Здравствуйте! Далее комментарии по коду работы:

> применен способ заполнения пропусков с помощью ImageDataAugmentor с использованием библиотеки аугментации изображений albumentations

Заполнение пропусков нужно в табличных данных, а здесь-то пропусков в данных нет. Поэтому я не очень понимаю что имеется в виду под «заполнением пропусков». Это называется просто «аугментации».

> EfficientNet, в свою очередь, показал очень скромные результаты, не доходившие до 90%.

Где-то ошибка в коде, потому что модель сама по себе хорошая. Далее по коду постараюсь понять где именно проблема.

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

CenterCrop можно заменить на RandomCrop, если цель – создание случайных аугментаций.

```
train_datagen = ImageDataAugmentor(rescale=1/255,
```

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. Для EfficientNet из keras.applications нормализация не требуется.

```
test_generator = test_datagen.flow_from_directory(
```

Совершенно правильно, что валидацию вы делаете без аугментаций.

```
checkpoint = ModelCheckpoint('best_model.hdf5',
                             monitor = 'val_accuracy',
                             verbose = 1,
                             mode = 'max',
                             save_best_only = True)
```

Правильно добавили параметр save_best_only.

```
model.add(L.Dense(256, activation='relu'))
model.add(L.BatchNormalization())
```

Здесь непонятно из каких соображений добавлены эти слои именно с такими параметрами. Обычно такие слои не добавляют. Но если вы пробовали разные виды голов, и такая оказалась наилучшей, то хорошо сделать таблицу и в ней показать разные архитектуры головы и достигнутые значения точности.

Есть еще проблема в том, что код дублируется на каждом шаге (step 1- step 4). Можно написать функцию с параметрами и вызвать ее 4 раза, код получится намного лаконичнее.

```
tta_steps = 10
predictions = []
```

```
for i in range(tta_steps):
    preds = model.predict(test_sub_generator, verbose=1)
    predictions.append(preds)
```

Если аугментации сильные, то 10 шагов может не хватить, и тогда точность от TTA может наоборот снизиться.

Проблема с EfficientNet скорее всего связана с неправильной нормализацией данных (нужно нормализацию вообще не делать с EfficientNet).

Отзыв подготовил ментор проекта Олег Зяблов. Если есть вопросы, можете задать их в канале #0-project_7-ford_vs_ferrari, постараюсь помочь разобраться. Успехов в дальнейшем обучении! Обязательно подключайтесь на итоговый созвон-вебинар по проекту **29 января**. Анонс вебинара появится позже в канале #0-project_7-ford_vs_ferrari.