МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИМ. Р.Е. АЛЕКСЕЕВА

ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Курс “Аппаратное и программное обеспечение роботизированных систем”

Отчет по лабораторной работе №3

Выполнил: Сорокин Е.А.

Проверил: Гай В.Е.

Нижний Новгород 2021

**Тема работы:**

Классификация изображений с использованием свёрточных нейронных сетей.

**Задание:**

Выполнить анализ статьи, разобрать структуру сети, реализовать сеть в Keras, оценить точность работы сети.

Вариант данных: CIFAR10

Вариант модели сети: VGG16

**Листинг программы:**

# Подключение модулей

from keras.models import Sequential

from keras.layers.core import Activation, Flatten, Dense, Dropout

from keras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D

from tensorflow.python.keras.preprocessing.image import ImageDataGenerator

from keras.applications.vgg16 import VGG16, preprocess\_input

from keras.datasets import cifar10

from keras.utils import np\_utils

from keras.preprocessing.image import ImageDataGenerator

from keras.optimizers import SGD, Adam

import cv2, numpy as np

import keras

#загрузили данные

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Размер изображений

img\_width, img\_height = 32, 32

# Размер мини-выборки

batch\_size = 100

# Кол-во изображений для обучения

nb\_train\_samples = 50000

# Кол-во изображений для теста

nb\_test\_samples = 10000

# normalize inputs from 0-255 to 0.0-1.0

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

# one hot encode outputs

# one hot encode outputs

y\_train = np\_utils.to\_categorical(y\_train)

y\_test = np\_utils.to\_categorical(y\_test)

num\_classes = y\_test.shape[1]

# Создание экземпляра модели сети VGG16

vgg16\_net = VGG16(weights='imagenet', include\_top=False, input\_shape=(img\_width, img\_height, 3))

# weights - веса предварительно обученной сети

# include\_top = false означает, что мы загружаем только сверточную часть сети, без квалификационной

# input\_shape - размер тензора

# Сверточную часть сети обучать не надо

vgg16\_net.trainable = True

trainable = False

for layer in vgg16\_net.layers:

  if layer.name == 'block1\_conv1':

    trainable = True

  layer.trainable = trainable

vgg16\_net.summary()

Model: "vgg16"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_18 (InputLayer) [(None, 32, 32, 3)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv1 (Conv2D) (None, 32, 32, 64) 1792

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_conv2 (Conv2D) (None, 32, 32, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block1\_pool (MaxPooling2D) (None, 16, 16, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv1 (Conv2D) (None, 16, 16, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_conv2 (Conv2D) (None, 16, 16, 128) 147584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block2\_pool (MaxPooling2D) (None, 8, 8, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv1 (Conv2D) (None, 8, 8, 256) 295168

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv2 (Conv2D) (None, 8, 8, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_conv3 (Conv2D) (None, 8, 8, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block3\_pool (MaxPooling2D) (None, 4, 4, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv1 (Conv2D) (None, 4, 4, 512) 1180160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv2 (Conv2D) (None, 4, 4, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_conv3 (Conv2D) (None, 4, 4, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block4\_pool (MaxPooling2D) (None, 2, 2, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv1 (Conv2D) (None, 2, 2, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv2 (Conv2D) (None, 2, 2, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_conv3 (Conv2D) (None, 2, 2, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

block5\_pool (MaxPooling2D) (None, 1, 1, 512) 0

=================================================================

Total params: 14,714,688

Trainable params: 14,714,688

Non-trainable params: 0

# Создание модели составной сети

model = Sequential()

# Добавляем сверточные слои

model.add(vgg16\_net)

# Преобразуем двумерный массив vgg16 в одномерный

model.add(Flatten())

# Полносвязный слой

model.add(Dense(256, activation='relu'))

# Слой регуляризации (для предотвращения переобучения)

model.add(Dropout(0.5))

# Кол-во классов

model.add(Dense(num\_classes, activation='softmax'))

model.summary()

Model: "sequential\_17"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

vgg16 (Functional) (None, 1, 1, 512) 14714688

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_17 (Flatten) (None, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_34 (Dense) (None, 256) 131328

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_17 (Dropout) (None, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_35 (Dense) (None, 10) 2570

=================================================================

Total params: 14,848,586

Trainable params: 14,848,586

Non-trainable params: 0

# Компилируем составную сеть   'SGD'

# opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)# Инициализировать оптимизатор RMSprop

epochs = 20

lrate = 0.001

decay = lrate/epochs

sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False) #Adam(lr=1e-5)

model.compile(loss='categorical\_crossentropy',

              optimizer=sgd,

              metrics=['accuracy'])

model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs = epochs, batch\_size=batch\_size)

Epoch 1/20

500/500 [==============================] - 17s 32ms/step - loss: 1.5561 - accuracy: 0.4458 - val\_loss: 0.7625 - val\_accuracy: 0.7380

Epoch 2/20

500/500 [==============================] - 16s 31ms/step - loss: 0.7379 - accuracy: 0.7504 - val\_loss: 0.6465 - val\_accuracy: 0.7743

Epoch 3/20

500/500 [==============================] - 16s 31ms/step - loss: 0.5811 - accuracy: 0.8060 - val\_loss: 0.6570 - val\_accuracy: 0.7715

Epoch 4/20

500/500 [==============================] - 15s 30ms/step - loss: 0.4934 - accuracy: 0.8351 - val\_loss: 0.5178 - val\_accuracy: 0.8221

Epoch 5/20

500/500 [==============================] - 16s 31ms/step - loss: 0.4119 - accuracy: 0.8620 - val\_loss: 0.4948 - val\_accuracy: 0.8340

Epoch 6/20

500/500 [==============================] - 16s 31ms/step - loss: 0.3511 - accuracy: 0.8809 - val\_loss: 0.4897 - val\_accuracy: 0.8383

Epoch 7/20

500/500 [==============================] - 15s 30ms/step - loss: 0.2960 - accuracy: 0.8999 - val\_loss: 0.4969 - val\_accuracy: 0.8354

Epoch 8/20

500/500 [==============================] - 15s 31ms/step - loss: 0.2606 - accuracy: 0.9117 - val\_loss: 0.5261 - val\_accuracy: 0.8282

Epoch 9/20

500/500 [==============================] - 16s 31ms/step - loss: 0.2044 - accuracy: 0.9311 - val\_loss: 0.4601 - val\_accuracy: 0.8531

Epoch 10/20

500/500 [==============================] - 16s 31ms/step - loss: 0.1696 - accuracy: 0.9439 - val\_loss: 0.4897 - val\_accuracy: 0.8525

Epoch 11/20

500/500 [==============================] - 15s 31ms/step - loss: 0.1374 - accuracy: 0.9558 - val\_loss: 0.5055 - val\_accuracy: 0.8535

Epoch 12/20

500/500 [==============================] - 15s 31ms/step - loss: 0.1187 - accuracy: 0.9612 - val\_loss: 0.5299 - val\_accuracy: 0.8517

Epoch 13/20

500/500 [==============================] - 15s 31ms/step - loss: 0.0913 - accuracy: 0.9702 - val\_loss: 0.5611 - val\_accuracy: 0.8565

Epoch 14/20

500/500 [==============================] - 15s 31ms/step - loss: 0.0718 - accuracy: 0.9777 - val\_loss: 0.5739 - val\_accuracy: 0.8563

Epoch 15/20

500/500 [==============================] - 16s 31ms/step - loss: 0.0518 - accuracy: 0.9852 - val\_loss: 0.6376 - val\_accuracy: 0.8481

Epoch 16/20

500/500 [==============================] - 16s 31ms/step - loss: 0.0500 - accuracy: 0.9852 - val\_loss: 0.6121 - val\_accuracy: 0.8571

Epoch 17/20

500/500 [==============================] - 16s 31ms/step - loss: 0.0399 - accuracy: 0.9884 - val\_loss: 0.6536 - val\_accuracy: 0.8581

Epoch 18/20

500/500 [==============================] - 15s 31ms/step - loss: 0.0319 - accuracy: 0.9907 - val\_loss: 0.6815 - val\_accuracy: 0.8535

Epoch 19/20

500/500 [==============================] - 16s 31ms/step - loss: 0.0244 - accuracy: 0.9925 - val\_loss: 0.7077 - val\_accuracy: 0.8564

Epoch 20/20

500/500 [==============================] - 16s 31ms/step - loss: 0.0255 - accuracy: 0.9928 - val\_loss: 0.7032 - val\_accuracy: 0.8610

<keras.callbacks.History at 0x7fe609a0fe90>

# Final evaluation of the model

scores = model.evaluate(x\_test, y\_test, verbose=0)

print("Accuracy: %.2f%%" % (scores[1]\*100))

Accuracy: 86.10%