

Хотин Павел ИУ5-24М

```
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'

```
(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)
```

```
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'
```

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='r
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=m
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), input_shape=(32, 32, 3), padding='same', activation='
```

```

model.add(Conv2D(128, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu'))
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'))

```

```

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

```

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))

```

Model: "sequential_18"

Train on 50000 samples, validate on 10000 samples

```
Epoch 1/25
50000/50000 [=====] - 412s 8ms/step - loss: 1.7011 - a
Epoch 2/25
50000/50000 [=====] - 413s 8ms/step - loss: 1.2946 - a
Epoch 3/25
50000/50000 [=====] - 412s 8ms/step - loss: 1.1025 - a
Epoch 4/25
50000/50000 [=====] - 412s 8ms/step - loss: 0.9747 - a
Epoch 5/25
50000/50000 [=====] - 412s 8ms/step - loss: 0.8670 - a
Epoch 6/25
50000/50000 [=====] - 411s 8ms/step - loss: 0.7864 - a
Epoch 7/25
50000/50000 [=====] - 414s 8ms/step - loss: 0.7189 - a
Epoch 8/25
50000/50000 [=====] - 412s 8ms/step - loss: 0.6600 - a
Epoch 9/25
50000/50000 [=====] - 411s 8ms/step - loss: 0.6061 - a
Epoch 10/25
50000/50000 [=====] - 412s 8ms/step - loss: 0.5632 - a
Epoch 11/25
50000/50000 [=====] - 414s 8ms/step - loss: 0.5259 - a
Epoch 12/25
50000/50000 [=====] - 412s 8ms/step - loss: 0.4853 - a
Epoch 13/25
50000/50000 [=====] - 413s 8ms/step - loss: 0.4565 - a
Epoch 14/25
50000/50000 [=====] - 413s 8ms/step - loss: 0.4276 - a
Epoch 15/25
50000/50000 [=====] - 411s 8ms/step - loss: 0.3999 - a
Epoch 16/25
50000/50000 [=====] - 411s 8ms/step - loss: 0.3810 - a
Epoch 17/25
50000/50000 [=====] - 411s 8ms/step - loss: 0.3557 - a
Epoch 18/25
50000/50000 [=====] - 412s 8ms/step - loss: 0.3352 - a
Epoch 19/25
50000/50000 [=====] - 413s 8ms/step - loss: 0.3209 - a
Epoch 20/25
50000/50000 [=====] - 415s 8ms/step - loss: 0.3025 - a
Epoch 21/25
50000/50000 [=====] - 416s 8ms/step - loss: 0.2839 - a
Epoch 22/25
50000/50000 [=====] - 414s 8ms/step - loss: 0.2681 - a
Epoch 23/25
50000/50000 [=====] - 415s 8ms/step - loss: 0.2570 - a
Epoch 24/25
50000/50000 [=====] - 414s 8ms/step - loss: 0.2507 - a
Epoch 25/25
50000/50000 [=====] - 413s 8ms/step - loss: 0.2383 - a
Accuracy: 76.63%
```


Ответы на контрольные вопросы:

1) Что такое свертка?

Двумерная свертка — это довольно простая операция: начинаем с ядра, представляющего

Ядро “скользит” над двумерным изображением, поэлементно выполняя операцию умножения 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, таким образом, двумерную матрицу признаков. Признаки на выходе являются взвешенными суммами (где

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. Регуляризация помогает повысить точность и уменьшить вероятность переобучения. Тем самым на тестовых данных и тестовыми будет уменьшена.

6) **Как вид регуляризации использовался в лабораторной?**

В лабораторной используется dropout(исключение). Он характеризуется исключением определенных нейронов во время обучения нейронной сети. Это очень эффективный способ усреднения моделей, так как обученные нейроны получают в сети больший вес. Такой приём значительно увеличивает качество модели на тренировочных данных, а также повышает качество предсказаний модели на новых тестовых данных.

