8.2. Руководство к практическому занятию 6.

«Классифицирование изображений на мобильных устройствах с помощью трансферного обучения»

Введение

Трансферное обучение в ИИ – это метод, который берет обычно небольшую часть крупной натренированной модели и повторно использует ее в новой модели *для родственной задачи*, тем самым избавляясь от необходимости доступа к очень крупным тренировочным данным и вычислительным ресурсам для тренировки исходной модели. В целом трансферное обучение в ИИ по-прежнему остается открытой задачей, так как во многих ситуациях, в которых человеку требуется всего несколько эмпирических примеров, чтобы научиться понимать что-то новое, искусственному интеллекту понадобится гораздо больше времени на его тренировку, пока он не научится. Тем не менее в области распознавания изображений трансферное обучение оказалось очень эффективным.

Современные глубоко обучающиеся модели распознавания изображений – это, как правило, глубокие нейронные сети, или, конкретнее, **глубокие сверточные нейронные сети** (Convolutional Neural Network, CNN) с многочисленными слоями. Нижние слои такой CNN-сети отвечают за заучивание и распознавание более низкоуровневых признаков, таких как края, контуры и части изображения, в то время как последний слой определяет категорию изображения. Для разных типов объектов, таких как породы собак или разновидности цветов, нам не нужно заново заучивать параметры или веса нижних слоев сети. На самом деле нам пришлось бы тренировать современную CNN-сеть на протяжении многих недель с нуля, пока она не заучит все веса, как правило, миллионы или даже больше, прежде чем она начнет распознавать изображения. Трансферное обучение в случае классифицирования изображений позволяет нам просто-напросто повторно натренировать последний слой такой CNN-сети с помощью нашего конкретного набора изображений, как правило, менее чем за час, оставляя веса всех других слоев без изменения и достигая примерно такой же точности, как если бы мы тренировали всю сеть с нуля в течение нескольких недель.

Второе основное преимущество трансферного обучения заключается в том, что для вторичной тренировки последнего уровня CNN-сети требуется лишь небольшое количество тренировочных данных. Если бы нам пришлось тренировать миллионы параметров глубокой CNN-сети с нуля, то нам понадобился бы очень крупный объем тренировочных данных. А вот в случае вторичной тренировки, например для распознавания пород собак, нам потребуется всего 100+ изображений для каждой породы, чтобы построить модель с более высокой точностью классифицирования пород собак, чем исходная модель классифицирования изображений.

Обычно в трансферном обучении при классификации изображений используются две самые лучшие предварительно натренированные CNN-модели TensorFlow: первая модель – Inception v3, более точная модель, чем MobileNet, оптимизирована по точности, но с более крупным размером (около 20M). Другая модель, MobileNet, оптимизирована по размеру (менее 5M) и эффективности для работы на мобильных устройствах.

Поскольку мы разрабатываем приложение для мобильных устройств, то будем использовать модель MobileNet, а также набор данных о породах собак, для вторичной тренировки моделей и генерирования более качественных моделей распознавания пород собак.

Подготавливаем вторичную тренировку модели

В исходном коде TensorFlow, который установлен, есть сценарий Python, **tensorflow/examples/image_retraining/retrain.py**, и его можно использовать для вторичной тренировки моделей Inception v3 или MobileNet. Перед запуском сценария вторичной тренировки модели MobileNet для распознавания пород собак сначала необходимо скачать набор данных о собаках Stanford Dogs Dataset (http://vision.stanford.edu/aditya86/ImageNetDogs), который содержит изображения 120 пород собак (требуется скачать только изображения по ссылке, а не аннотации). Распакуйте скачанный файл изображений собак images.tar в папку ~/Downloads, и в папке ~/Downloads/Images вы должны увидеть список папок, как показано на следующем ниже скриншоте. Каждая папка соответствует одной породе собак и содержит около 150 изображений (вам не нужно указывать метки явным образом для изображений, так как имена папок используются для маркировки изображений, содержащихся в папках):

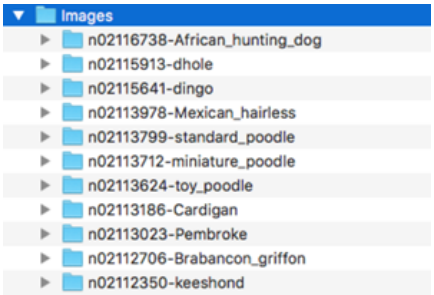

Рис.1. Изображения набора Dogset, распределенные по папкам, или меткам пород собак

Для дальнейшей тренировки нужно создать каталоги /tf_file, и /tf_file/dogs_bottleneck_mobilenet. Далее вводим команду вторичной тренировки модели:

```
python tensorflow/examples/image_retraining/retrain.py
 --output_graph=/tf_files/dog_retrained_mobilenet10_224.pb
 --output_labels=/tf_files/dog_retrained_labels_mobilenet.txt
 --image_dir ~/Downloads/Images
 --bottleneck_dir=/tf_files/dogs_bottleneck_mobilenet
 --architecture mobilenet_1.0_224
```

Параметры:

--output_graph – обозначает имя и путь вторично натренированной модели;

--output_labels – является файлом, состоящим из имен папок (меток) набора изображений, который затем используется вторично натренированной моделью для классифицирования новых изображений;

--image_dir – является путем к набору изображений, используемому для вторичной тренировки модели;

--bottleneck_dir – используется для кеширования результатов, сгенерированных в слое bottleneck (бутылочном горлышке), слое перед последним слоем; последний слой выполняет классифицирование, используя его результаты. Во время вторичной тренировки каждое изображение используется несколько раз, но значения слоя bottleneck для изображения остаются неизменными даже при будущих повторах сценария retrain.py. Поэтому первый запуск занимает гораздо больше времени, так как он должен создать результаты слоя bottleneck.

--architecture – определяет одну из 16 моделей MobileNet, и значение mobilenet_i.0_224 означает использование модели, в которой 1.0 – это количество параметров (другие три возможных значения: 0.75, 0.50 и 0.25 (1.0 означает большинство параметров и дает высокую точность, но при этом самый большой размер, и 0.25 – наоборот)) и 224 – это размер входного изображения (другие три значения 192, 160 и 128). Если в конец значения параметра архитектуры добавить _quantized: --architecture mobilenet_1.0_224_quantized, то в этом случае модель также будет проквантована, в результате чего получится вторично натренированная модель размером около 5.1 Мб. Неквантованная модель имеет размер около 17 Мб. Про квантование модели будет написано ниже.

Еще дополнительно можно указывать параметр:

--model_dir – обозначает путь к каталогу, в который сценарий retarin.py автоматически должен скачать модель, если в указанном каталоге ее еще нет.

Во время вторичной тренировки вы будете наблюдать 3 значения через каждые 10 шагов, всего по умолчанию 4000 шагов. Первые и последние 20 шагов и конечная тренировочная точность выглядят следующим образом (в вашем случае цифры могут отличаться):

```
INFO: tensorflow:2018-01-03 10:42:53.127219: Step 0: Train accuracy = 21.0%
INFO: tensorflow:2018-01-03 10:42:53.127414: Step 0: Cross entropy = 4.767182
INFO: tensorflow:2018-01-03 10:42:55.384347: Step 0: Validation accuracy = 3.0% (N=100)
INFO: tensorflow:2018-01-03 10:43:11.591877: Step 10: Train accuracy = 34.0%
INFO: tensorflow:2018-01-03 10:43:11.592048: Step 10: Cross entropy = 4.704726
INFO: tensorflow:2018-01-03 10:43:12.915417: Step 10: Validation accuracy = 22.0% (N=100)
...
...
INFO: tensorflow:2018-01-03 10:56:16.579971: Step 3990: Train accuracy = 93.0%
INFO: tensorflow:2018-01-03 10:56:16.580140: Step 3990: Cross entropy = 0.326892
INFO: tensorflow:2018-01-03 10:56:16.692935: Step 3990: Validation accuracy = 89.0% (N=100)
INFO: tensorflow:2018-01-03 10:56:17.735986: Step 3999: Train accuracy = 93.0%
INFO: tensorflow:2018-01-03 10:56:17.736167: Step 3999: Cross entropy = 0.379192
```

INFO: tensorflow:2018-01-03 10:56:17.846976: Step 3999: Validation accuracy = 90.0% (N=100)

INFO: tensorflow: Final test accuracy = 91.0% (N=2109)

Тренировочная точность (train accuracy) – это точность классифицирования на изображениях, которые нейронная сеть использовала для тренировки, и *контрольная точность* (validation accuracy) – точность на изображениях, которые для тренировки нейронной сети не использовались. Поэтому контрольная точность является более надежной мерой того, насколько точной является модель, и она обычно должна быть немного меньше тренировочной точности, но ненамного, если тренировка сходится и выполняется хорошо, то есть если натренированная модель не слишком плотно подогнана под тренировочные данные (не подвержена переподгонке) и если она не подогнана под тренировочные данные плохо (не подвержена недоподгонке).

Если тренировочная точность становится высокой, а контрольная точность остается низкой, то это означает, что модель переподогнана (или переобучена). Если же тренировочная точность остается низкой, то это говорит о том, что модель недоподогнана (недообучена). Кроме того, *перекрестная энтропия* (Cross entropy) является значением функции потери, и если вторичная тренировка выполняется хорошо, то это значение должно в целом становиться все меньше и меньше. И наконец, *окончательная контрольная точность* (Final test accuracy) относится к изображениям, которые не использовались ни для тренировки, ни для контроля. Это, как правило, наиболее точное значение, которое мы можем увидеть относительно вторично натренированной модели.

Как показывают приведенные выше результаты, к концу вторичной тренировки мы видим, что контрольная точность аналогична тренировочной точности (90% и 93% по сравнению с 3% и 21% в начале), а окончательная контрольная точность составляет 91%. Перекрестная энтропия также падает с 4,767 в начале до 0,379 в конце. Так что теперь у нас получилась довольно хорошая модель распознавания собак.

С целью дальнейшего повышения точности вы можете поиграть с другими параметрами сценария retrain.py, такими как шаги тренировки (--how_many_training_steps), скорость заучивания (--learning_rate) и улучшение графических данных (--flip_left_right, --random_crop, --random_scale, --random_brightness). Как правило, это самый утомительный процесс, который сопряжен с большим объемом «грязной работы», как назвал ее Эндрю Нг (Andrew Ng), один из самых известных экспертов по глубокому обучению, в своем видеоролике Nuts and Bolts of Applying Deep Learning to speech («Составные части применения глубокого обучения к обработке речи») (видеоролик доступен по адресу: https://www.youtube.com/watch?v=F1ka6a13S9I).

Исследование графа модели и результатов классификации

После вторичного обучения будет сгенерирован файл меток dog_retrained_labels_mobilenet.txt, а также файл вторично натренированной модели dog_retrained_mobilenet10_224.pb.

Протестировать модель, сгенерированную ранее с помощью сценария label_image, можно следующим образом:

```
bazel-bin/tensorflow/examples/label_image/label_image
 --graph=/tf_files/dog_retrained_mobilenet10_224.pb
 --image=/tmp/lab1.jpg
 --input_layer=input
 --output_layer=final_result
 --labels=/tf_files/dog_retrained_labels_mobilenet.txt
 --input_height=224
 --input_width=224
 --input_mean=128
 --input_std=128
```

Эта команда выведет на экран пять самых лучших результатов классифицирования, которые будут выглядеть следующим образом (поскольку сети варьируются случайным образом, вероятно, вы увидите не совсем то же самое):

```
n02099712 labrador retriever (41): 0.824675
```

n02099601 golden retriever (64): 0.144245

n02104029 kuvasz (76): 0.0103533

n02087394 rhodesian ridgeback (105): 0.00528782

n02090379 redbone (32): 0.0035457

Следует обратить внимание на очень важные параметры --input_iayer (input) и --output_iayer (final_result)– они должны быть такими же, как и те, что определены в модели, для того чтобы классифицирование работало правильно. Их можно получить из графа, из модели, из файла dog_retrained.pb (все это названия одного и того же). Для этого есть два инструмента TensorFlow, которые могут быть очень полезными. Первый называется summarize_graph (резюмировать граф), второй – TensorBoard. Рассмотрим сначала, как можно собрать и запустить summarize_graph:

bazel build tensorflow/tools/graph_transforms: summarize_graph

bazel-bin/tensorflow/tools/graph_transforms/summarize_graph

--in_graph=/tf_files/dog_retrained_mobilenet10_224.pb

Результат:

Found 1 possible inputs: (name=input, type=float(1), shape=[1,224,224,3])

No variables spotted.

Found 1 possible outputs: (name=final_result, op=Softmax)

Found 4348281 (4.35M) const parameters, 0 (0) variable parameters, and 0 control_edges

Op types used: 92 Const, 28 Add, 27 Relu6, 15 Conv2D, 13 Mul, 13 DepthwiseConv2dNative,

10 Dequantize, 3 Identity, 1 MatMul, 1 BiasAdd, 1 Placeholder, 1 PlaceholderWithDefault, 1

AvgPool, 1 Reshape, 1 Softmax, 1 Squeeze

Использование TensorBoard

Инструмент TensorBoard дает более полную картину графа модели. Если у вас платформа TensorFlow установлена непосредственно из двоичного файла, то можно просто запустить локальный сервер TensorBoard, так как по умолчанию он установлен в папку /usr/iocai/bin. Однако, если вы установили TensorFlow из исходного кода, то для сборки локального сервера TensorBoard необходимо выполнить следующие ниже команды:

git clone https://github.com/tensorflow/tensorboard

cd tensorboard/

bazel build //tensorboard

Теперь убедитесь, что у вас есть папка /tmp/retrained_iogs, созданная автоматически при выполнении сценария retrain.py, и выполните команду

bazel-bin/tensorboard/tensorboard --logdir /tmp/retrain_logs

Затем запустите браузер и направьте его по URL-адресу http://locaihost:6006.

Сначала вы увидите график точности, как показано на следующем ниже скриншоте:

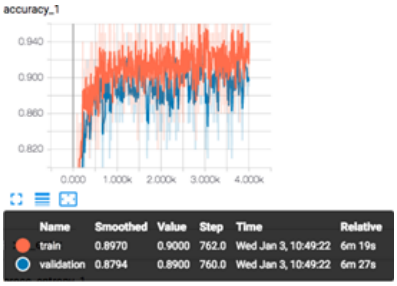

Рис.2. Тренировочная и контрольная точность вторично натренированной модели

График перекрестной энтропии на следующем ниже скриншоте выглядит так, как мы его описывали ранее, рассматривая результаты выполнения сценария retrain.py:

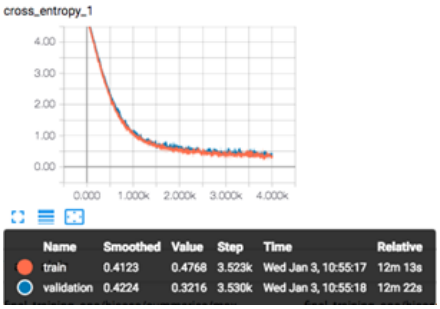

Рис.3. Тренировочная и контрольная перекрестная энтропия вторично натренированной модели

Теперь перейдите на вкладку Graphs (Графы), и вы увидите полный граф нейронной сети со всеми параметрами слоевоперацию с именем Input и еще одну с именем final_result, как показано ниже (ваша схема может отличаться):

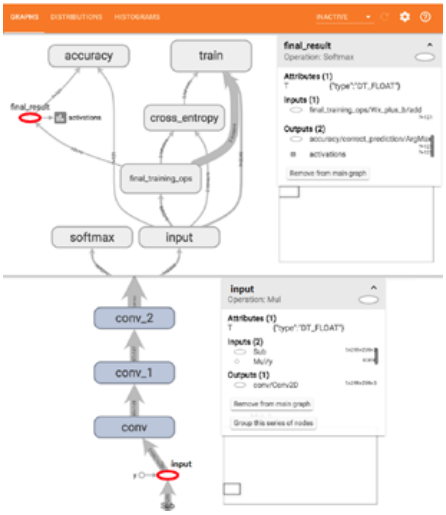

Рис.4. Узлы Mul и final_result во вторично натренированной модели

TensorBoard в Google Colab NoteBook

Для использования TensorFlow в Google Colab нужно в ячейке Ноутбука ввести команду:

```
%tensorboard --logdir logs
```

Подробнее об использовании TensorFlow в Google Colab можно почитать - тут.

Оптимизация модели

Обычно размер файла вторично натренированной модели файл (*.pb) слишком велик, (может быть более 80 Мб), и он должен пройти два этапа оптимизации перед ее развертыванием на мобильных устройствах:

1. **Очистка от неиспользуемых узлов**: удаление узлов модели, которые используются только во время тренировки, но не требуются во время выведения заключения;

2. Квантование модели: конвертирование всех 32-разрядных параметров лит уменьшить размер модели примерно до 25% от ее первоначального размера, сохраняя при этом точность выводимого заключения примерно такой же.

Существует два способа выполнения двух предыдущих задач:

1. Применение инструмента strip_unused (удалить неиспользуемые).
2. Способ связан с применением инструмента transform_graph (трансформировать граф).

Рассмотрим, как работает первый метод. Сначала выполните следующие ниже команды, чтобы создать модель, в которой все неиспользуемые узлы удалены:

```
bazel build tensorflow/python/tools: strip_unused

bazel-bin/tensorflow/python/tools/strip_unused

 --input_graph=/tf_files/dog_retrained_mobilenet10_224.pb

 --output_graph=/tf_files/stripped_ dog_retrained_mobilenet10_224.pb

 --input_node_names=input

 --output_node_names=final_result

 --input_binary=true
```

Далее выполните следующую ниже команду квантования модели:

```
python tensorflow/tools/quantization/quantize_graph.py

 --input=/tf_files/stripped_dog_retrained_mobilenet10_224.pb

 --output_node_names=final_result

 --output=/tf_files/quantized_stripped_dogs_retrained_mobilenet10_224.pb

 --mode=weights
```

Использование инструмента transform_graph рекомендуется в TensorFlow 1.4. Он отлично работает со сценарием Python label_image, но все же иногда вызывает неправильные результаты распознавания при развертывании в приложениях для iOS и Android.

```
bazel build tensorflow/tools/graph_transforms: transform_graph

bazel-bin/tensorflow/tools/graph_transforms/transform_graph

--in_graph=/tf_files/dog_retrained.pb

--out_graph=/tf_files/transform_dog_retrained.pb

--inputs='Mul'

--outputs='final_result'

--transforms='

strip_unused_nodes(type=float, shape="1,299,299,3")

fold_constants(ignore_errors=true)

fold_batch_norms

fold_old_batch_norms

quantize_weights'
```

На проверочных данных сценарий label_image с обеими моделями – квантованной quantized_stripped_dogs_retrained.pb и трансформированной transform_ dog_retrained.pd – работает правильно, но только первая из них работает верно в приложениях для iOS и Android.

Использование вторично натренированных моделей в примере приложения для Android

Чтобы применить модель, вторично на MobileNet, в приложении TF Classify для Android необходимо выполнить следующие действия:

1. Откройте пример приложения TensorFlow для Android, расположенный в папке tensorfiow/exampies/Android, с помощью среды разработки Android Studio.

2. Перетащите вторично натренированную модель, dog_retrained_mobilenet10_224.pb, а также файл меток dog_retrained_labels.txt в папку ресурсов assets приложения.

3. Откройте файл app\src\main\java\com\ailabby\mobileai\hellotensorflow\MainActivity.java и для использования модели, вторично натренированной на модели MobileNet, замените следующий программный код:

```
private static final int INPUT_SIZE = 224;
private static final int IMAGE_MEAN = 117;
private static final float IMAGE_STD = 1;
private static final String INPUT_NAME = "input";
private static final String OUTPUT_NAME = "output";
```

на строки (пути к файлам скорректируйте на свои):

```
private static final int INPUT_SIZE = 224;
private static final int IMAGE_MEAN = 128;
private static final float IMAGE_STD = 128;
private static final String INPUT_NAME = "input";
private static final String OUTPUT_NAME = "final_result";
private static final String MODEL_FILE = "file:///android_asset/dog_retrained_
mobilenet10_224.pb";
private static final String LABEL_FILE = "file:///android_asset/dog_retrained_labels.txt";
```

Для тестирования программы вместо камеры телефона, можно задать файл с собачкой :

```
private static final String IMG_FILE = "labrador1.jpg";
```

4. Подключите устройство Android к компьютеру и запустите на нем приложение. Затем нажмите на приложении TF Classify и наведите фотокамеру на несколько фотографий собак, и вы увидите самые лучшие с точки зрения распознавания результаты на экране.

Последнее изменение: Понедельник, 6 марта 2023, 12:53

◄ 8.1. Теория: Фреймворки Keras и OpenCV в мобильных приложениях с элементами искусственного интеллекта

Перейти на...