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 keras.optimizers import SGD

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

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 = 32

# 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

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

# Сверточная часть сети

vgg16\_net.trainable = True

trainable = False

for layer in vgg16\_net.layers:

  if layer.name == 'block5\_conv1':

    trainable = True

  layer.trainable = trainable

vgg16\_net.summary()

Model: "vgg16"

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

Layer (type) Output Shape Param #

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

input\_5 (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: 7,079,424

Non-trainable params: 7,635,264

# 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='sigmoid'))

model.summary()

Model: "sequential\_2"

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

Layer (type) Output Shape Param #

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

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

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

flatten\_2 (Flatten) (None, 512) 0

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

dense\_4 (Dense) (None, 256) 131328

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

dropout\_2 (Dropout) (None, 256) 0

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

dense\_5 (Dense) (None, 10) 2570

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

Total params: 14,848,586

Trainable params: 7,213,322

Non-trainable params: 7,635,264

# Компилируем составную сеть

epochs = 5

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'])

# Обучение составной сети

model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs = 5, batch\_size=batch\_size)

Epoch 1/5

1563/1563 [==============================] - 1254s 801ms/step - loss: 1.4192 - accuracy: 0.5033 - val\_loss: 0.8768 - val\_accuracy: 0.6947

Epoch 2/5

1563/1563 [==============================] - 1314s 841ms/step - loss: 0.8624 - accuracy: 0.7055 - val\_loss: 0.7972 - val\_accuracy: 0.7256

Epoch 3/5

1563/1563 [==============================] - 1348s 863ms/step - loss: 0.7308 - accuracy: 0.7505 - val\_loss: 0.7585 - val\_accuracy: 0.7423

Epoch 4/5

1563/1563 [==============================] - 1359s 869ms/step - loss: 0.6421 - accuracy: 0.7782 - val\_loss: 0.7378 - val\_accuracy: 0.7457

Epoch 5/5

1563/1563 [==============================] - 1363s 872ms/step - loss: 0.5889 - accuracy: 0.7958 - val\_loss: 0.7437 - val\_accuracy: 0.7476

<keras.callbacks.History at 0x7fdbc05c4e10>

# Final evaluation of the model

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

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

Accuracy: 74.76%