**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:**

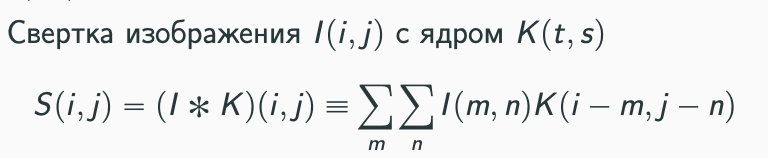
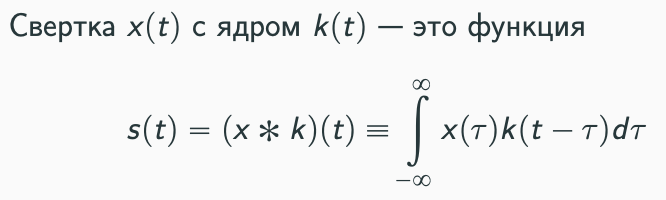
model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=epochs, batch\_size=batch\_size)  
# Final evaluation of the model  
scores = model.evaluate(x\_test, y\_test, verbose=0)  
print("Accuracy: %.2f%%" % (scores[1]\*100))

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.

1. ***Напишите математическую операцию свертки.***



1. ***Какие свойства сверточного слоя?***  
   Разреженность взаимодействия нейронов  
   Разделяемые (общие) параметры  
   Эквивариантность представления  
   Работа со входом различного размера
2. **Сколько этапов в сверточном слое? Какие?**
3. Свертка входа с множеством ядер
4. Пропускание откликов через нелинейную активацию (детектор)
5. Объединение соседних активаций (pooling)
6. **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.
7. **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.