Создание нейронной сети классификации одежды
ЛАБОРАТОРНАЯ РАБОТА

http://www.kimrt.ru
Онлайн-курс
Суперкомпьютерные технологии в задачах моделирования

Задача классификации одежды Fashion-MNIST — это новый стандартный набор данных, используемый в компьютерном зрении и глубоком обучении. Хотя набор данных относительно прост, его можно использовать в качестве основы для обучения и практики разработки, оценки и использования глубоких сверточных нейронных сетей. Это включает в себя разработку надежной системы тестирования для оценки производительности модели, исследование улучшений модели, а также сохранение модели и ее загрузка, чтобы делать прогнозы на основе новых данных [1, 2].

Набор данных Fashion-MNIST предлагается в качестве более сложной замены набора данных MNIST. Сначала установим TensorFlow и импортируем его, а также установим дополнительные библиотеки. Затем мы загружаем fashion-mnist, который является одним из наборов данных Keras.

Листинг 1. Загрузка размеченного набора данных

```
# установка библиотек
import tensorflow as tf
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
%matplotlib inline

#Загрузка датасета
fashion_mnist = keras.datasets.fashion_mnist

# Разделение датасета на тренировочные и тестовые данные
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

Это набор данных, состоящий из 60000 небольших квадратных изображений (размером 28x28 пикселей) десяти видов предметов одежды, таких как обувь, футболки, платья и многое другое. Сопоставление всех целых чисел от 0 до 9 с метками классов показано ниже:

- 0: T-shirt/top,
- 1: Trouser,

- 2: Pullover,
- 3: Dress,
- 4: Coat,
- 5: Sandal,
- 6: Shirt,
- 7: Sneaker,
- 8: Bag,
- 9: Ankle boot.

После данных операций необходимо создать массив class_names, содержащий значения видов одежды. Также создается график первых пятнадцати изображений в наборе данных, показывающий, что изображения действительно представляют собой фотографии предметов одежды в градациях серого (см. листинг 2).

Листинг 2. Визуализация датасета

```
  def plot(images, labels, predictions=None):
      n_cols = min(5, len(images))
      n_rows = math.ceil(len(images) / n_cols)
      fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols+3, n_rows+4))

      if predictions is None:
          predictions = [None] * len(labels)

      for i, (x, y_true, y_pred) in enumerate(zip(images, labels, prediction
s)):
          ax = axes.flat[i]
          ax.imshow(x, cmap=plt.cm.binary)

          ax.set_title(f"lbl: {class_names[y_true]}")

          if y_pred is not None:
              ax.set_xlabel(f"pred: {y_pred}")

          ax.set_xticks([])
          ax.set_yticks([])
  plot(x_train[:15], y_train[:15])
```

В цветовом режиме изображения «Градации серого» используется до 256 оттенков серого [3]. Каждый пиксель изображения в градациях серого содержит значение яркости в диапазоне от 0 (черный) до 255 (белый). Перед непосредственным созданием модели необходимо провести нормализацию данных следующими строчками кода.

```
x_train = x_train / 255
x_test = x_test / 255
```

Таким образом, интенсивность серого цвета пикселей определяется диапазоном значений от 0 до 1 (где 0 – черный, 1 – белый) в массивах *x_train* и *x_test*.

Базовым строительным блоком нейронной сети является слой. Слои извлекают представления из данных, переданных им. Большая часть сетей глубокого обучения состоит из связанных вместе простых слоев.

```
# Создание модели
model = tf.keras.models.Sequential([
# ЗДЕСЬ ПРОПИСЫВАЮТСЯ СЛОИ МОДЕЛИ ])
```

Первый слой в этой сети, tf.keras.layers.Flatten, преобразует формат изображений из 2х-мерного массива (28 на 28 пикселей) в одномерный массив 28 * 28 = 784 пикселей. Рассматривайте этот слой как разложенные из стека и построенные в линию слои пикселей в изображении. Этот слой не имеет параметров для обучения, это только преобразование данных. После того как пиксели преобразованы сеть состоит из последовательности двух tf.keras.layers.Dense слоев. Это плотносвязанные или полностью связанные нейронные слои. Первый Dense слой имеет 512 узлов (или нейронов). Второй (последний) слой – 10-узловой софтмакс слой. Он возвращает массив десяти расчетам вероятностей, которые в сумме равны 1. Каждый узел содержит значение, которое показывает вероятность того, что текущее изображение принадлежит одному из 10 классов.

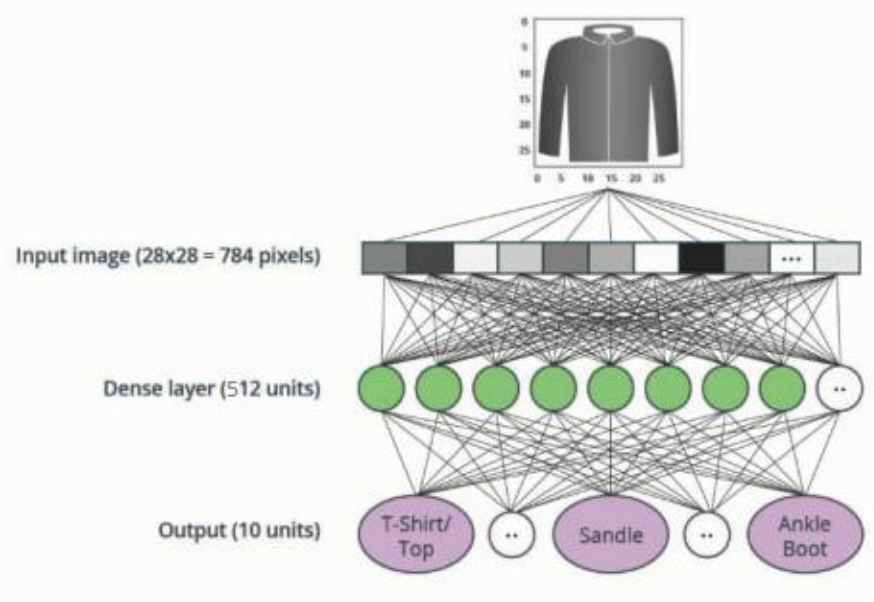

Рис. 1. Схематический рисунок сети

Перед тем как модель будет готова к тренировке ей потребуются еще некоторые настройки. Настройки добавляются на этапе компиляции модели:

Функция потери – она измеряет погрешность модели (loss) в ходе тренировки. Мы хотим минимизировать эту функцию, чтобы управлять моделью в правильном направлении.

Оптимизатор – управляет тем, как модель обновляется (ее веса и скорость обучения), основываясь на данных, которые видит, и функции потери.

Метрики – используются для наблюдения за тренировочным и тестовым шагами.

```
    # компилируем модель
    model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

Тренировка модели нейронной сети требует следующих шагов:

- передать модели тренировочные данные – в этом примере, train_images и train_labels массивы;
- модель обучается связывать изображения с метками;

Прогнозы делаются на тестовом наборе – в этом примере, x_test массив. Проверяем, что прогнозы соответствуют меткам из y_test массива. Чтобы начать тренировку, вызовем model.fit метод – модель "приспосабливается" ("fit") к тренировочным данным. Этому методу мы передаем сами тренировочные данные (x_train и y_train), количество эпох тренировки (epochs), использование многопоточности (use_multiprocessing), размер поступающих партий картинок (batch_size), а также максимальное количество процессов (workers), которые необходимо запустить при использовании многопоточности.

```
model = model.fit(x_train, y_train, epochs=#, use_multiprocessing=True, batc
h_size=#, workers=3)
```

Убедитесь, что после тренировки модели ее точность не меньше 90%.

```
Epoch 1/10
938/938 [==============================] - 7s 7ms/step - loss: 0.4894 - accuracy: 0.8262
Epoch 2/10
938/938 [==============================] - 8s 9ms/step - loss: 0.3650 - accuracy: 0.8674
Epoch 3/10
938/938 [==============================] - 6s 6ms/step - loss: 0.3216 - accuracy: 0.8815
Epoch 4/10
938/938 [==============================] - 6s 6ms/step - loss: 0.3007 - accuracy: 0.8889
Epoch 5/10
938/938 [==============================] - 6s 6ms/step - loss: 0.2818 - accuracy: 0.8961
Epoch 6/10
938/938 [==============================] - 6s 6ms/step - loss: 0.2676 - accuracy: 0.9001
Epoch 7/10
938/938 [==============================] - 6s 6ms/step - loss: 0.2535 - accuracy: 0.9051
Epoch 8/10
938/938 [==============================] - 6s 6ms/step - loss: 0.2423 - accuracy: 0.9093
Epoch 9/10
938/938 [==============================] - 7s 7ms/step - loss: 0.2331 - accuracy: 0.9121
Epoch 10/10
938/938 [==============================] - 6s 6ms/step - loss: 0.2214 - accuracy: 0.9171
```

Рис. 2. Результаты тренировки модели

На рис. 2 можно заметить, что на каждом этапе тренировки значение потерь уменьшается, а точность постепенно возрастает. После того, как мы убедились, что модель обучена, приступим к предсказанию результатов.

Листинг 3. Предсказание результатов

```
preds = np.argmax(model.predict(x_test), axis=-1)

# выбираем случайные 20 картинок из датасета
rand_idxs = np.random.permutation(len(x_test))[:20]

pr = [class_names[i] for i in preds[rand_idxs]]
```

```
plot(x_test[rand_idxs], y_test[rand_idxs], pr)
```

Рис. 2. Результат предсказаний

Под меткой *lbl* указывается истинный класс одежды, под меткой *pred* – предсказание, основанное на обученной модели. Как можно заметить, в некоторых случаях две эти метки не сходятся. Это нормально, так как наша модель не обучена на 100%, поэтому она может допускать ошибки. Повысить точность модели можно при увеличении количества эпох обучения.

Выполните следующие задания.

Задание 1. Напишите нейронную сеть классификатора одежды, использовав при тренировке модели 10 эпох и размер партий, равный 64. Убедитесь, что точность модели не меньше 90%.

Задание 2. Постройте в Excel график зависимости скорости обучения модели при использовании и не использовании многопоточности на основе процессов (параметр use_multiprocessing) от величины параметра batch_size.

Ниже представлен пример реализации такого графика при использовании Google Colaboratory CPU: Intel(R) Xeon(R) CPU @ 2.20GHz, GPU: Tesla T4.

multiprocessing	batch_size				
	32	64	128	256	512
True	41,3	23	11	6,3	3,9
False	41,6	23	13	6,5	4,1

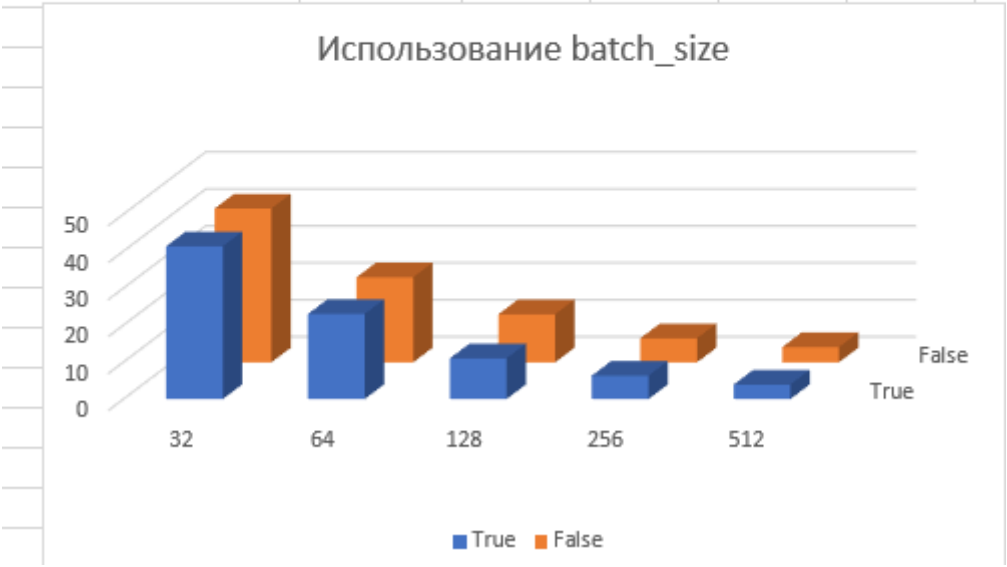

Рис. 3. Анализ влияния параметров multiprocessing и batch_size

http://www.kimrt.ru/index/course_stm/0-24

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. CoRR, abs/1708.07747. http://arxiv.org/abs/1708.07747
2. Dataset Card for FashionMNIST. https://huggingface.co/datasets/fashion_mnist
3. Цветовые режимы изображения. https://helpx.adobe.com/ru/photoshop-elements/using/using-image-modes-color-tables.html