Машинное обучение, DS-поток

Домашнее задание 10

Правила:

- Дедлайн **22 мая 23:59**. После дедлайна работы не принимаются кроме случаев наличия уважительной причины.
- Выполненную работу нужно отправить на почту `mipt.stats@yandex.ru` , указав тему письма `"[ml] Фамилия Имя – задание 10"` . Квадратные скобки обязательны. Если письмо дошло, придет ответ от автоответчика.
- Прислать нужно ноутбук, его pdf-версию (без архивов) и ссылку на google диск с весами лучшей модели. Названия файлов должны быть такими: `10.N.ipynb` и `10.N.pdf` , где `N` - ваш номер из таблицы с оценками.
- Теоретические задачи необходимо оформить в tехе/markdown или же прислать фотку в правильной ориентации рукописного решения, **где все четко видно**.
- Решения, размещенные на каких-либо интернет-ресурсах не принимаются. Кроме того, публикация решения в открытом доступе может быть приравнена к предоставлении возможности списать.
- Для выполнения задания используйте этот ноутбук в качестве основы, ничего не удаляя из него.

Баллы за задание:

- Часть 1 – 20 баллов
- Часть 2 – 5 баллов

Часть 1: Сверточные сети

В этой домашней работе вам предстоит построить сверточную сеть для классификации картинок из датасета **"Best Artworks Of All Time"** (https://www.kaggle.com/ikarus777/best-artworks-of-all-time). По изображению картины нужно предсказать автора (художника), написавшего ее.

Пожалуйста, прочитайте то, что написано ниже, там изложены требования к вашей работе и полезные советы!

Цель задания – построить нейросеть, чтобы достичь максимально возможного accuracy. В конце задания вы должны будете предоставить отчет о проделанной работе.

Оценивание вашей работы:

В сумме за домашнюю работу можно получить 25 баллов.

- $+3$ балла за работу за предоставление детального отчета (требования к нему изложены ниже).
- $+3$ балла за построение нейросети, которая достигает значения Accuracy не менее 20% на тестовом датасете.
- $+2$ балла за каждый следующий пройденный порог.
  - 25% Accuracy
  - 30% Accuracy
  - 32.5% Accuracy
  - 35% Accuracy
  - 37.5% Accuracy
  - 40% Accuracy
- $+2$ балла за выполнение "задания на ваше знание статистики". Вы можете найти его в конце всех советов и требований

- +5 баллов за выполнение второй части задания (Transfer Learning)

Требование к работе:

- В этой части задания **запрещено** использовать предобученные нейросети. Для этого есть вторая часть домашней работы и она оценивается отдельно!
- **Запрещено** использовать тестовые данные за исключением вычисления финальной оценки качества.
- Сохраните веса лучшей модели с помощью `torch.save` (инструкция (https://pytorch.org/tutorials/beginner/saving_loading_models.html)) и пришлите ссылку на файл с весами на google диске. Так мы сможем проверить выполнение предыдущего пункта. **Работы без присланных весов не будут засчитаны.**

  Пример сохранения и загрузки весов:

  ```
  # Save:
  torch.save(model, PATH)

  # Load:
  model = torch.load(PATH)
  model.eval()
  ```

  Более подробную инструкцию можно найти по ссылке (https://pytorch.org/tutorials/beginner/saving_loading_models.html).

Требования к отчёту:

- Опишите свои эксперименты: с чего вы начали, что попробовали улучшить и почему, заработало это или нет, какие вы сделали из этого выводы.
- Опишите вашу лучшую архитектуру, методы обучения и интересные моменты.

Советы:

Архитектура нейросети:

- Это задание может быть решено последовательностью сверток, пулингов, батчнорма и активаций, но не стоит останавливаться на этом.
- Можно рассмотреть такие архитектуры как Inception family (https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/), ResNet family (https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035?gi=9018057983ca), Densely-connected convolutions (https://arxiv.org/abs/1608.06993). Однако вам нужно будет реализовать их самостоятельно.
- Попробуйте сначала построить простую нейросеть, чтобы понять как с ними работать, перед тем как использовать resnet-152.
- Также можно попробовать разные активации: `tanh` , `leaky relu` и другие.

Переобучение:

Если ваша нейросеть переобучается (лосс на тесте падает, а на валидации растет), вот некоторые методы, как с этим бороться:

- Попробуйте добавить Dropout. Не бойтесь удалять много данных, но всегда проверяйте, что это не испортило вам качество.

- Добавьте L2 регуляризацию весов (начните с небольшого значения). Регуляризация контролируется параметром `weight_decay` оптимизатора.
- Попробуйте уменьшать `learning rate` с течением времени. В этом поможет `torch.optim.lr_scheduler`.
- Уменьшите число нейронов в сети.
- Прерывайте обучение (https://github.com/Bjarten/early-stopping-pytorch), если сеть начала переобучаться.

Процесс обучения:

- Воспользуйтесь GPU google colab или любой другой GPU, которая у вас есть.
- Для сокращения вычислительной сложности можно поэксперементировать с параметром `stride`.
- Эксперементируйте с оптимизаторами: `rmsprop`, `nesterov_momentum`, `adam`, `adagrad` и далее. В этом вам поможет `torch.optim`.
- Помните, что некоторым нейросетям требуется 10 эпох, чтобы сойтись, а некоторым – 500. Большие нейросети дольше обучаются.
- Если вы достигли какого-то порога на валидации лучше подождать примерно 10 эпох перед тем как останавливать обучение.

Аугментация данных:

- Вы можете использовать любые библиотеки для аугментации данных, например: torchvision.transforms (https://pytorch.org/docs/stable/torchvision/transforms.html), albumentations (https://albumentations.readthedocs.io/en/latest/api/augmentations.html), augmentor (https://augmentor.readthedocs.io/en/master/), imgaug (https://imgaug.readthedocs.io/en/latest/).
- Попробуйте добавить шум.
- Повернуть картинку + приблизить, чтобы убрать черные края.
- Отразить её вертикально или горизонтально.
- Сократить размер картинки, это позволит сократить параметры сети.
- При аугментации всегда нужно помнить с какими данными мы работаем (разворачивание собаки на 180 градусов вам, наверняка, не поможет, потому что таких примеров, скорее всего, не будет в тестовой выборке).

И главное:

- Тестируйте только **одну идею за раз**.
- Сохраняёте веса моделей (для каждого эксперимента, через N эпох, при достижении наилучшего качества на валидации), чтобы нечаянно не потерять результаты долгой работы.
- Рисуйте кривые обучения (loss и метрику качества) для обучения и валидации.

Задание на применение ваших знаний статистики:

Как всегда у всех горят сроки, заказчик просит скорее получить хорошую модель. Вы обучаете модель и видите по кривой обучения, что некоторый прирост в качестве еще есть. Только обучается она очень долго. Как понять, хватит ли уже проведенных итераций или нужно еще? Давайте проверим, значимо ли отличаются эти изменения. Проверьте стат значимость разницы в качестве на последней итерации и на одной из предыдущих итераций. Если результаты значимо отличаются, то имеет смысл дообучить модель.

P.S. Баллы даются за попытку реализации и за выводы, почему эта идея заработала/не заработала

In [ ]:

```
import os
import shutil

import matplotlib.pyplot as plt
%matplotlib inline
```

Скачаем данные по ссылке:

In [ ]:

```
!wget -O data.tar.xz https://www.dropbox.com/s/w90m55pl7ylgiaf/data.tar.xz?dl=0
!tar -xf data.tar.xz data
!ls data/train | wc -l
!find data/train -type f | wc -l
```

В `train` датасете 51 художник (класс) и 6116 изображения картин (объектов).

Посмотрим на данные:

In [ ]:

```
path_to_img = 'data/train/William_Turner/William_Turner_9.jpg'
image = plt.imread(path_to_img)
plt.figure(figsize=(12, 5))
plt.imshow(image);
```

Разобьем `train` выборку на `train` и `val`:

In [ ]:

```
os.makedirs('data/val', exist_ok=True)

TRAIN_FRAC = 0.7
```

In [ ]:

```
# считываем названия директорий
ARTIST_LIST = {i:name for i, name in enumerate(os.listdir('data/train/'))}
IMAGES_DIR = 'data/train/'

max_train_images = 0

# создаем директорию с валидационной выборкой для каждого художника
for artist in ARTIST_LIST.values():
    os.makedirs(f'data/val/{artist}/', exist_ok=True)

    # считываем выборку картин художника
    artist_path = f'{IMAGES_DIR}/{artist}/'
    images_filename = os.listdir(artist_path)

    # выделяем часть картин для валидации
    num_train = int(len(images_filename) * TRAIN_FRAC)
    max_train_images = max(max_train_images, num_train)
    val_images = images_filename[num_train:]

    print(f'{artist} | train images = {num_train} | val images = {len(val_image
s)}')

    # сохраняем валидационную выборку
    for image_filename in val_images:
        source = f'{IMAGES_DIR}/{artist}/{image_filename}'
        destination = f'data/val/{artist}/{image_filename}'
        shutil.copy(source, destination)
        os.remove(source)
```

Данный датасет очень не сбалансирован по классам, возможные пути решения:

- Random oversampling – включаем несколько копий объектов меньших классов, увеличивая их до размера большего класса
- Random undersampling – не включаем часть объектов больших классов в обучающую выборку

Предлагаем вам самим подумать как стоит бороться с дисбалансом классов и написать код

In [ ]:

```

```

Убедимся еще раз, что в папке train и val все разложено по папкам-классам (авторам). Эта структура папок важна для использования классов PyTorch по работе с данными (`ImageFolder` и `DataLoader`):

In [ ]:

```
!ls data/train
```

In [ ]:

```
!ls data/val
```

Время для ваших экспериментов!

Пока ваши нейросети будут обучаться можно начать заполнять отчет, который находится чуть ниже.

Можете смело использовать код с семинара.

In [ ]:

```
data_dir = 'data/'

train_dataset = datasets.ImageFolder(os.path.join(data_dir, 'train'),
                                     transform=transforms.ToTensor())
val_dataset = datasets.ImageFolder(os.path.join(data_dir, 'val'),
                                   transform=transforms.ToTensor())
```

In [ ]:

```

```

Протестируйте своё решение:

In [ ]:

```
# Используйте test_dataset только для финальной оценки качества
test_dataset = ...
```

In [ ]:

```

```

In [ ]:

```
test_accuracy = ...
```

In [ ]:

```
print("Итоговый результат:")
print("  test accuracy:\t\t{:.2f} %".format(
    test_accuracy * 100))

if test_accuracy * 100 > 40:
    print("Achievement unlocked: Transformer!")
elif test_accuracy * 100 > 35:
    print("Achievement unlocked: LSTM!")
elif test_accuracy * 100 > 30:
    print("Achievement unlocked: RNN!")
elif test_accuracy * 100 > 25:
    print("Achievement unlocked: perceptron!")
else:
    print("We need more \"layers\"! Follow instructons below")
```

Обязательно заполните отчет. Опишите свои эксперименты: с чего вы начали, что попробовали улучшить и почему, заработало это или нет, какие вы сделали из этого выводы. Также обязательно опишите вашу лучшую архитектуру, методы обучения и интересные моменты.

Отчет: ....

После __ [минут, часов, дней] обучения, я получил следующие результаты:

- accuracy on training: __
- accuracy on validation: __
- accuracy on test: __

Часть 2: Transfer Learning

Попробуйте теперь использовать предобученную модель для классификации и сравните результаты. Сделав эту часть задания вы сможете получить 5 баллов.

Вы можете пробовать любые предобученные архитектуры. Некоторые из них можно найти по ссылке torchvision.models (https://pytorch.org/docs/stable/torchvision/models.html).

Загрузите веса модели:

In [ ]:

```
```

Все предобученные модели можно разделить на две части:

- Сверточная часть, которая работает как feature extractor.
- Классификатор.

Скорее всего вам потребуется заменить предобученный классификатор, чтобы использовать модель для работы с новым датасетом. Наиболее популярные подходы при замене классификатора:

- Fully-connected слои.
- Global average pooling. Подробнее можно прочитать в статье (https://arxiv.org/pdf/1312.4400.pdf).
- Linear SVM (https://arxiv.org/pdf/1306.0239.pdf), если хочется чего-нибудь необычного.

Несколько советов:

- Так как входные данные разных моделей могут иметь разный размер, вам может потребоваться поменять исходный размер изображения. В этом могут помочь, например, `cv2.resize`, `skimage.resize` или `torch.transforms.Resize`.
- Для дообучения слоев с предобученными весами можно использовать меньший learning rate, чем для обучения слоев со случайной инициализацией. Так как в самом начале обучения градиенты от случайно инициализированного классификатора могут слишком сильно изменить хорошие предобученные веса.

Стратегии:

- Использовать предобученную нейросеть как feature extractor, убрав последний FC слой. Обучить новый классификатор на полученном признаковом описании.
- При обучении классификатора продолжить backpropagation на более глубокие слои нейросети ("разморозить" их). При этом возможно "разморозить" все слои или зафиксировать несколько начальных слоёв и не обучать их.
- Вы можете совместить стратегии: сначала обучить классификатор, а потом постепенно размораживать слои и обучать их с меньшим learning rate-ом.

Какую стратегию вы выбрали и почему?

Вывод: __

In [ ]:

```
```

Сравните результаты предобученной модели и результаты из первой части. Сделайте вывод.

In [ ]:

```
```