ТЕХНОЛОГИЧЕСКАЯ КАРТА ЗАНЯТИЯ

Тема занятия: Практика Transfer Learning

Аннотация к занятию: на данном занятии обучающиеся научатся использовать Transfer Learning для классификации изображений.

Цель занятия: сформировать у обучающихся представление о Transfer Learning для классификации изображений.

Задачи занятия:

- познакомить обучающихся с подразделом машинного обучения — Transfer Learning;
- научить работать со слоями (замораживать и менять слои сети);
- рассмотреть, как работать с произвольными датасетами картинок;
- применить полученные знания на практике.

Ход занятия

Этап занятия	Время	Деятельность педагога	Комментарии, рекомендации для педагогов
Организационный этап	2 мин.	Добрый день! Добро пожаловать на урок.	Приветствие. Создание в классе атмосферы психологического комфорта
Постановка цели и задач занятия. Мотивация учебной деятельности обучающихся	10 мин.	**Вопрос для обсуждения** Как работает Transfer Learning? **Ответы обучающихся** На этом практическом занятии мы научимся использовать Transfer Learning. Мы загрузим предобученную на ImageNet нейросеть ResNet, заморозим несколько первых её слоев, заменим последний слой сети и дообучим её на новом датасете классификации пород собак.	Способствовать обсуждению мотивационных вопросов

Изучение нового материала	50 мин.	Начинаем, как обычно, с импорта библиотек. Импортируем сразу все нужные нам библиотеки. Это NumPy и Matplotlib, вспомогательные библиотеки, модули PyTorch и Torchvision. Из Torchvision мы импортируем не только datasets и transforms, с которыми мы уже работали, но и ещё один модуль — models. В нём как раз и лежат архитектуры нейросетей VGG и ResNet. В первую очередь давайте научимся эти нейросети загружать. Загрузим модель ResNet-18: вариант ResNet, в которой 18 слоёв. Делается это так: model.resnet18. Подаём аргумент pretrained=True на вход. pretrained=True означает, что будет загружена версия нейросети, у которой веса уже предобучены на ImageNet. Если поставить pretrained=False, то будет загружена модель со случайными весами, не предобученная. Но мы на этом практическом занятии будем учиться модель дообучать, поэтому загружаем предобученную нейросеть. После загрузки модели выведем её на экран. Запускаем. Нейросеть скачалась, и тут вывелась информация о её устройстве. Видим, что нейросеть состоит из знакомых нам элементов: свёрточных слоёв, батч норма, функции активации ReLU. В самом низу после всех свёрточных слоев стоит AveragePooling и один полносвязный слой. Слева, около каждого слоя, написано его название. Полносвязный слой называется fc. Некоторые свёрточные слои объединены в блоки. Здесь видно название каждого блока. Это блок layer1, у него два вложенных блока: 0 и 1. И внутри них уже слои. Вот так устроена наша нейросеть. Прежде чем мы будем её дообучать на новую задачу, протестируем на одной картинке:	Для справки: https://tonais.ru/library/transfer-learning-pytorch-python Перед уроком рекомендуется ознакомиться с материалами, представленными на сайте.

		прогоним через нейросеть какую-нибудь картинку и получим результат. Тут есть один очень важный момент. Помните, мы говорили про трансформации, которые можно совершать с картинками перед тем как из них формируется батч для подачи в сеть? Вот ноутбук предыдущей практики. При загрузке датасета MNIST мы указывали transform.ToTensor: это означало, что при формировании батчей из картинок CIFAR к ним применяется трансформация ToTensor(). Она переводит картинки в формат тензора. Кроме преобразования ToTensor, есть ещё множество разных, которые можно совершать с картинками. И когда исследовали обучали нейросеть ResNet, которую мы только что скачали, они применяли к картинкам несколько определённых преобразований. При использовании resnet-18 нужно применять те же самые преобразования ко всем картинкам, которые мы подаём на вход сети. В этой ячейке записаны преобразования, которые применялись к картинкам ImageNet при обучении ResNet. Давайте по ним пройдёмся. Во-первых, ResNet принимает на вход изображения размером 224x224. Поэтому перед формированием батчей каждое изображение приводится к этому размеру с помощью двух преобразований: resize и centercrop. Сначала resize с аргументом 226 приводит каждую картинку к размеру 226. Затем из полученной картинки с помощью centercrop вырезается центральный кусок размером 224. Зачем делать именно так, а не просто с помощью resize сразу приводить картинку к размеру 224 — это хороший вопрос. Почему-то авторы решили сделать именно	

		так. Ну а раз авторы предобрабатывали свои картинки именно таким образом, то и мы будем делать так же. Тут у вас, возможно, возник вопрос: а почему мы не использовали Resize раньше, когда делали нейросеть для классификации CIFAR? Дело в том, что в таких датасетах, как MNIST и CIFAR, все картинки заранее приведены к одному размеру. В MNIST все картинки уже размерлм 28x28, а в CIFAR — 32x32. Дополнительно менять их размер не нужно. Но обычно датасеты для реальных задач содержат картинки разных размеров. Потому что собираются такие датасеты чаще всего из интернета. И ImageNet — не исключение. Поэтому перед подачей картинок в сеть их нужно приводить к одинаковому размеру. Далее применяется преобразование ToTensor. Картинки переводятся в формат torch.Tensor. Наконец, последнее преобразование — это нормализация. Нормализация — это приведение распределений пикселей в картинках к одному виду. Если пиксели картинок датасета распределены одинаково, то нейросети проще научиться их классифицировать. Нормализация картинок здесь играет ту же роль, что и нормализация данных в классическом машинном обучении. Для нейросетей нормализовать данные очень важно: так они будут намного лучше учиться. Аргументы здесь — это коэффициенты нормализации, которые применялись при обучении ImageNet. Мы будем использовать такие же. Очень важно при дообучении или тестировании нейросети использовать те же трансформации, которые применялись к	

		входным данным при первичном обучении нейросети. Нейросеть обучила свои веса хорошо работать с картинками определённого вида: определённого размера и с определённым распределением значений пикселей. Чтобы она хорошо обработала новую картинку или эффективно дообучилась под новый датасет, нужно привести новые картинки к тому же виду. Все эти трансформации мы по очереди записываем в массив и собираем из них пайплайн с помощью transforms.Compose. Ниже при формировании датасета мы подадим эту переменную в качестве аргумента transforms. Для стандартных нейросетей вроде ResNet в PyTorch можно эти преобразования самому не писать. Можно присвоить resnet_transform вот такое значение. Это берёт из модуля transforms трансформации специально для resnet18. Итак, трансформации мы задали. Давайте теперь скачаем картинку собачки из интернета и прогоним её через предобученную нейросеть. В этой ячейке скачивается картинка и сохраняется в файл doggie.jpg. Запустим. Теперь тут в файлах мы видим, что картинка появилась. Откроем картинку с помощью библиотеки Pillow и посмотрим на неё. Вот такая картинка пёсика у нас. Теперь перед подачей картинки в сеть прогоним её через наши трансформации. Сделать это просто: подаём картинку в resnet_transforms. Получаем трансформированную картинку.	

		Выведем её размерность. Видим, что теперь это пайторчевый тензор, у которого размерность каналов стоит впереди. Давайте посмотрим, как выглядит трансформированная картинка. Выведем её на экран с помощью plt.imshow. Но перед этим нужно её из формата тензора перевести в NumPy, потому что Matplotlib не умеет визуализировать картинки, представленные в формате тензора. Кроме того, нужно поменять местами каналы: сделать так, чтобы цветовой канал был последним. Это легко сделать с помощью permute. Указываем порядок каналов, который мы хотим видеть. Сначала мы хотим поставить первый канал, затем второй, а затем уже нулевой. Теперь мы можем это запустить. Вот так выглядит картинка после трансформаций. Она стала квадратной и размером двести двадцать четыре на двести двадцать четыре. У неё изменились цвета: это из-за нормализации пикселей. Картинка теперь выглядит странно, но ничего страшного: нейросеть обучилась классифицировать картинки именно такого вида, и для неё это проще. Теперь подадим картинку на вход нейросети и получим ответ. Но просто одну картинку на вход модели мы подать не можем, потому что нейросети всегда ожидают получить на вход батч картинок. Входящий тензор должен быть размерности четыре, а не три. Давайте сделаем из нашей картинки батч размером один. Это делается с помощью метода reshape. Здесь мы пишем размерности, которые хотим получить. Перед тем, как подадим картинку на вход, нужно сделать model.eval(). Напомню, что это нужно делать всегда, когда вы	

		собираетесь тестировать вашу модель, а не обучать. Если не сделать model.eval(), вы получите совсем другие ответы нейросети. Подаём этот тензор на вход модели и выводим её ответ. Ответ модели на картинку — тысяча чисел: оценки вероятности принадлежности картинки к тысяче классов ImageNet. Давайте возьмём argmax от этих значений и узнаем, на какой позиции здесь стоит самое большое число, то есть, к какому номеру класса модель отнесла нашу картинку. Это номер двести семь. Или двести восемь, если считать с единицы. Давайте узнаем, какому классу соответствует номер двести семь. Для этого перейд`м по этой ссылке и найд`м номер 207. Это golden retriever. Кажется, это верный класс: наша собака действительно похожа на золотистого ретривера. Таким образом, мы научились загружать предобученную нейросеть, определять для неё трансформации, прогонять через неё картинку и получать ответ. Давайте теперь переходить к дообучению сети. Мы будем дообучать нейросеть на датасете классификации пород собак. Этот датасет нестандартный. Его нельзя скачать из PyTorch, нужно скачивать из интернета. Здесь в ячейках есть код, который поможет вам скачать данные на Google Colab. Если запустить две эти ячейки, то в Colab появится папка с названием dogs, в которой будет датасет. Но давайте разберём ещё другой вариант работы с датасетом. Можно положить данные на свой Google Диск. Например, у меня на Google Диске лежит датасет собак в папке dogs. Данные уже	

		разбиты на тренировочную, валидационную и тестовую части, каждая лежит в своей папке. Как работать с датасетами, которые лежат на диске в виде файлов, а не скачиваются из модуля PyTorch datasets? Смотрите, чтобы в PyTorch легко можно было работать с датасетом картинок, папка с данными должна иметь определённый вид. Тренировочная, валидационная и тестовая части должны находиться в разных папках. И каждая из этих папок внутри тоже должна быть устроена определённым образом. Они должны содержать столько подпапок, сколько классов в датасете. Каждая подпапка соответствует одному классу, а её название — название класса. Внутри папки класса лежат картинки, соответствующие этому классу. В этой папке, например, лежат тестовые картинки собак породы afgan. Если ваш датасет устроен именно таким образом, то работать с ним в PyTorch будет очень просто. Вернёмся в «ноутбук». Во-первых, чтобы получить из «ноутбука» доступ к папке на Google Диске, нужно диск примонтировать. Это деляется двумя строчками кода. Запускаем ячейку. Нужно пройти авторизацию и разрешить Google Colab доступ к файлам на Google Диске. Теперь из Google Colab можно обращаться к любым файлам, которые лежат у вас на Google диске. Теперь давайте создадим PyTorch-датасеты для тренировочной, валидационной и тестовой частей датасета. В этом нам поможет	

		ImageFolder. На вход этой функции нужно передать путь к папкам с частями датасета. Путь к моим папкам на Google Диске такой. Пишем /content/drive/My Drive/. Это путь к корню моего Google Диска. Далее: мои папки train, val и test находятся в папке Data/dogs. Дописываем сюда data/dogs. Для каждой части датасета вписываем название папки с этой частью: train, val и test. И подаём в качестве трансформаций наши resnet_transforms. Запускаем. Вот так у нас получились три части датасета. Можно вывести переменную train_data и посмотреть информацию о данных: сколько в них картинок, по какому пути на диске лежит датасет и так далее. Теперь заведем три даталоадера: по одному на каждую часть датасета. Делается это как обычно. Поставим butch size 64. Shuffle True на третьей части, False на двух других частях. Данные готовы. Подготовим нейросеть для дообучения: поменяем её последний слой и заморозим первые слои. Сначала поменяем последний слой. Ещё раз загрузим модель и посмотрим на неё. Название последнего полносвязного слоя — fc. По этому названию мы можем получить доступ к слою model.fc. Выведем: это действительно слой нашей сети. Поэтому заменить слой очень просто. Просто напишем, что model.fc теперь равно новому линейному слою nn.Linear с количеством выходящих нейронов семьдесят, потому что в нашей задаче классификации пород собак 70 классов. Количество входящих нейронов — обязательно 512, такое же, как было.	

		Ну и выведем модель после замены слоя. Прокрутим вниз. Видно, что слой действительно поменялся: здесь 70 нейронов. Осталось заморозить свёрточные слои. Мы только что поняли, что доступ к каждому слою можно получить по его названию. Мы могли бы пройтись по слоям по их названиям и заморозить каждый отдельно. Но так делать долго. Смотрите, доступ к слоям сети тоже можно получить с помощью метода children. Этот метод выдаёт список всех элементов нейросети. Давайте выведем его размер. Десять. Но в модели resnet-18 должно быть 18 слоёв, почему тут 10? Ответ прост. Ещё в самом начале мы обсуждали, что некоторые слои модели объединены в блоки. Children выдаёт список верхнеуровневых блоков, которые есть у сети. Если слой находится вне блока, как этот, то он будет из элементов списка children. Если слои объединены в блоки, то элементом списка children будет весь блок. Выведем, к примеру, пятый элемент списка children. Это как раз блок. Поэтому элементов в списке children меньше, чем слоёв. Давайте в нашей нейросети заморозим все свёрточные слои и оставим обучаться только последний полносвязный слой. Так тоже можно — замораживать вообще все свёрточные слои, а не несколько первых. Мы предполагаем, что при обучении на ImageNet все фильтры выучили достаточно полезные паттерны, чтобы классификатор — полносвязный слой — смог успешно обучиться по этим паттернам классифицировать породы собак. Мы, конечно, можем ошибаться, и, возможно, оптимально было бы заморозить другое количество слоёв. Но это мы поймём только в ходе экспериментов.	

		Как мы будем замораживать слои? Давайте пройдёмся по model.children() и заморозим все веса всех слоёв всех блоков, кроме последнего. Последний элемент model.children — это и есть полносвязный слой fc. Проходимся по model.children. Пока мы не дошли до последнего, десятого элемента, делаем вот что: проходимся по всем параметрам текущего блока layer и для каждого параметра ставим атрибут requires_grad равно False. Этот атрибут определяет, будет ли для этого параметра в процессе обучения сети вычисляться градиент. Если градиент вычисляться не будет, то и обновляться этот параметр тоже не будет. Так мы заморозим все параметры, то есть все веса всех слоёв всех блоков до последнего. Теперь можно обучать нашу нейросеть. Перенесём её на GPU, как мы делали это в прошлой практике. Определим оптимизатор и лосс-функцию. Заведём TensorBoard для наблюдения за ходом обучения. В настройках поставим галочку для автоматического обновления результатов. Весь код в этих ячейках точно такой же, как и в прошлом модуле. Ячейка с функциями train и evaluate точно такая же. X_batch и y_batch уже перенесены здесь на GPU. Запустим обучение нашей модели на трёх эпохах. Идём выше в TensorBoard смотреть, как обучается модель.	

		Модель обучилась. Мы можем проверить её качество на тренировочной и тестовой выборках, как мы это делали в прошлый раз. Запустим. Вот мы и прошли весь путь от загрузки предобученной нейросети и датасета с диска до дообучения нейросети на новом датасете. Теперь вы можете самостоятельно дообучать разные нейросети под новые задачи. При этом использовать любые датасеты из интернета: мы научились с ними работать. Вы можете экспериментировать с кодом самостоятельно: попытаться улучшить его, чтобы он выдавал лучший результат. Можно попробовать загрузить другую нейросеть, например, VGG или другой вариант ResNet. Можно попробовать заморозить другое количество слоёв. Вы можете попытаться построить свою свёрточную нейросеть для решения задачи классификации пород собак. Сравнить: будет ли ваша нейросеть решать задачу классификации собак лучше, чем предобученные нейросети из PyTorch. В этой ячейке вы можете, как и в прошлый раз, загрузить графики TensorBoard в tensorboard dev. Последнее, о чём мы поговорим, — сохранение и загрузка весов модели. Обученные модели можно сохранять в файл и загружать обратно из файла. Сохранить модель очень просто: пишем torch.save, на вход подаём переменную, в которой находится модель, и путь к файлу, в который вы хотите эту модель сохранить. Обычно файлы с моделями имеют расширение pt. Назовём наш файл model_dogs и запустим ячейку.	

		Теперь, если мы посмотрим в файлы, то увидим, что появился файл model_dogs.pt. Тут лежит наша модель. Этот файл можно скачать на компьютер и загрузить на какой-то другой сервер. Загрузить модель из файла тоже очень просто. Для этого есть функция torch.load. В неё передаём путь к файлу с моделью. Загрузим нашу модель в новую переменную model_new. Убедимся, что модель загрузилась. Она должна совпадать с model. Давайте протестируем model_new на тестовом датасете. Запустим. Видим, что результат на тесте тот же, что был у model. С загруженной моделью можно работать дальше и дообучить её на нашем датасете собак. Вот так просто можно сохранять и загружать модели нейросетей в PyTorch.	
Закрепление изученного материала	15 мин.	**Вопросы для обсуждения** ● Как с помощью ImageFolder научить загружать внешние датасеты картинок? ● Для чего используется трансформация картинок?	Педагог организует беседу по вопросам

Этап подведения итогов занятия (рефлексия)	8 мин.	**Вопросы для обсуждения** ● Чему я научился? ● С какими трудностями я столкнулся? ● Какие вопросы у тебя остались? Что осталось непонятным?	Педагог способствует размышлению обучающихся над вопросами
Информация о домашнем задании, инструктаж по его применению	5 мин.	-	

Рекомендуемые ресурсы для дополнительного изучения:

1. Transfer Learning и PyTorch в Python. [Электронный ресурс] – Режим доступа: https://tonais.ru/library/transfer-learning-pytorch-python
2. Transfer Learning: как быстро обучить нейросеть на своих данных. [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/binarydistrict/blog/428255/
3. Погружение в свёрточные нейронные сети: передача обучения (transfer learning). [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/467967/