```
# create model
        self.model = Sequential()
        if rescale:
            self.model.add(InputLayer(input_shape=input_shape))
            self.model.add(Rescaling(scale=1./255))
```

Для каждой модели нужна своя нормализация, и как правило это не 1/255, например для Xception надо использовать tf.keras.applications.xception.preprocess_input

```
self.model.add(Dense(512, activation='relu'))
        self.model.add(BatchNormalization())
```

Как правило в современных сверточных архитектурах и при файн-тюнинге не добавляют скрытый полносвязный слой и BatchNorm. Можно конечно так делать, но не факт что это даст прирост качества. Скорее наоборот это даст большее переобучение, по крайней мере я на практике такое наблюдал. Больше слоев не всегда значит лучше.

```
ModelCheckpoint(filename, monitor = ['val_accuracy'], verbose=1, mode='max')
```

Здесь нужен параметр save_best_only=True, иначе будет сохранять каждую эпоху независимо от точности.

```
plt.plot(epochs, acc, 'b', label='Training acc')
…
plt.show()
```

Отрисовку можно выделить в отдельную функцию, так код будет лучше поделен на смысловые части.

```
# тестирование, результат и графики обучения/тестирования
        scores = self.model.evaluate(self.test_generator,
steps=len(self.test_generator), verbose=1)
```

А зачем дополнительно делать evaluate после обучения, если он и так делается в fit() после каждой эпохи? Например, scores[0] будет равен max(history.history['val_accuracy']). Точнее, был бы равен, если бы в ModelCheckpoint был параметр save_best_only=True, в противном случае best_model.hdf5 равен model_last.hdf5.

Кстати, можно было бы попробовать больший patience в ReduceLROnPlateau и большее количество эпох (или завершение по EarlyStopping). Когда вы заново запускаете md_xception.train_and_test, точность продолжает расти. Это значит, что patience=3 по-видимому слишком мал.

> рейтиг для данного сабмишна 0.92644, что немного ниже, чем при первоначальной поптыке обучить нейросеть с нуля. модель переобучилась, но что было сделано не так - непонятно.

Не знаю, но у меня получалось так же: при постепенной разморозке слоев точность ниже.

> Из всех доступных на сегодня СНС я выбрал для пробы ResNet50V2 и EfficientNet-B7 по нескольким причинам:
высокий рейтинг
относительно небольшое количество параметров ("всего" 60+ миллионов)
имплементация на tensorflow

Можно было еще посмотреть модели на tensorflow hub, там много более новых моделей.

```
md_effnet7 = ModelSet(bm_effnet7, rescale=False)
```

Здесь совершенно верно, для EfficientNet из keras.applications не нужна нормализация, так как она уже встроена в модель.

У вас не делается визуализация изображений с аугмнетациями. Ее полезно делать, потому что так можно отследить ошибки.

```
#return np.array(all_predictions)
    all_predictions = np.array(all_predictions)
    plt.figure(figsize=(15, 3))
    plt.imshow(all_predictions.argmax(axis=-1)[:, :50], cmap='nipy_spectral')
    plt.xlabel('Номер изображения')
    plt.ylabel('Номер попытки')
    plt.show()
```

Очень хорошо что вы делаете визуализацию предсказаний с TTA, вот только в ноутбуке ее не видно, почему-то вместо всех изображений белые квадраты.

```
md_list = [md_effnet7, md_resnet, md_xception]
md_coef = [0.96764, 0.95505, 0.94831]
```

Странно что в качестве коэффициентов берется точность модели. Если первая модель имеет точность 99.9%, а вторая 98%, то ясно что первую надо брать с намного большим коэффициентом. А так коэффициенты получатся почти одинаковые.

```
md_coef = md_coef / np.sum(md_coef)
```

Делить на np.sum(md_coef) не обязательно, так как далее идет операция argmax. Результат будет тот же при делении на любое положительное число.

> Полученный score 0.96704, результат не стал лучше

Вообще странно что не стал, возможно как раз причина в том, что у всех моделей примерно одинаковые веса. Можно было бы назначить EfficientNet вес 0.9, остальным по 0.05.

Для диагностики проблемы (почему TTA и ансамблирование не помогли), нужно иметь таблицу сравнения точности на сабмите для:

1. EfficientNet без TTA
2. EfficientNet с одной итерацией TTA
3. EfficientNet с 10 итерациями TTA
4. Xception с 10 итерациями TTA
5. И т. д.

По этой таблице возможно было бы видно, в чем проблема.

Кстати, можно было бы сделать дополнительные итерации TTA только для тех изображений, для которых модель выдала несколько разных ответов на 10 итерациях TTA.

> На этом шаге я рассчитывал для каждой из моделей сделать предсказания классов для каждой из 3 моделей (Xception, ResNet50V2, EfficientNetB7), объединить их в pandas.DataFrame и обучить на нем RandomForestClassifier из пакета sklearn. Но как-нибудь в другой раз.

Только это нужно делать не на той обучающей выборке, на которой обучались сети. Например, обучить RandomForestClassifier на той выборке, которая была валидационной при обучении сетей.

> почему fine tuning с постепенной разморозкой базовой модели стабильно показывает результат хуже, чем обучение модели целиком?

Сложный вопрос, я честно говоря не знаю ответа. У меня получался такой же результат. Возможно проблема в слоях батч-нормализации, и с их заморозкой ситуация бы изменилась. В общем, тут нужны дополнительные эксперименты. Можно попробовать применять разный learning rate к разным частям сети.

> почему не помогает TTA?

Скорее всего где-то ошибка, хотя я не нашел где именно. Для диагностики можно было бы применить TTA к валидационному датасету и построить на нем график зависимости точности от количества итераций TTA. Думаю, что по этому графику много стало бы понятнее.

> почему запрет разморозки слоев BatchNormalization на порядки ухудшает результат обучения

Этого в коде не нашел, но вероятно где-то ошибка, так не должно быть.