стороны это еще не гарантирует, что это наилучший выбор при файн-тюнинге.

model.add(L.Dense(CLASS\_NUM, activation='softmax'))

loss = keras.losses.CategoricalCrossentropy(from\_logits = **True**)

Здесь ошибка: если вы указываете from\_logits=True, то это значит, что softmax расчитывается внутри функции потерь, и в выходном слое он не нужен. А так у вас получается двойной softmax. Скорее всего это приведет к ухудшению результата обучения.

model.evaluate\_generator

model.fit\_generator

Эти методы являются устаревшими, сейчас они эквивалентны model.evaluate и model.fit.

* Когда запускала ноутбук первый раз Сеть Xception показала accuracy 15%, сеть ResNet50V2 - 6%, сеть EfficientNetB6 - 12%.

А зачем запускать evaluate на сетях, которые не файн-тюнились под задачу. Разве это будет информативно?

* Когда запускала ноутбук первый раз Сеть Xception показала accuracy 15%, сеть ResNet50V2 - 6%, сеть EfficientNetB6 - 12%. По итогам, будем работать с сетью Xception

Эти цифры означают точность сетей, которые не обучались вообще под задачу? Допустим вы создали 3 разные сети и получили 3 разных результата (15%, 6%, 12%). Но если бы вы все 3 раза использовали одну и ту же сеть (например каждый раз Xception), то тоже получили бы 3 совершенно разных результата. Модель нужно сначала обучать под задачу, а потом тестировать, а не наоборот.

predictions = L.Dense(CLASS\_NUM, activation='softmax')(x)

*# this is the model we will train*

model = Model(inputs=base\_model.input, outputs=predictions)

model.compile(loss="categorical\_crossentropy"

Здесь у вас уже нет двойной softmax, это правильно.

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

Здесь надо добавить save\_best\_only=True, иначе модель будет сохраняться каждую эпоху независимо от точности.

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

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

Это одна и та же модель, потому что в ModelCheckpoint вы не указали save\_best\_only=True.

ImageDataGenerator(rescale = 1/255

Вообще для каждой сети (Xception, EfficientNet и др.) нужей свой rescaling входных данных. Например, EfficientNet принимает изображение со значениями пикселей от 0 до 255. Если подавать значения от 0 до 1, то это будет отличаться от того, что использовалось при обучении этой сети на ImageNet, что может сказаться на качестве.

tta\_steps = 10 *# берем среднее из 10 предсказаний*

predictions = []

**for** i **in** range(tta\_steps):

preds = model.predict\_generator(test\_sub\_generator,

steps=len(test\_sub\_generator), verbose=1)

predictions.append(preds)

Я не очень понимаю что вы сдесь делаете. Если вы делаете предсказания на каждом тестовом изображении 10 раз, то нужно делать их с аугментациями, иначе все 10 предсказаний в точности совпадут. А у вас test\_sub\_generator генерирует изображения без аугментаций.

* В результате экспериментов, Xception показала себя с наилучшей стороны, показав наибольшее значение метрики accuracy по сравнению с сетями ResNet50V2 и EfficientNetB6.

Вы сравнивали эти сети до их обучения, разве есть смысл в таком сравнении? Представьте, что вы отбираете первоклашек в свой класс, у ва конкурс 5 человек на место, и для этого дали первоклассникам решать ЕГЭ. Они конечно не знают материала ЕГЭ и просто поставят галочки случайно. Потом вы выбираете из них тех, кто набрал больше баллов. Разумен ли такой выбор?