```
EPOCHS = 5  # it was defined empirically that 5 epochs is enough for learning
```

Странно что у вас так получилось, 5 эпох почти всегда недостаточно для достижения максимальной точности.

У вас в ноутбуке отсутствуют выводы ячеек, поэтому вся визуализация, которую вы делаете в разделе EDA, в итоге не видна.

```
random_image = train_df.sample(n=9)
random_image_paths = random_image['Id'].values
random_image_cat = random_image['Category'].values

for index, path in enumerate(random_image_paths):
    im = PIL.Image.open(PATH+f'train/{random_image_cat[index]}/{path}')
    plt.subplot(3,3, index+1)
    plt.imshow(im)
    plt.title('Class: '+str(random_image_cat[index]))
    plt.axis('off')
plt.show()
```

В качестве альтернативы, можно было написать это таким кодом:

```
samples = train_df.sample(frac=1)

fig, axes = plt.subplots(2, 8, figsize=(15, 5))
for ax, (row_idx, row) in zip(axes.flat, samples.iterrows()):
      path, category = row['Id'], row['Category']
      img = PIL.Image.open(PATH+f'train/{random_image_cat[index]}/{path}')
      ax.imshow(img)
      ax.set_title(f'Class: {category}')
      ax.axis('off')
plt.show()
```

В функции show_first_images:

```
generator = itertools.islice(generator, count)
fig, axes = plt.subplots(nrows=1, ncols=count, figsize=figsize)
for batch, ax in zip(generator, axes):
```

Этот код можно упростить, убрав первую строку, она ни на что не влияет. Наверное обновлю бейзлайн, потому что первая строка лишняя.

```
# training a model with 8 Epochs model.fit(train_generator2,
validation_data=val_generator, epochs=EPOCHS, callbacks=[history])
```

Но параметр EPOCHS в ноутбуке равен 5, почему написано 8 эпох?

Самое главное – не видно ни логов, ни графиков обучения, потому что отсутствуют выводы ячеек. Возможно посмотрев на график стало бы понятно, что точность еще имеет потенциал к росту и стоит продолжать далее.

- I must mention that I tried up to 10 different augmentations with Albumentation library, both from best practices and from my own experiments. Neither of them worked well.

Да, дополнительные аугментации часто не дают прироста точности. Например, в [туториале](#) от специалистов Google по файн-тюнингу BiT-transfer вообще используется только random crop и random horizontal flip (см. раздел «BiT Hyper-Rule»).

```
A.OneOf([
            A.CenterCrop(height=224, width=200),
            A.CenterCrop(height=200, width=224),
        ],p=0.5),
```

Если мы делаем аугментации, то логичнее делать random crop.

```
# training a model
model.fit(album_generator, validation_data=val_generator, epochs=1,
callbacks=[history])
```

- In general, Albumentation library not only failed to improve the metrics, but it even didn't show any good results. My assumption is, that it is because of the specifics of the images.

Почему тестируя с albumentations вы обучаете всего одну эпоху? На 1 эпохе от аугментаций будет только падение точности, так как они в целом усложняю задачу обучения для сети. По аналогии с dropout: если поставить его равным 0, то точность после первой эпохи будет выше, но после N-й эпохи станет ниже, чем с dropout > 0.

```
optimizer=optimizers.Adam(learning_rate=ExponentialDecay(
                  0.0009, decay_steps=100, decay_rate=0.9)),
```

Можно попробовать ReduceLROnPlateau, часто это работает лучше, чем Exponential decay. У Exponential decay есть проблема: если выбрать слишком высокую скорость затухания, то LR быстро приблизится к 0, и сеть перестанет обучаться.

```
sub_datagen_enb3 = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the
dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range = 30,  # randomly rotate images in the range (degrees,
0 to 180)
    zoom_range = 0.2, # Randomly zoom image
    width_shift_range=0.1,  # randomly shift images horizontally (fraction
of total width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction
of total height)
    horizontal_flip = True,  # randomly flip images
    vertical_flip=False  # randomly flip images
```

В TTA у вас сильные аугментации, поэтому может потребоваться много шагов TTA, в противном случае точность может только ухудшиться.

Можно было сделать TTA сначала на валидационном датасете и проверить, повышает ли он точность, то есть сравнить точность:

1. Предсказаний без TTA
2. Предсказаний на каждом шаге TTA
3. Усредненных предсказаний

- Let's try another model - EfficientNetB7

Можно было сразу поискать одну из новых моделей, например

import tensorflow_hub as hub
model =
hub.KerasLayer("https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_b0/feature_vector/2")

- Let's try another base model, from different family – Xception

Здесь надо помнить, что этой модели, в отличие от EfficientNet, нужна нормализация входных изображений методом tf.keras.applications.xception.preprocess_input. Его можно встроить как первый слой модели:

keras.layers.Lambda(tf.keras.applications.xception.preprocess_input)

```
predictions_ens = 0.4*model_enb3.predict(sub_generator_enb3) +
0.3*model_enb7.predict(sub_generator_enb7) +
0.3*model_x.predict(sub_generator_x)
```

Подбирая коэффициенты действительно можно добиться прироста точности. Кстати, сумма коэффициентов не обязана быть равна 1, так как далее выполняется argmax.

- Unfortunately Albumentations library didn't show any good results

По-моему не совсем корректная формулировка, выбранные аугментации не дали результата, не библиотека.

В целом хорошая объемная работа, но стоило использовать большее количество эпох (и меньшую скорость затухания learning rate), тогда точность могла бы быть еще выше. И конечно размер изображений определяющим образом влияет на результат.