- Основная идея этого решения: взять предобученую на ImageNet сеть Xception и дообучить под нашу задачу

Это идея бейзлайна, но бейзлайн – на то и базовое решение, что оно не обязательно оптимально. Так что не обязательно брать именно Xception. Кстати бейзлайн является устаравшим, с тех пор библиотека keras изменилась. Есть более новый бейзлайн, к сожалению пока нет ссылки на платформе на него, скоро появится.

```
!pip install efficientnet
```

Сейчас есть более легкий способ использовать EfficientNet, так как эта сеть есть в модуле keras.applications. В более ранних версиях ее не было, приходилось устанавливать отдельный пакет.

```
for data_zip in ['train.zip', 'test.zip']:
    with zipfile.ZipFile("../input/"+data_zip,"r") as z:
        z.extractall(PATH)
```

Как вариант, это можно было бы записать проще (работает только в IPython-средах):

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

```
albumentations.OneOf([
        albumentations.CenterCrop(height=224,
                                  width=200),
        albumentations.CenterCrop(height=200,
                                  width=224),
```

Аугментации предназначены чтобы срздавать разнообразные изображения, поэтому почему бы не использовать RandomCrop вместо CenterCrop?

```
rescale=1./255,
```

Модель вероятно будет работать лучше, если нормализация изображений при файн-тюнинге будет выполняться так же, как и при обучении. Поэтому надо знать с какой нормализацией модель обучалась, а это бывает по-разному, поэтому по каждой модели нужно читать документацию. Например, для сети из пакета efficientnet.keras есть своя функция нормализации, пример здесь.

```
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.

```
efn.EfficientNetB7
```

Выбирать самую большую модель – не обязательно самый хороший вариант. Во-первых она намного дольше обучается (шаг оптимизации занимает больше времени), во-вторых глубокие модели сильнее переобучаются (а на файн-тюнинге тоже можно переобучиться), глубоким моделям нужно больше обучающих примеров, так что не исключено, что более маленькая модель могла бы показать результат лучше.

```
model.add(layers.Dense(256, activation='relu'))
model.add(layers.BatchNormalization())
```

Опять таки не факт, что это оптимальный вариант, но спорить не буду, т. к. наверняка сказать не могу. Было бы полезно сравнить графики обучения двух моделей, у одной из которых есть скрытый полносвязный слой в «голове», а у другой нет. Возможно у второй точность была бы выше.

```
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.

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

Здесь можно было бы увеличить patience, потому что точность модели хоть и имеет тенденцию расти с каждой эпохой, но все же может случайно колебаться, поэтому не стоит уменьшать LR раньше времени.

```
albumentations.Rotate(limit=30,
```

Большой угол поворота в TTA, а чем сильнее сильнгее аугментации, тем сильнее они ухудшают точность. Поэтому чем сильнее аугментации, тем больше итераций TTA требуется, в противном случае TTA может наоборот ухудшить точность.

- подобраны переменные (размер картинки, батч и т.д.)

Подбор означает сравнение разных вариантов, а у вас в ноутбуке используется только один вариант размера картинки и батча. Если вы все-таки сравнивали разные варианты, то было бы хорошо написать об этом, занести результаты сравнения в таблицу.

- SOTA архитектура сетей - EfficientNetB7

Не сказать что это SOTA, уже есть более новые и эффективные модели, такие как efficientnet_v2, RegNetY, BiT-transfer:

https://tfhub.dev/google/collections/efficientnet_v2/1
https://tfhub.dev/adityakane2001/collections/regnety/1
https://tfhub.dev/google/collections/bit/1

- добавлена Batch Normalization

Представьте, что вам на работе дали задание обучить сеть для классификации – и вы показываете как результат своей работы данный ноутбук. Вас спрашивают: а зачем в голову добавлена батч-нормализация? Как обосновать выбор? На мой взгляд нужно все-таки как-то обосновывать, основываясь либо на статьях из интернета, либо на экспериментальном сравнении. Хотя не стоит всему верить что пишут по нейронным сетям в интернете. Даже в научных статьях иногда ошибаются, а в профессиональной среде в Deep learning есть огромное число различных «верований» о том, что какие-то подходы лучше других. Далеко не все из этого при проверке оказывается верно. Причина в том, что Deep learning – сложная область в том плане, что все сильно зависит от задачи, архитектуры и множества мелочей. Если батч-нормализация и

полносвязный скрытый слой, добавленные в голову сети, в каком-то эксперименте дали прирост точности, то вовсе не факт что они будут давать прирост точности всегда.

- TTA (Test Time Augmentation)

Было бы хорошо сравнить точность с TTA и без TTA, а то даже не понятно улучшилась ли она и насколько.