```
!pip install git+https: // github.com/mjkvaak/ImageDataAugmentor
```

Библиотека ImageDataAugmentor, на мой взгляд, имеет мало смысла, ведь в обычном ImageDataGenerator есть параметр preprocessing_func, куда можно передать функцию из albumentations.

> В качестве исследуемых моделей выбраны Xception, InceptionV3, ResNet101V2 и EfficientNetB3. Все выборы сделаны исходя и Leaderboard'a, представленного на сайте https://paperswithcode.com/.

Это если выбирать только из тех моделей, что есть в keras.applications. Например, в tensorflow hub есть много и более новых моделей.

> Эти два алгоритма являются более чувствительными к весам исследуемой модели.

Не совсем понял что имеется в виду

```
train_datagen = ImageDataGenerator(
    rescale=1. / 255, # т.к мы используем модели без встроенного rescale
```

В разных моделях нужна разная нормализация, но почти нигда она не равна делению на 255. Например, для Xception требуется нормализация tf.keras.applications.xception.preprocess_input – это не то же самое, что деление на 255. Правильная нормализация (то есть такая же, какая была при предобучении) может немного улучшить точность, хотя это не принципиальный момент.

```
horizontal_flip=False, # переворот относительно горизонтали
```

Почему False? По идее машины симметричны, значит можно делать flip.

```
# Аугментации валидационной выборки
val_generator = train_datagen.flow_from_directory(
```

А зачем делать аугментации на валидационной выборке?

Обычно learning rate снижают в ходе обучения. Можно использовать либо ReduceLROnPlateau, либо keras.optimizers.schedules.

```
model.evaluate_generator(
```

Этот legacy-метод сейчас эквивалентент .evaluate().

> Мы получили достаточно хорошие метрики, хотя похоже, что модель начала переобучаться. На графике точности четко виден пик на 4 эпохе, после которого начинается ее снижение и образуется низина вплоть до 7 эпохи, на которой снова происходит рост метрики.

Модель всегда в той или иной степени переобучается, научаясь получать ответ ориентируясь на те детали, которые вообще говоря не важны (например, на фон). Более подробно я об этом писал здесь. Но это не всегда повод останавливать обучения, потому что, как вы заметили, дальше точность может продолжить расти.

Если вы сравниваете несколько моделей, то как вариант можно было создать функцию для компиляции и обучения и вызывать ее на разных моделях без дублирования кода.

> На графиках функции потерь и точности видно, что модель в промежутке после 4 и до 7 эпохи имела плохие показатели метрик, хуже чем на самой 4 эпохе. К 7 эпохе показатели снова улучшились, однако затем произошел очередной спад точности и рост функции потерь. Можно сделать вывод, что модель начала переобучаться.

Наверное такие несущественные колебания на графике можно просто считать случайным шумом.

> Предыдущие модели принимали диапазон от -1 до 1, поэтому для них проводились необходимые преобразования.

Получается несоответствие: вы пишете, что диапазон от -1 до 1, но при этом делите на 255, получая диапазон от 0 до 1.

> Для дальнейшей работы необходимо переделать ImageDataGenerator, убрав из него rescale. Остальные параметры остаются без изменений.

Чтобы не создавать два раза генераторы, можно встраивать нормализацию в модель, например добавляя такой слой:

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

Или так, как вы сделали далее (но это работает немного по-другому):

```
Lambda(lambda x: x/255),
```

Когда вы экспериментируете с разными моделями, вы используете одну и ту же голову:

```
model_resnet = Sequential([
    base_model,
    head
])
```

Здесь head та же самая, что использовалась в предыдущей модели, то есть с теми же матрицами весов. Это достаточно странно. Если новая модель возвращает тензор другого размера, то будет ошибка, как у вас случалось с efficientNet.

Дело в том, что создав head, вы не просто описали архитектуру, а создали конкретные слои с конкретными весами.

> Для улучшения исследуемых моделей попробуем воспользоваться другим аугментором (ImageDataAugmentor), использующим albuminations.

Albumentations можно использовать и в ImageDataGenerator с помощью параметра preprocessing_func.

```
learning_rate=ExponentialDecay(LR, decay_steps=100, decay_rate=0.95))
```

Каждая эпоха занимает 779 шагов, так что LR умножается на 0.95^7.79 = 0.67 после каждой эпохи. Это может привести к тому, что после 5-10 эпох LR станет настолько малым, что сеть перестанет обучаться.

ResNet101V2 без ImageDataAugmentor'a:74.59%

ResNet101V2 с ImageDataAugmentor'ом:90.02%

Может быть здесь проблема в том, что вы используете голову от предыдущей модели, а не создаете ее заново.

Заметил, что вы не делаете TTA, это могло бы повысить точность.

В целом хорошая, подробная работа!