Курс «Глубокое обучение в компьютерном зрении»

Свёрточные нейронные сети (СНС)

Практическое задание

Реализовать и обучить (с нуля) СНС для задачи классификации изображений на датасете CIFAR-10

Библиотеки: [Python, Tensorflow]

Выполнил ***Соковнин ИЛ***

The CIFAR-10 dataset

https://www.cs.toronto.edu/~kriz/cifar.html

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

```
%tensorflow_version 2.x
```

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

import tensorflow as tf
import random
```

cifar10 - цветной датасет 10 классов, картинки 32x32 с цветными изображениями

```
# Название классов из набора cifar10
classes=['самолет', 'автомобиль', 'птица', 'кот', 'олень', 'собака', 'лягушка', 'лошадь', 'корабль', 'грузовик']
```

Загрузка и подготовка датасета CIFAR-10

```
(train_x, train_y), (test_x, test_y) =  tf.keras.datasets.cifar10.load_data()

# Преобразуем картинки в 4d-тензор:
# -1 - вычисли размерность batcha сам
# 32x32 - пространственное измерение
# 3 канала
train_x = train_x.reshape(-1, 32, 32, 3).astype(np.float32) / 255.
test_x = test_x.reshape(-1, 32, 32, 3).astype(np.float32) / 255.
```

Сохранение...

```
print(train_x.shape, train_x.dtype)
print(test_x.shape, test_x.dtype)
print(train_y.shape, train_y.dtype)
print(test_y.shape, test_y.dtype)
```

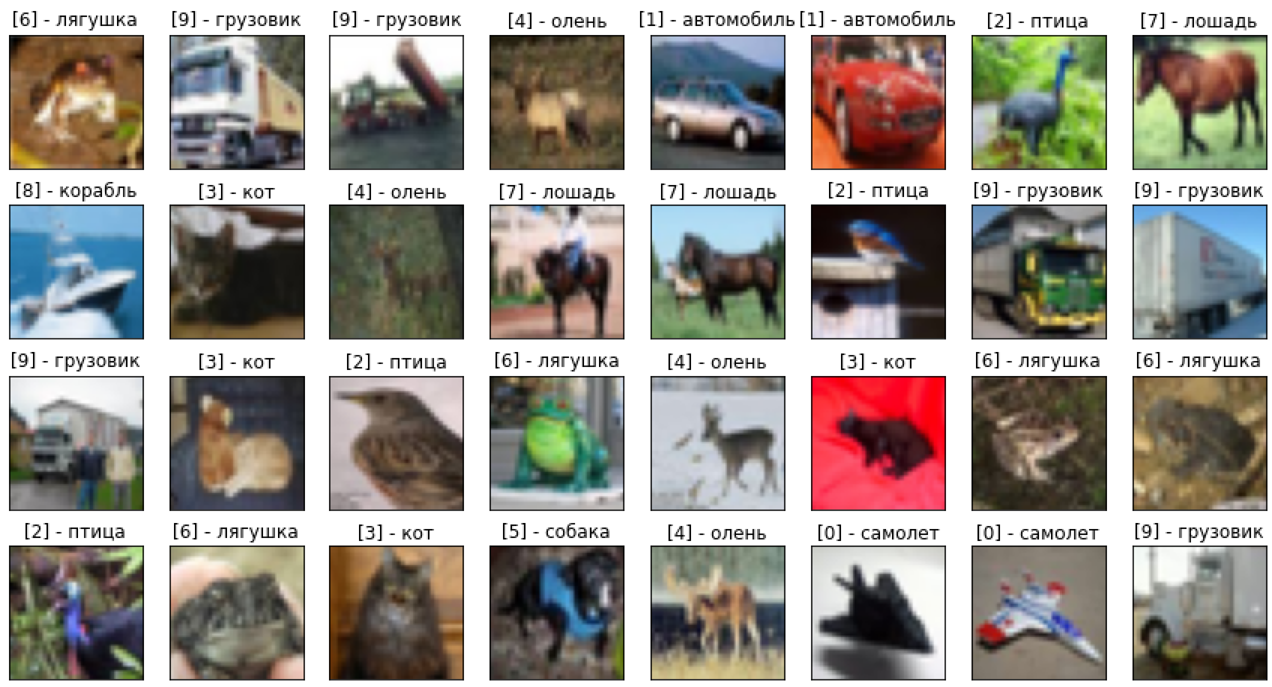
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 13s 0us/step
170508288/170498071 [==============================] - 13s 0us/step
(50000, 32, 32, 3) float32
(10000, 32, 32, 3) float32
(50000, 1) int32
(10000, 1) int32
```

```
np.unique(train_y)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int32)
```

Визуализация датасета cifar10

```
# Возьмём несколько образцов из train и нарисуем их
some_samples = train_x[:32, ...]
some_samples_y = train_y[:32, ...]

fig = plt.figure(figsize=(15, 8))
for j in range(some_samples.shape[0]):
    ax = fig.add_subplot(4, 8, j+1)
    ax.imshow(some_samples[j,:,:])
    plt.title(str(train_y[j]) + ' - ' + classes[int(train_y[j])])
    plt.xticks([]), plt.yticks([])
plt.show()
```




```
# # First 25 images in the train dataset
# plt.figure(figsize = (13, 15))
# for i in range(25):
#     image = np.array(train_x[i,:,:])
#     plt.subplot(5, 5, i+1)
#     plt.title(str(train_y[i]) + ' - ' + classes[int(train_y[i])])
#     plt.imshow(image)
#     # plt.axis('off')
# plt.show()
```

Вариант 1

Создание модели CNN

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPool2D((2, 2), (2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D((2, 2), (2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    # tf.keras.layers.Dense(10, activation='softmax')
    tf.keras.layers.Dense(10)
])
```

Архитектура модели

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

                              None, 15, 15, 32)        0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

Подготовка к обучению

Компиляция модели

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Обучение модели

```
NUM_EPOCHS = 10

model.fit(train_x, train_y, epochs=NUM_EPOCHS, validation_data=(test_x, test_y))
```

```
Epoch 1/10
1563/1563 [==============================] - 19s 5ms/step - loss: 1.5058 - accuracy: 0.4547 - val_loss: 1.2510 - val_accuracy: 0.5529
Epoch 2/10
1563/1563 [==============================] - 7s 5ms/step - loss: 1.1437 - accuracy: 0.5968 - val_loss: 1.0468 - val_accuracy: 0.6300
Epoch 3/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.9903 - accuracy: 0.6536 - val_loss: 0.9651 - val_accuracy: 0.6623
Epoch 4/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8925 - accuracy: 0.6860 - val_loss: 0.9486 - val_accuracy: 0.6698
Epoch 5/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.8251 - accuracy: 0.7122 - val_loss: 0.8772 - val_accuracy: 0.6966
Epoch 6/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.7698 - accuracy: 0.7327 - val_loss: 0.8834 - val_accuracy: 0.6947
Epoch 7/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.7212 - accuracy: 0.7468 - val_loss: 0.9040 - val_accuracy: 0.6932
Epoch 8/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.6815 - accuracy: 0.7617 - val_loss: 0.8481 - val_accuracy: 0.7148
Epoch 9/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.6421 - accuracy: 0.7745 - val_loss: 0.9092 - val_accuracy: 0.7029
Epoch 10/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.6048 - accuracy: 0.7886 - val_loss: 0.8747 - val_accuracy: 0.7079
<keras.callbacks.History at 0x7f5769ae3650>
```

Оценка качества модели

```
model.evaluate(test_x, test_y)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.8747 - accuracy: 0.7079
[0.8747029304504395, 0.7078999876976013]
```

Пример инференса модели

```
sample = test_x[0, ...]
prediction = model(sample[None, ...])[0]
print(prediction)
```

```
tf.Tensor(
[-1.3040925  -5.1617813   0.23061705  3.9649518  -4.893924    3.7847888
 -0.79691887 -4.315691   -1.0234367  -2.9266372 ], shape=(10,), dtype=float32)
```

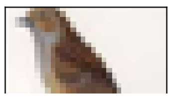
Функция для инференса (вывод) и отображения результата предсказания

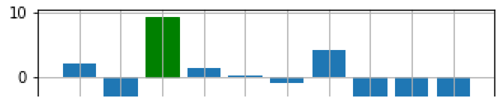
```
def test_digit(sample):

    prediction = model(sample[None, ...])[0]  # Распределение вероятностей по классам
    ans = np.argmax(prediction)  # Получаем индекс максимальной вероятности.

    # fig = plt.figure(figsize=(12,4))
    fig = plt.figure(figsize=(12,2))

    ax = fig.add_subplot(1, 2, 1)
    ax.imshow(sample[:,:])
    plt.xticks([]), plt.yticks([])

    ax = fig.add_subplot(1, 2, 2)
    bar_list = ax.bar(np.arange(10), prediction, align='center')
    bar_list[ans].set_color('g')
    ax.set_xticks(np.arange(10))
    ax.set_xlim([-1, 10])
    ax.grid(True)

    plt.show()

                                     mat(ans) + ' - ' + classes[ans])
```

Сохранение…

Запуск предсказания для изображения случайной цифры из CIFAR-10

```
import random
idx = random.randint(0, test_x.shape[0])
sample = test_x[idx, ...]
test_digit(sample)

print('True Answer: {}'.format(test_y[idx]) + ' - ' + classes[int(test_y[idx])])
```






```
# classes=['самолет', 'автомобиль', 'птица', 'кот', 'олень', 'собака', 'лягушка', 'лошадь', 'корабль', 'грузовик']
```

Вариант 2

Создание пайплайна данных

```
NUM_EPOCHS = 10
BATCH_SIZE = 128

train_ds = tf.data.Dataset.from_tensor_slices((train_x, train_y))
train_ds = train_ds.shuffle(buffer_size=train_x.shape[0])
train_ds = train_ds.repeat(NUM_EPOCHS)
train_ds = train_ds.batch(BATCH_SIZE)
```

Создание модели CNN

```
class Model(tf.keras.Model):

    def __init__(self):
        # Конструктор.

        super(Model, self).__init__()

        self.conv1 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3), padding='same')
        self.conv2 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')
        self.conv3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')
        self.fc1 = tf.keras.layers.Dense(64, activation='relu')
        self.fc2 = tf.keras.layers.Dense(10, activation=None)
        self.max_pool = tf.keras.layers.MaxPooling2D((2, 2), (2, 2))
        self.flatten = tf.keras.layers.Flatten()

    def call(self, inp):

        out = self.conv1(inp)
        out = self.max_pool(out)
        out = self.conv2(out)
        out = self.max_pool(out)
        out = self.conv3(out)
        out = self.conv3(out)
        out = self.flatten(out)
        out = self.fc1(out)
        out = self.fc2(out)

        return out

model = Model()
```

Функция потерь и функция вычисления точности

```
def loss(logits, labels):
    return tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=logits, labels=np.ravel(labels)))

def accuracy(logits, labels):
    predictions = tf.argmax(logits, axis=1, output_type=tf.int32)
    return tf.reduce_mean(tf.cast(tf.equal(predictions, np.ravel(labels)), dtype=tf.float32))
```

Подготовка к обучению

```
LEARNING_RATE = 0.001  # Скорость обучения
optimizer = tf.keras.optimizers.Adam(LEARNING_RATE)
writer = tf.summary.create_file_writer('logs/adam')
```

Цикл обучения модели

Сохранение…

```
%%time
                                              numerate(train_ds):

    # Forward
    with tf.GradientTape() as tape:
        logits = model(images)
        loss_value = loss(logits, labels)

    # Backward
    grads = tape.gradient(loss_value, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    # Calc and display loss/accuracy
    if iteration % 200 == 0:
        test_logits = model(test_x[:256, ...])
        accuracy_value = accuracy(test_logits, test_y[:256, ...])

        print("[%4d] Accuracy: %5.2f %%" % (
            iteration, accuracy_value.numpy()*100))
```

```
        with writer.as_default():
            tf.summary.scalar('accuracy', accuracy_value, iteration)
            tf.summary.scalar('loss', loss_value, iteration)
```

```
[   0] Accuracy: 12.11 %
[ 200] Accuracy: 43.36 %
[ 400] Accuracy: 53.91 %
[ 600] Accuracy: 58.20 %
[ 800] Accuracy: 61.72 %
[1000] Accuracy: 58.59 %
[1200] Accuracy: 65.62 %
[1400] Accuracy: 68.75 %
[1600] Accuracy: 69.92 %
[1800] Accuracy: 70.31 %
[2000] Accuracy: 69.14 %
[2200] Accuracy: 71.88 %
[2400] Accuracy: 71.88 %
[2600] Accuracy: 73.44 %
[2800] Accuracy: 71.88 %
[3000] Accuracy: 73.44 %
[3200] Accuracy: 73.83 %
[3400] Accuracy: 75.00 %
[3600] Accuracy: 76.95 %
[3800] Accuracy: 76.17 %
CPU times: user 48.6 s, sys: 921 ms, total: 49.5 s
Wall time: 49 s
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_3 (Conv2D)           multiple                  896       
                                                                 
 conv2d_4 (Conv2D)           multiple                  18496     
                                                                 
 conv2d_5 (Conv2D)           multiple                  36928     
                                                                 
 dense_2 (Dense)             multiple                  262208    
                                                                 
 dense_3 (Dense)             multiple                  650       
                                                                 
 max_pooling2d_2 (MaxPooling  multiple                 0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         multiple                  0         
                                                                 
=================================================================
Total params: 319,178
Trainable params: 319,178
Non-trainable params: 0
_________________________________________________________________
```

Оценка качества модели

```
%%time

# Делаем прямое распространение на всей тестовой выборке
# Получаем все ответы на всей тестовой выборке
test_logits = model(test_x)
accuracy_value = accuracy(test_logits, test_y).numpy()
print("Final Accuracy: %5.2f %%" % (accuracy_value * 100))
```

```
Final Accuracy: 71.89 %
CPU times: user 1.73 s, sys: 55.8 ms, total: 1.79 s
Wall time: 1.77 s
```

```
%load_ext tensorboard
%tensorboard --logdir logs  # logs - дирректория с нашими summaries
```

TensorBoard SCALARS TIME SERIES INACTIVE

Show data download links
Ignore outliers in chart scaling

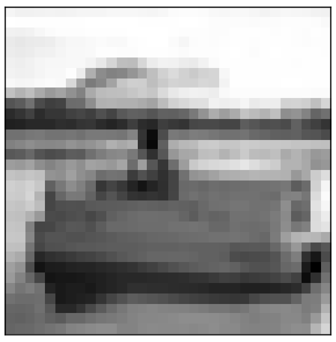
Filter tags (regular expressions supported)

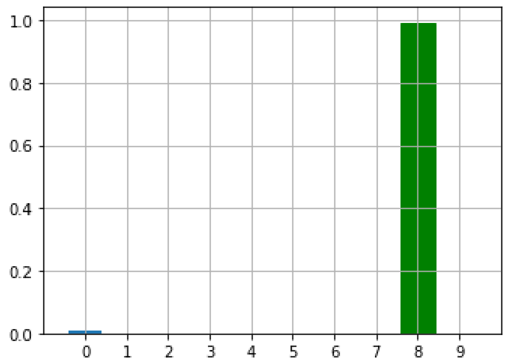
accuracy

```
def test_item(sample):
    # Собственная функция тестирования картинки.
    # Смотрим какое у нас распределение по классам.

    logits = model(sample[None, ...])[0]
    prediction = tf.nn.softmax(logits)
    ans = np.argmax(prediction)

    fig = plt.figure(figsize=(12,4))

    ax = fig.add_subplot(1, 2, 1)
    ax.imshow(sample[:,:,0], cmap='gray')
    plt.xticks([]), plt.yticks([])

    ax = fig.add_subplot(1, 2, 2)
    bar_list = ax.bar(np.arange(10), prediction, align='center')
    bar_list[ans].set_color('g')
    ax.set_xticks(np.arange(10))
    ax.set_xlim([-1, 10])
    ax.grid(True)

    plt.show()

    print('Predicted: {}'.format(ans) + ' - ' + classes[ans])
```

Запуск предсказания для изображения случайной цифры из CIFAR-10

```
import random
idx = random.randint(0, test_x.shape[0])
sample = test_x[idx, ...]
test_item(sample)

print('True Answer: {}'.format(test_y[idx]) + ' - ' + classes[int(test_y[idx])])
```






```
Predicted: 8 - корабль
True Answer: [8] - корабль
```

Вывод: Путем подбора гиперпараметров удалось добиться приемлемой точности работы нейросети.

Сохранение...

0 сек. выполнено в 15:34