Классификация редких дорожных знаков

Борис Фаизов, Влад Шахуро

Graphics & Media Lab
Vision Group

Это задание посвящено классификации редких дорожных знаков. Она является нерешённой для большого количества классов редких знаков. В случае появления нового вида дорожных знаков, трудно собрать достаточное количество реальных обучающих примеров. Качественные синтетические выборки позволяют решить проблему с отсутствующими классами и со сборкой данных с примерами этих знаков. В этом задании Вам предстоит сгенерировать собственную простую синтетическую выборку для задачи классификации дорожных знаков и показать её эффективность. Также Вам будет предложено реализовать специальный классификатор, который работает при помощи метода ближайших соседей. При этом улучшаются значения метрик. Для удобства выполнения задание разделено на пункты.

Таблица 1: Примеры реальных(сверху) и синтетических(снизу) знаков.

Для выполнения задания нужно скачать из проверяющей системы:

1. Скрипт для тестирования `run.py`. Добавьте ему права на выполнение (`chmod +x run.py` в консоли).

2. Архив с тестами. Разархивируйте его в папку `tests` и рядом с этой папкой положите тестовый скрипт `run.py` и создайте файл решения `rare_traffic_sign.py`. В файле решения необходимо написать все функции в данном задании, а затем, после успешного локального тестирования, сдать его в проверяющую систему. Для запуска тестов Вам понадобится библиотека `pytest`.

3. Дополнительные файлы для решения. В этом задании это:

   - Файл `classes.json`. Он содержит информацию о классах, их порядковых номерах и их частоте.
   - Архив `cropped-train.zip`. Он содержит обучающую выборку датасета RTSD.
   - Архив `icons.zip`. Он содержит иконки всех типов дорожных знаков.
   - Архив `smalltest.zip` и файл `smalltest_annotations.csv`. Они содержат маленькую часть тестовой выборки датасета RTSD.
   - Архив `background_images.zip`. Он содержит изображения фона для аугментации иконок знаков.

Файлы должны располагаться следующим образом:

```
├── rare_traffic_sign.py
├── run.py
├── classes.json
├── cropped-train/
├── icons/
├── smalltest/
├── smalltest_annotations.csv
├── background_images/
└── tests/
    ├── 00_unittest_dataset_input/
    ├── 01_unittest_custom_batch_sampler_input/
    ├── 02_unittest_index_sampler_input/
    ├── 03_unittest_smalltest_input/
    └── ...
```

1 Критерии оценки

Максимальная оценка за задание — 10 баллов. Баллы за каждый пункт задания указаны далее. Можно менять предложенную структуру кода, если она все равно позволит проходить тесты и применять модели.

2 Чтение датасета (0.5 балла)

В архиве *cropped-train.zip* Вам предлагается обучающая выборка **RTSD** для классификации знаков. Структура данных следующая: для каждого класса есть своя папка, в которой содержатся картинки со знаками данного класса. В архиве *smalltest.zip* - маленький фрагмент тестовой выборки, это просто папка с картинками. А в файле *smalltest_ annotations.csv* располагается разметка маленькой тестовой выборки. Также в файле *classes.json* располагается соответствие класов и их индексов с пометкой, является ли данный класс редким или частем.

В таблице 2 приведена статистика по количеству знаков редких и частых классов. Редкими мы считаем те классы, которые отсутствуют в обучающей выборке.

	Все	Редкие	Частые
Train set	79896	0	79896
Test set	25613	1622	23991

Таблица 2: Статистика по классам для задачи классификации набора RTSD.

Реализуйте класс `DatasetRTSD`, который будет генерировать данные при обучении дальнейших моделей. Этот датапровайдер читает и сохраняет список файлов из переданных ему директорий с данными. Также реализуйте класс `TestData`, который генерирует данные для валидации и тестирования. Проверьте реализацию с помощью юнит-тестов:

```
$ ./run.py unittest dataset
```

3 Реализация метрик

Мы будем считать *Accuracy* по всем примерам и по-отдельности для редких(*rare*) и частых(*freq*) типов. Для редких и частых типов точность вырождается в микроусредненный *Recall*, так как мы представляем, что знаков другого типа нет. Формула для редких знаков (частые аналогично):

$$M = \frac{|\#\ correctly\ classified\ rare\ signs|}{|\#\ all\ rare\ signs|}$$

Добавьте `calc_metric()` из `run.py`.

4 Простой классификатор (1.5 балла)

Реализуйте простой классификатор на основе предобученной нейросети (например, **ResNet-50**) в классе *CustomNetwork*. Необходимо заменить последний полносвязный слой в нейросети на линейный слой с $internal\_features$ параметрами. Далее надо добавить отдельным параметром функцию активации *ReLU* и полносвязный слой для классификации с нужным числом классов (205). Также реализуйте функцию `train_simple_classifier()` для обучения классификатора. Обученную модель нужно сохранить в файле `simple_model.pth`.

Для тестирования модели нужно реализовать функции `apply_classifier()` и `test_classifier()`. Первая функция применяет переданный классификатор и возвращает список из предсказаний. Вторая функция, используя первую, возвращает метрику на всех знаках, на редких знаках и на частых знаках при наличии разметки тестовой выборки. Вы можете проверить, что Ваша модель корректно тестируется на открытой части выборки при помощи юнит-теста (за него выставляется 0.5 балла если модель на открытой части тестовой выборки наберет на частых классах точность $> 70\%$):

```
$ ./run.py unittest smalltest
```

Баллы:

- 0.5 балла - Модель на полной тестовой выборке на частых классах на сервере набирает точность $> 70\%$.
- 1.5 балла - Модель на полной тестовой выборке на частых классах на сервере набирает точность $> 75\%$.

5 Обучение с простыми синтетическими данными (3 балла)

В этом пункте Вам предлагается сгенерировать простую синтетическую выборку для задачи классификации дорожных знаков. Примеры таких картинок в таблице 1. В архиве *icons.zip* содержатся четырёхканальные иконки дорожных знаков разных классов. В последнем канале содержится маска дорожного знака, чтобы можно было его встраивать на фон. В архиве *background_images.zip* содержатся 1000 снимков фона, на которые можно встраивать знаки. Схема генерации дорожных знаков следующая:

1. Иконка с дорожным знаком ресайзится до случайного размера от 16 до 128.
2. Делается паддинг иконки по краям случаного размера от 0% до 15% от размера иконки.
3. Случайно изменяется цвет иконки. Для этого она сначала переводится в цветовую палитру HSV (из RGB). Далее по отдельности случайным образом изменяются компоненты этой палитры. Далее иконка назад переводится в RGB.
4. Иконка поворачивается на случайный угол от -15 до 15 градусов.

5. Иконка «размазывается», как будто находится в движении. Для этого она сворачивается с специально построенным ядром. Это ядро представляет собой повернутый на случайный угол от -90 до 90 градусов сдвиг иконки.

6. Обработка Гауссовым фильтром.

7. Встраивание полученной иконки по маске в фрагмент реального фона.

Вам необходимо реализовать пропущенные функции в классе `SignGenerator`. Также нужно дописать функции `generate_one_icon()` и `generate_all_data()`.

После этого нужно сгенерировать простую синтетическую выборку при помощи функции `generate_all_data()`, в которой будет по 1000 картинок каждого класса. Эта функция запускает пул параллельно работающих процессов, каждый из которых будет генерировать иконку своего типа. Это необходимо, так как процесс генерации очень долгий. Каждый процесс работает в функции `generate_one_icon`.

На смеси полученной синтетической и настоящей выборки нужно обучить простую модель из первого пункта. Для этого реализуйте пропущенный код в `train_synt_classifier()`. Качество модели должно вырасти. Новую обученную модель нужно сохранить в файле `simple_model_with_synt.pth`.

Баллы:

- 1 балл - Модель на полной тестовой выборке на сервере набирает точность $> 60\%$ на всех знаках и полноту $> 39\%$ на редких знаках.
- 2 балл - Модель на полной тестовой выборке на сервере набирает точность $> 65\%$ на всех знаках и полноту $> 43\%$ на редких знаках.
- 3 балла - Модель на полной тестовой выборке на сервере набирает точность $> 70\%$ на всех знаках и полноту $> 47\%$ на редких знаках.

6 Улучшенный классификатор (5 баллов)

В этом пункте Вы построите специальный классификатор, который работает лучше для редких знаков в данной задаче. Он состоит из двух отдельных частей:

- Обученная специальным образом нейросеть для классификации дорожных знаков. В ней будет дополнительно введена отдельная функция потерь на признаки, генерируемые в предпоследнем слое сети.
- Метод ближайших соседей для классификации дорожных знаков по признакам с предпоследнего слоя нейросети.

Общая схема работы описывается так. Берется знак, прогоняется через нейросеть, извлекаются признаки с предпоследнего слоя. По ним классификация выполняется при помощи метода ближайших соседей.

6.1 Архитектура нейронной сети (0.5 балла)

Сначала напишем код для обучения нейронной сети. Тут Вам нужно добавить какой-нибудь вариант функции потерь на признаки предпоследнего слоя нейросети. Для этого сначала реализуйте класс `FeaturesLoss`, который будет считать такую функцию потерь для примеров в батче. Можно использовать любую подходящую функцию. Например, одну из тех, что были рассмотрены на семинаре (*contrastive loss*, *triplet loss*, ...)

Например, для *contrastive loss* формула, по которой она будет считаться, имеет вид:

$$L = \frac{1}{2}\frac{\sum_{i,j=1}^{N}[y_i = y_j] * ||f_i - f_j||_2^2}{\sum_{i,j=1}^{N}[y_i = y_j]} + \frac{1}{2}\frac{\sum_{i,j=1}^{N}[y_i \neq y_j] * max(m - ||f_i - f_j||_2, 0)^2}{\sum_{i,j=1}^{N}[y_i \neq y_j]}$$

Где y_i - реальная метка объекта i, а f_i - вектор признаков объекта i, извлеченный нейросетью. m - margin (например, $m = 2$).

Далее реализуйте класс `CustomBatchSampler`, который будет семплировать подходящие батчи. Дело в том, что классов 205, а в выборке они представлены неравномерно. Поэтому какие-то классы будут семплироваться часто, какие-то реже, в одном батче большинство объектов будут разного класса. Мы же хотим контролировать, чтобы внутри одного батча было $classes\_per\_batch$ разных классов и для каждого класса было ровно $elems\_per\_class$ объектов (например, $classes\_per\_batch = 32$ и $elems\_per\_class = 4$). Тогда наша *loss*-функция будет иметь смысл. Проверьте реализацию с помощью юнит-тестов:

```
$ ./run.py unittest custom_batch_sampler
```

Теперь нужно реализовать функцию `train_better_model()`, в которой будет правильно проинициализирован $features\_criterion$, а обучение будет происходить на смеси реальных и синтетических данных.

Веса обученной нейросети надо сохранить в файле `improved_features_model.pth`

6.2 Модель с k-NN (0.5 балла)

Реализуйте класс `ModelWithHead` и функцию `train_head()`, в которой будет происходить обучение и предсказание k-NN-головы. Чтобы обучить k-NN нужны данные. Мы будем использовать в качестве данных случайные $examples\_per\_class$ (например, $examples\_per\_class = 20$) примеров каждого класса из **синтетических** данных. Эти примеры далее будем называть "индексом".

Чтобы создать этот индекс, реализуйте класс `IndexSampler`, который при передаче в *DataLoader* в качестве *sampler* будет семплировать нужное число примеров всех классов. Полученный датапровайдер будет передаваться в функцию *train_head()* класса *ModelWithHead*. В ней нужно извлечь нейросетью признаки предпоследнего слоя из всех объектов индекса, нормализовать эти признаки и обучить на них k-NN для классификации знаков на 205 классов по признакам. Проверьте реализацию с помощью юнит-тестов:

```
$ ./run.py unittest index_sampler
```

Обученный k-NN надо сохранить в файле `knn_model.bin` при помощи функции *save_head()*.

Полученную модель протестируйте с помощью уже реализованной функции `test_classifier()`. Метрики должны вырасти.

6.3 Баллы за итоговую модель (4 балла)

Полученные Вами модель `improved_features_model.pth` и k-NN `knn_model.bin` нужно загрузить на сервер. В зависимости от полученных значений метрик будет выставляться оценка.

- 1 балл - Модель на полной тестовой выборке на сервере набирает точность $> 65\%$ на всех знаках и полноту $> 48\%$ на редких знаках.
- 2 балла - Модель на полной тестовой выборке на сервере набирает точность $> 70\%$ на всех знаках и полноту $> 50\%$ на редких знаках.
- 3 балла - Модель на полной тестовой выборке на сервере набирает точность $> 70\%$ на всех знаках и полноту $> 55\%$ на редких знаках.
- 4 балла - Модель на полной тестовой выборке на сервере набирает точность $> 75\%$ на всех знаках и полноту $> 60\%$ на редких знаках.