```
PYTHONHASHSEED = 0
```

Эта переменная нигде не используется.

```
print('Распаковываем картинки')
for data_zip in ['train.zip', 'test.zip']:
    with zipfile.ZipFile(DATA_PATH+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}

- Кол-во фото в трейне может не хватить для хорошего обучения модели нейронной сети, поэтому будем применять различные виды аугментации данных

Вообще аугментации применяют даже тогда, когда доступно очень много изображений.

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

Если мы делаем аугментации, то 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.

- Также надо быть внимательнее с командой "from tensorflow.keras.applications import EfficientNetB7", при этом модель обучается не корректно и не позволяет достичь хороших показателей. Нужно использовать устаревшие методы, такие как "import efficientnet.tfkeras as efn" (с последним выводом возможно я ошибаюсь, не было достаточно времени удостовериться в этом на 100%).

Может быть проблема в том, что вы делили значения пикселей на 255?

```
train_datagen = ImageDataAugmentor(
        rescale=1/255,
```

Дело в том, что EfficientNetB7 из модуля keras.applications принимает на вход значения пикселей от 0 до 255, а нормализация делается уже внутри модели.

```
model.add(base_model)
model.add(Layer.GlobalAveragePooling2D())
model.add(Layer.Dense(256,
                      activation='relu'))
model.add(Layer.BatchNormalization())
model.add(Layer.Dropout(0.25))
model.add(Layer.Dense(CLASS_NUM, activation='softmax'))
```

Почти все, кто выполняет этот проект, добавляют скрытый полносвязный слой и слой batchNorm в голову сети. И не было еще почти ни одной работы, где сравнивалась бы голова со скрытыми слоями и BN и без них. Вдруг без них лучше?

```
checkpoint = 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.

```
learning_rate=ExponentialDecay(
                  0.0009, decay_steps=100, decay_rate=0.9)
BATCH_SIZE           = 8
```

Обратите внимание на эти параметры. С таким маленьким размером батча эпоха занимает 1849 шагов, значит за 1 эпоху learning rate умножается на 0.9^18.49 = 0.14. Если он был 9e-4, то станет примерно 1e-4 после первой эпохи и примерно 2e-5 после второй, и так далее. К третьей эпохе он уменьшится настолько, что сеть перестанет обучаться.

```
EPOCHS = 4 # эксперименты показали, что такого количества эпох будет достаточно
```

Возможно именно из-за того, что LR у вас уменьшается слишком быстро, и эпохи после 4-й уже не оказывают никакого влияния. Вообще на этом датасете сети можно обучать десятки эпох при более правильно выбранных параметрах.

```
predictions_tta = []
for _ in range(EPOCHS):
    predictions_tta.append(model.predict(sub_generator, verbose=1))
    sub_generator.reset()
```

В sub_generator у вас из аугментаций только сдвиг и horizontal_flip. Сдвиг оказывает мало влияния на функционирование сверточных сетей, а у horizontal_flip всего 2 возможных состояния. Стоило добавить больше аугментаций в sub_generator. Но по крайней мере у вас они есть. Многие начинают делать TTA вообще без аугментаций в sub_generator и после этого делает вывод, что TTA бесполезен и не дает прироста точности.

Можно еще делать так: сделать сначала 10 попыток на каждом изображении, а затем делать дополнительные попытки на тех изображениях, в которых результат остался спорным.

- Экспериментальным путем выведено,что наиболее эффективной предобученной моделью является EfficientNet начиная с версии 3 и выше. Мной использовалась третья и седьмая модель.

EfficientNet хорошая модель, но сейчас есть и более новые и эффективные модели, например: https://colab.research.google.com/github/google/automl/blob/master/efficientnetv2/tfhub.ipynb

- В другом варианте применялся метод ExponentialDecay для регулировки learning rate.

Хорошо, что он применялся, но его настройки были выбраны так, что LR практически занулялся уже после 3 эпох.

- Архитектуры "голов" сознательно применялись разные.

Было бы неплохо увидеть таблицу сравнения разных голов и результатов обучения. Потому что по вашему коду не видно, что тестировались разные результаты. В ноутбуке только 2 варианта голов,

но при этом они применялись с разными архитектурами и поэтому результаты нельзя сравнивать напрямую.

- Можно сделать вывод, что модель с седьмой версией EfficientNetB7 (95,92%) существенно обгонит третью, если изменить показатель image_size с 224 до 512.

Получается, что вы сделали сразу 2 изменения: изменили модель и изменили разрешение. Тогда непонятно что дало прирост: изменение разрешения или изменение модели.