```
# Загружаем обвязку под keras для использования продвинутых библиотек
аугментации, например, albuminations
!pip install git+https://github.com/mjkvaak/ImageDataAugmentor update
tensorflow
```

В ImageDataGenerator есть параметр preprocessing_function, где тоже можно выполнить аугментацию через albumentations, поэтому в целом нет необходимости ставить дополнительную библиотеку ради этого.

```
# Загрузка модели efficientnet
!pip install -q efficientnet
```

В новых версиях Keras модель EfficientNet есть в модуле keras.applications.

```
H_IMG_SIZE           = 420 # Горизонтальный размер рисунка
V_IMG_SIZE           = 320  # Вертикалбный размер рисунка
```

Как правило за H обозначается высота (height), за W ширина (width). У вас наоробот за H обозначается ширина, согласно комментариям к коду, что странно. Хотя дальше вы используете эти переменные так:

```
target_size=(V_IMG_SIZE, H_IMG_SIZE),
```

В этом параметре сначала идет высота, потом ширина. То есть у вас 320 – высота, 420 – ширина, хотя в комментарии написано наоборот.

```
# добавляем размытие по Гауссу и шум с вероятностью 50%
a.Blur(p=0.05),
a.GaussNoise(p=0.05),
```

Но p=0.05 – это 5%, а не 50%

```
datagen = ImageDataAugmentor(
                        rescale=1./255,
```

Для разных сетей нужна разная нормализация, и как правило это не деление на 255. Вряд ли это сильно повлияет на точность классификации, но немного повлиять может.

> Проверка генерированных данных train_generator, test_generator

В ноутбуке пропали выводы ячеек, так что не могу посмотреть примеры. Но это совершенно правильно, что вы делаете визуализацию изображений с аугментациями, поскольку так можно оценить адекватность аугментаций и отследить возможные ошибки.

```
# Add a fully-connected layer
model.add(Dense(256,
                      activation='relu',
                      bias_regularizer=l2(1e-4),
                      activity_regularizer=l2(1e-5)))

# Add batch normalization
model.add(BatchNormalization())
model.add(Dropout(0.25)) # and dropout
```

А точно эти слои нужны? Обычно их не добавляют.

```
# Add функцию контрольной точки, чтобы сохранить лучшую модель
checkpoint = ModelCheckpoint('best_model.hdf5',
                             monitor = ['val_accuracy'],
                             verbose = 1,
                             mode = 'max')
```

Здесь нужно добавить параметр save_best_only=True

```
# Добавим lr_scheduler (экспоненциально уменьшать скорость через 2 эпохи)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss',
                                 factor=0.2, # уменьшим lr в 5 раз
                                 patience=2, # если нет улучшения через 2 эпохи
- уменьшить lr

                                 min_lr=0.0000001,
                                 verbose=1,
                                 mode='auto')
```

У вас параметр patience равен 2. Представьте, что вы тренируете спортсмена, и прекращаете тренировки если в течение 2 тренировок подряд его результаты не растут. В итоге спортсмен мог бы тренироваться годами, а вы прекратите тренировать его уже через неделю. Но это ведь неправильно: может быть много случайностей, которые влияют на результат в какой-то из дней. Так же и с нейросетями: график точности подвержен случайным колебаниям, поэтому patience лучше делать существенно выше. Остановку при этом можно делать по earlyStopping, как у вас и сделано.

```
model.evaluate_generator(
```

Этот метод устарел и сейчас по сути эквивалентен evaluate, а в будущем возможно будет удален.

```
label_map = (train_datagen.class_indices)
label_map = dict((v,k) for k,v in label_map.items()) #flip k,v
predictions = [label_map[k] for k in predictions]
```

Мне кажется что конкретно на данном датасете смысла в этом действии нет, ведь label_map будет содержать записи такие как 1 -> «1». То есть удалив эти две строки, вы получите тот же результат в файле submission.csv.

```
test_datagen = datagen.flow_from_directory(
    PATH+'train/',
```

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.

> Конец Step 1
Step 2

Вы несколько раз создаете генераторы для каждого шага, потому что меняется размер изображения. Можно было бы сделать функцию для создания генератора, и передавать в нее размер изображения. Так код был бы лаконичнее, и легче было бы вносить изменения.

> Эксперимент проводился на двух моделях: функциональной и последовательной.

В коде увидел только последовательную модель, но вообще выбор между функциональной и последовательной моделью не принципиален. Это просто два способа построения, но результат (граф вычислений) получится один и тот же.

> Точность модели составила около 39% при image_shape = (90,120);

При таком разрешении точность должна получаться около 85-95%, если у вас 39%, значит либо где-то ошибка, либо вы не разморозили модель целиком, а обучали только голову.

> с базовой моделью EfficientNetB5 точность предсказания достигли 94.93%. Наблюдается не устойччивость точности предсказания.

Было бы нагляднее представить результаты в таком виде: структура модели – параметры обучения – график точности.

В целом хороший результат, хоть и есть однако несколько типичных проблем, которые встречаются почти у всех, но точность достигнута лучше большинства. Всегда приятно видеть сабмит с точностью 97% и выше :)