```
fig,ax = plt.subplots(2,1,figsize=(10,10))
    ax1,ax2 = ax.flatten()
```

Как альтернатива, можно писать так:
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,10))

```
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.

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

Если мы делаем аугментации, то наверное логичнее использовать RandomCrop.

```
A.RandomBrightness(limit=0.2, p=0.5),
A.OneOf([
        A.RandomBrightnessContrast(brightness_limit=0.3,
                                                contrast_limit=0.3),
        A.RandomBrightnessContrast(brightness_limit=0.1,
                                                contrast_limit=0.1)],
        p=0.5),
```

Два раза повторяется изменения яркости.

```
# Инициализируем генератор для валидации
test_datagen = train_gen.flow_from_directory(DATA_PATH+'train',
```

Валидация (test_datagen) делается также 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.

```
model.add(Layer.Dense(256,
                      activation='relu',
                      bias_regularizer=l2(1e-4),
                      activity_regularizer=l2(1e-5)))
```

Непонятно из каких соображения добавлены регуляризаторы. Они не всегда увеличивают точность, часто наоборот уменьшают.

In [26]:
```
model= Sequential() # Model.Sequential()
model.add(base_model)
model.add(Layer.GlobalAveragePooling2D())
model.add(Layer.Dense(256,
                      activation='relu',
                      bias_regularizer=l2(1e-4),
                      activity_regularizer=l2(1e-5)))
model.add(Layer.BatchNormalization())
model.add(Layer.Dropout(0.25))
model.add(Layer.Dense(CLASS_NUM, activation='softmax'))
```

In [19]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model (Functional)           (None, 1536)              161130204
_________________________________________________________________
dense (Dense)                (None, 10)                15370
=================================================================
Total params: 161,145,574
Trainable params: 15,370
Non-trainable params: 161,130,204
_________________________________________________________________
```

In [20]:
```
# Check the trainable status of the individual layers
for layer in model.layers:
    print(layer, layer.trainable)
```

```
<tensorflow.python.keras.engine.functional.Functional object at 0x0000017EA97DD3D0> False
<tensorflow.python.keras.layers.core.Dense object at 0x0000017EC7681DC0> True
```

Между этими ячейками кода несоответствие. В первой ячейке создается модель из 6 слоев, а в следующей оказывается, что в модели 2 слоя. И по номерам ячеек видно, что они выполнялись не в том порядке. Это запутывает.

```
# Добавим ModelCheckpoint.
# Эта функция позволяет сохранять прогресс обучения модели,
# чтобы в нужный момент можно было его подгрузить и дообучить модель.
checkpoint = ModelCheckpoint('models/best_model.hdf5',
                             monitor = ['val_accuracy'],
                             verbose = 1,
                             mode = 'max')
```

Без параметра save_best_only=True параметры monitor, mode не имеют смысла, модель сохраняется после каждого шага.

Не нашел, где у вас в коде делается нормализация, специфичная для каждой модели. Обучая модель с неправильной нормализацией, можно получить заниженную точность.

Предсказание после fine-tuning

```
test_gen = ImageDataAugmentor(rescale=1./255)
```

По-моему эта нормализация не подходит ни для одной из моделей, которые вы используете. Например, для EfficientNet не нужна нормализация, а для Xception нужна нормализация tf.keras.applications.xception.preprocess_input, которая выполняется по-другому.

```
EPOCHS                = 6 # эпох на обучение (было 8)
BATCH_SIZE            = 1 # 4(был 8)
LR                    = 1e-5 # (был 1e-3)
```

А вы уверены, что с таким маленьким learning rate 6 эпох будет достаточно? И кроме того сеть с батч-ноормализацией не будет обучаться с batch_size=1.

```
test_gen = ImageDataAugmentor(#rescale=1./255,
                          augment=AUGMENTATIONS,
                          seed=RANDOM_SEED,
                          validation_split=VAL_SPLIT
                         )
```

Зачем в генераторе изображений для сабмита указан validation_split?

```
test_datagen = train_gen.flow_from_directory(DATA_PATH+'/train',
                                             class_mode='categorical',
                                             batch_size=BATCH_SIZE,
                                             target_size=(IMG_SIZE,
IMG_SIZE),
                                             shuffle=True,
                                             subset='validation'
                                            )
```

У вас создается 2 генератора: test_datagen берет изображения из подмножества обучающих изображений, а test_sub_generator из изображений для сабмита. При этом названия они имеют почти одинаковые, со словом «test». Так можно запутаться.

- настройка параметров регуляризации полносвязного слоя нейронной сети подобраны переменные (размер картинки, батч, количество эпох)

Собственно настройки в ноутбуке нет, есть лишь использование одного значения. Если вы делали подбор оптимального значения, то было бы хорошо привести таблицу результатов – какая получается точность с разным значением регуляризации. Здесь также надо помнить о том, что в точности на валидации есть элемент случайности.

- применен способ заполнения пропусков с помощью ImageDataAugmentor

Какие пропуски имеются в виду?

- добавлена Batch Normalization в архитектуре “головы” модели

Добавлена, но зачем?)

- SOTA архитектура сетей - Xception, InceptionV3, EfficientNetB5

Все-таки это уже не SOTA, но все еще широко используемые сети.