Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №3 по курсу «Проектирование интеллектуальных систем»

Выполнил:
Хотин П.Ю.
ИУ5-24М

Москва, 2020 год

Импорт библиотек:

```
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os
from keras.constraints import maxnorm
from keras.optimizers import SGD
```

```
Using TensorFlow backend.
```

```
!mkdir ./sample_data/models
```

```
os.getcwd()
```

```
‘/content'
```

Скачиваем датасет и приводим к виду для обучения с помощью нейронной сети:

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train = x_train / 255.0
x_test = x_test / 255.0
```

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Задаем количество эпох и объем батча:

```
batch_size = 32
num_classes = y_test.shape[1]
```

```
epochs = 25
data_augmentation = True
num_predictions = 20
save_dir = os.path.join(os.getcwd(), '/sample_data/models')
model_name = ‘keras_cifar10_trained_model.h5'
```

Создание базовой модели:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same',
activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), input_shape=(32, 32, 3), padding='same',
activation='relu', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=[‘accuracy'])
```

Описание модели:

```
print(model.summary())

Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_37 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
```

```
dropout_31 (Dropout)         (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 8, 8, 128)         0
_________________________________________________________________
dropout_32 (Dropout)         (None, 8, 8, 128)         0
_________________________________________________________________
flatten_17 (Flatten)         (None, 8192)              0
_________________________________________________________________
dense_33 (Dense)             (None, 512)               4194816
_________________________________________________________________
dropout_33 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_34 (Dense)             (None, 10)                5130
=================================================================
Total params: 4,293,194
Trainable params: 4,293,194
Non-trainable params: 0
_________________________________________________________________
None
```

Начало тренировки модели:

```
model.fit(x_train, y_train, validation_data=(x_test, y_test),
epochs=epochs, batch_size=batch_size)
# Final evaluation of the model
scores = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/25
50000/50000 [==============================] - 412s 8ms/step - loss:
1.7011 - accuracy: 0.3824 - val_loss: 1.4105 - val_accuracy: 0.4999
Epoch 2/25
```

```
50000/50000 [==============================] - 413s 8ms/step - loss:
1.2946 - accuracy: 0.5350 - val_loss: 1.1536 - val_accuracy: 0.5784
Epoch 3/25
50000/50000 [==============================] - 412s 8ms/step - loss:
1.1025 - accuracy: 0.6065 - val_loss: 0.9935 - val_accuracy: 0.6567
Epoch 4/25
50000/50000 [==============================] - 412s 8ms/step - loss:
0.9747 - accuracy: 0.6526 - val_loss: 0.9299 - val_accuracy: 0.6743
Epoch 5/25
50000/50000 [==============================] - 412s 8ms/step - loss:
0.8670 - accuracy: 0.6939 - val_loss: 0.8519 - val_accuracy: 0.7036
Epoch 6/25
50000/50000 [==============================] - 411s 8ms/step - loss:
0.7864 - accuracy: 0.7231 - val_loss: 0.8092 - val_accuracy: 0.7198
Epoch 7/25
50000/50000 [==============================] - 414s 8ms/step - loss:
0.7189 - accuracy: 0.7486 - val_loss: 0.7791 - val_accuracy: 0.7269
Epoch 8/25
50000/50000 [==============================] - 412s 8ms/step - loss:
0.6600 - accuracy: 0.7678 - val_loss: 0.7483 - val_accuracy: 0.7426
Epoch 9/25
50000/50000 [==============================] - 411s 8ms/step - loss:
0.6061 - accuracy: 0.7825 - val_loss: 0.7405 - val_accuracy: 0.7456
Epoch 10/25
50000/50000 [==============================] - 412s 8ms/step - loss:
0.5632 - accuracy: 0.8014 - val_loss: 0.7311 - val_accuracy: 0.7483
Epoch 11/25
50000/50000 [==============================] - 414s 8ms/step - loss:
0.5259 - accuracy: 0.8151 - val_loss: 0.7243 - val_accuracy: 0.7519
Epoch 12/25
50000/50000 [==============================] - 412s 8ms/step - loss:
0.4853 - accuracy: 0.8275 - val_loss: 0.7293 - val_accuracy: 0.7511
Epoch 13/25
50000/50000 [==============================] - 413s 8ms/step - loss:
0.4565 - accuracy: 0.8381 - val_loss: 0.7307 - val_accuracy: 0.7530
Epoch 14/25
50000/50000 [==============================] - 413s 8ms/step - loss:
0.4276 - accuracy: 0.8470 - val_loss: 0.7279 - val_accuracy: 0.7536
Epoch 15/25
```

```
50000/50000 [==============================] - 411s 8ms/step - loss:
0.3999 - accuracy: 0.8572 - val_loss: 0.7274 - val_accuracy: 0.7562
Epoch 16/25
50000/50000 [==============================] - 411s 8ms/step - loss:
0.3810 - accuracy: 0.8650 - val_loss: 0.7386 - val_accuracy: 0.7563
Epoch 17/25
50000/50000 [==============================] - 411s 8ms/step - loss:
0.3557 - accuracy: 0.8738 - val_loss: 0.7279 - val_accuracy: 0.7630
Epoch 18/25
50000/50000 [==============================] - 412s 8ms/step - loss:
0.3352 - accuracy: 0.8812 - val_loss: 0.7345 - val_accuracy: 0.7614
Epoch 19/25
50000/50000 [==============================] - 413s 8ms/step - loss:
0.3209 - accuracy: 0.8844 - val_loss: 0.7475 - val_accuracy: 0.7601
Epoch 20/25
50000/50000 [==============================] - 415s 8ms/step - loss:
0.3025 - accuracy: 0.8923 - val_loss: 0.7454 - val_accuracy: 0.7610
Epoch 21/25
50000/50000 [==============================] - 416s 8ms/step - loss:
0.2839 - accuracy: 0.8996 - val_loss: 0.7414 - val_accuracy: 0.7646
Epoch 22/25
50000/50000 [==============================] - 414s 8ms/step - loss:
0.2681 - accuracy: 0.9053 - val_loss: 0.7611 - val_accuracy: 0.7642
Epoch 23/25
50000/50000 [==============================] - 415s 8ms/step - loss:
0.2570 - accuracy: 0.9075 - val_loss: 0.7567 - val_accuracy: 0.7642
Epoch 24/25
50000/50000 [==============================] - 414s 8ms/step - loss:
0.2507 - accuracy: 0.9127 - val_loss: 0.7604 - val_accuracy: 0.7675
Epoch 25/25
50000/50000 [==============================] - 413s 8ms/step - loss:
0.2383 - accuracy: 0.9164 - val_loss: 0.7704 - val_accuracy: 0.7663
Accuracy: 76.63%
```

Ответы на контрольные вопросы:

1) ***Что такое свертка?***

Двумерная свертка — это довольно простая операция: начинаем с ядра, представляющего из себя матрицу весов (weight matrix). Ядро “скользит” над двумерным изображением, поэлементно выполняя операцию умножения с той частью входных данных, над которой

оно сейчас находится, и затем суммирует все полученные значения в один выходной пиксель.

Ядро повторяет эту процедуру с каждой локацией, над которой оно “скользит”, преобразуя двумерную матрицу в другую все еще двумерную матрицу признаков. Признаки на выходе являются взвешенными суммами (где веса являются значениями самого ядра) признаков на входе, расположенных примерно в том же месте, что и выходной пиксель на входном слое.

2) ***Напишите математическую операцию свертки.***
convolution1.png
convolution2.png

3) ***Какие свойства сверточного слоя?***
Разреженность взаимодействия нейронов
Разделяемые (общие) параметры
Эквивариантность представления
Работа со входом различного размера

4) **Сколько этапов в сверточном слое? Какие?**

1. Свертка входа с множеством ядер
2. Пропускание откликов через нелинейную активацию (детектор)
3. Объединение соседних активаций (pooling)

5) **Что такое регуляризация? Зачем она нужна?**
Регуляризация — процесс уменьшения переобучения путем забывания определенных сигналов из тренировочных данных. Регуляризация помогает повысить точность и уменьшить вероятность переобучения. Тем самым разница между тренировочными данными и тестовыми будет уменьшена.

6) **Как вид регуляризации использовался в лабораторной?**
В лабораторной используется dropout(исключение). Он характеризуется исключением определённого процента случайных нейронов во время обучения нейронной сети. Это очень эффективный способ усреднения моделей внутри нейронной сети. В результате более обученные нейроны получают в сети больший вес. Такой приём значительно увеличивает скорость обучения, качество обучения на тренировочных данных, а также повышает качество предсказаний модели на новых тестовых данных.

Список литературы

[1] Google. Tensorflow. 2018. Apr. url - https://www.tensorflow.org/api_docs/python/tf/train/Saver.

[2] Google. TensorBoard. 2018. Apr. url - https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard.