МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание объектов на фотографиях»

Студент гр. 7383 _____ Лосев М.Л.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель.

Распознавание объектов на фотографиях (Object Recognition in Photographs)

CIFAR-10 (классификация небольших изображений по десяти классам: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль и грузовик).

Задачи.

- Ознакомиться со сверточными нейронными сетями
- Изучить построение модели в Keras в функциональном виде
- Изучить работу слоя разреживания (Dropout)

Выполнение работы.

1. Обучим сверточную нейронную сеть, предложенную в методических указаниях к работе. Поскольку на одну эпоху обучения уходит несколько минут, а аппаратное ускорение с помощью GPU недоступно, уменьшим количество образцов с 50 тыс. до 10 (для этого удобно взять тестовые данные, которых как раз 10 тыс., потому что разделение на обучающие и проверочные данные условно), а количество эпох уменьшим до 25. Это снизит точность предсказаний обученной модели, но, с другой стороны, модели, обученные при таких гиперпараметрах все равно можно будет сравнивать друг с другом. Таким образом, можно будет обучить модель на доступной вычислительной мощности за адекватное время.

   Код представлен в приложении А.

   График точности для фильтра размера 3x3 с Dorpout представлен на рисунке 1.

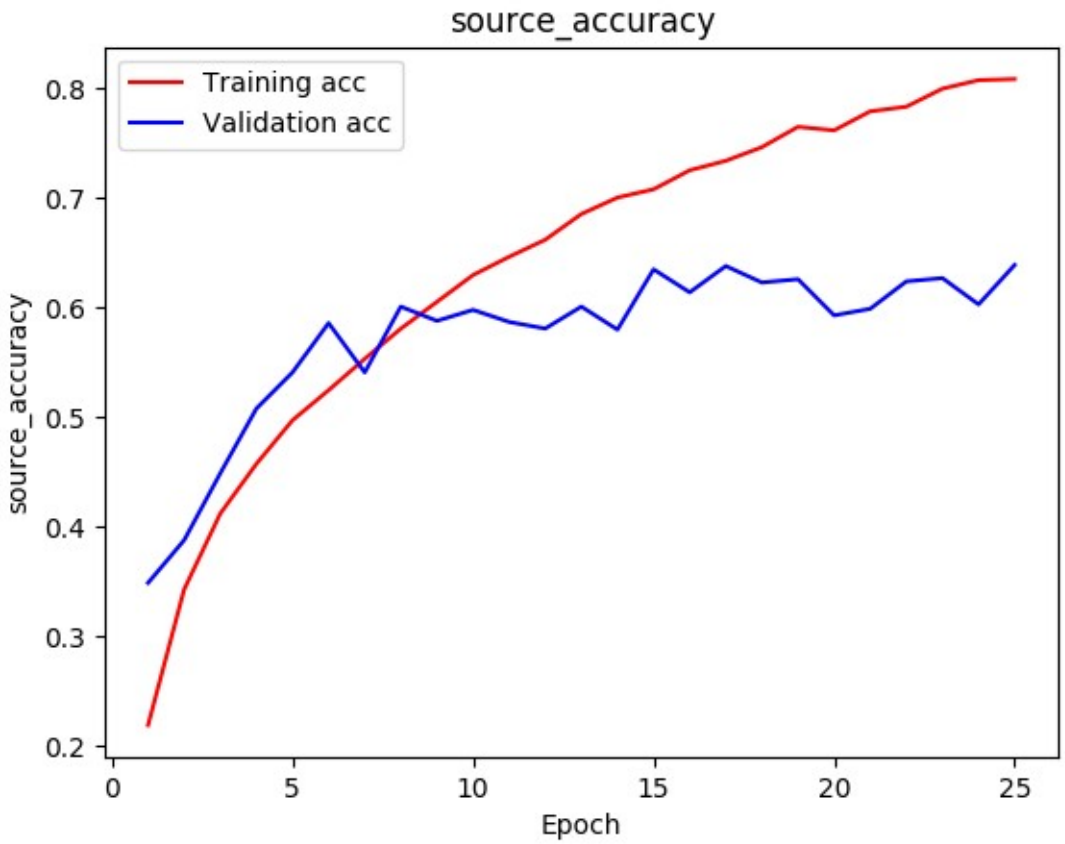

Illustration 1: Обучение модели с Dorpout и ядром 3x3

2. Чтобы исследовать работу сети без слоя Dorpout, уберем оба таких слоя. График обучения этой модели представлен на рисунке 2. Сравним с графиком обучения модели с Dorpout: видно, что точность на обучающих данных выросла, а на проверочных упала после удаления слоев Dorpout. Это говорит о том, что слои Dorpout в условиях решаемой задачи действительно эффективны в борьбе с переобучением.

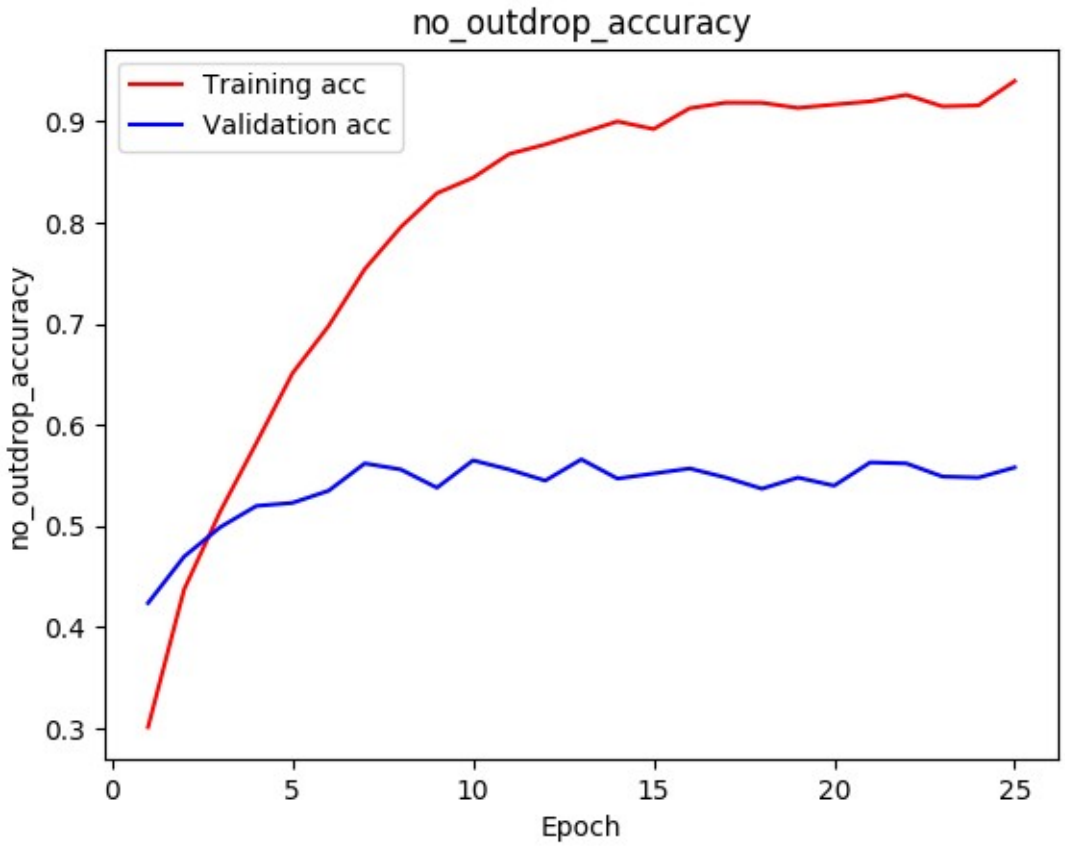

Illustration 2: Обучение модели без Dorpout

3. Так как картинка имеет небольшой размер (32x32), целесообразным представляется использовать малые размеры ядра: например, рассмотрим фильтры 3*3, 4*4 и 5*5. Модель с размером фильтра 3x3 уже была обучена, график представлен на рис.1. Графики обучения моделей с размерами фильтров 4x4 и 5x5 показаны на рисунках 3 и 4 соответственно. Они показывают, что ядро 3x3 подошло лучше.

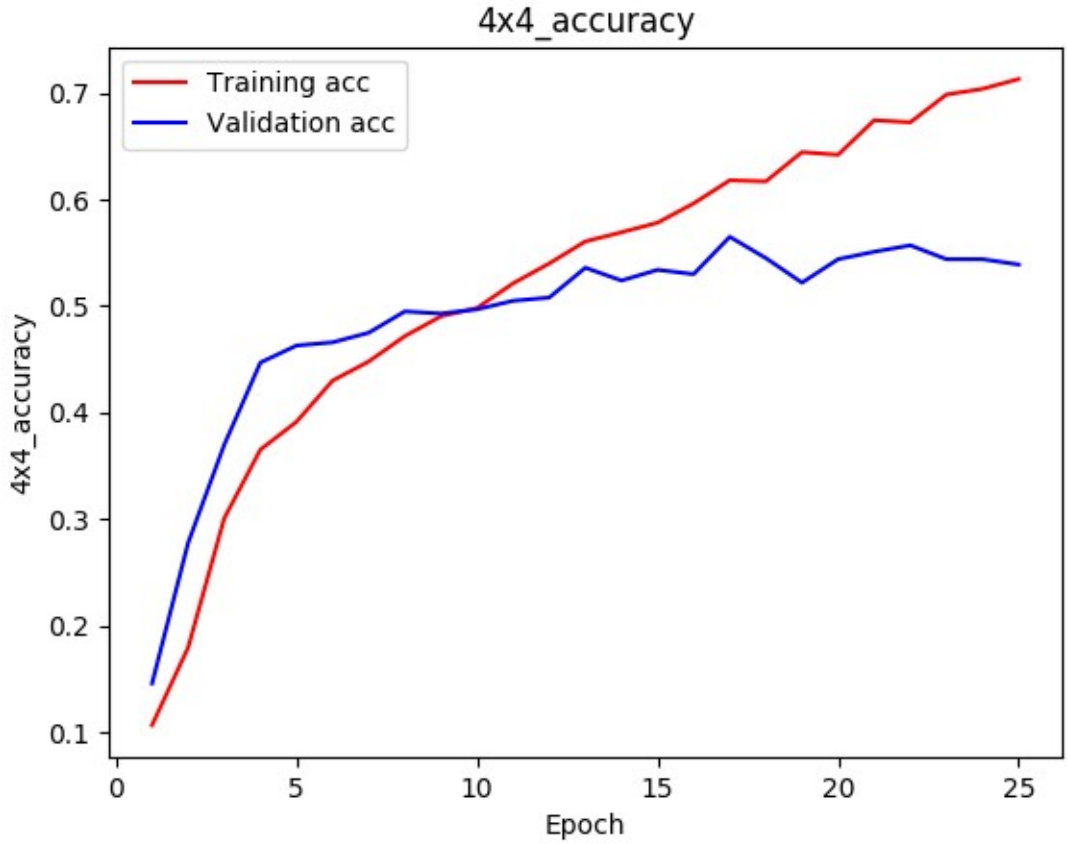

Illustration 3: Обучение модели с ядром 4x4

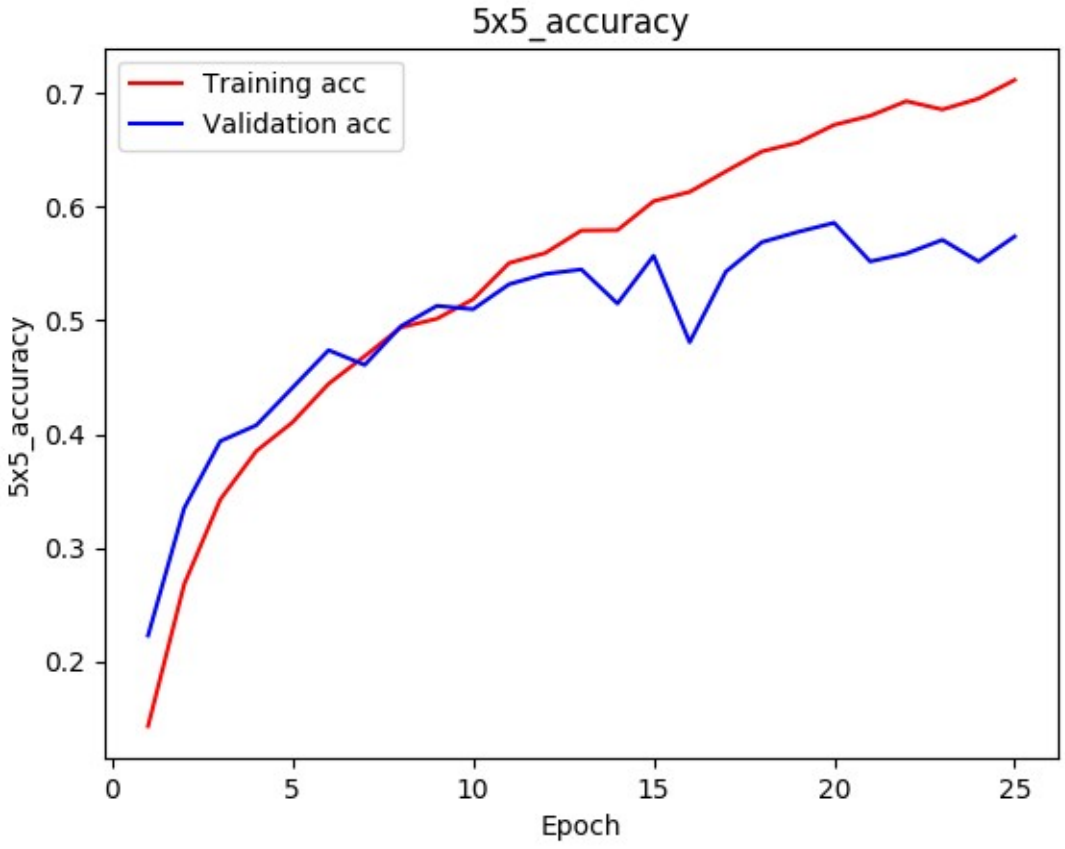

Illustration 4: Обучение модели с ядром 5x5

Вывод.

В ходе выполнения данной работы была создана сверточная нейронная сеть, классифицирующая изображения десяти категорий. Было исследовано влияение слоя Dropout на обучение: установленно, что он снижает переобучение. Было исследовано вияние размера ядра свертки на обучение: для изображения размера 32x32 лучше всего использовать фильтр 3x3, хотя для изображений больших размеров обычно применяются фильтры большего размера: от 3x3 до 7x7, хотя и это не предел.

Приложение А

```
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Input, Convolution2D, MaxPooling2D,
Dense, Dropout, Flatten
from keras.utils import np_utils
import numpy as np
import matplotlib.pyplot as plt
import os

def save_plot(label, history):
    picdir = './pics/'
    plt.clf()
    arg = range(1, len(history['accuracy']) + 1)
    plt.plot(arg, history['accuracy'], 'r', label='Training acc')
    plt.plot(arg, history['val_accuracy'], 'b', label='Validation
acc')
    plt.title(label)
    plt.xlabel('Epoch')
    plt.ylabel(label)
    plt.legend()
    plt.savefig(label + '.png')
    plt.savefig(picdir + label + '.png')

batch_size = 32 # in each iteration, we consider 32 training
examples at once
num_epochs = 25 # we iterate 25 times over the entire training set
kernel_size = 2 # we will use 3x3 kernels throughout
pool_size = 2 # we will use 2x2 pooling throughout
conv_depth_1 = 32 # we will initially have 32 kernels per conv.
layer...
conv_depth_2 = 64 # ...switching to 64 after the first pooling
layer
drop_prob_1 = 0.25 # dropout after pooling with probability 0.25
drop_prob_2 = 0.5 # dropout in the dense layer with probability
0.5
hidden_size = 512 # the dense layer will have 512 neurons

(X_train, y_train), (X_test, y_test) = cifar10.load_data() # fetch
CIFAR-10 data
num_train, depth, height, width = X_train.shape # there are 50000
training examples in CIFAR-10
print(X_train.shape)
num_test = X_test.shape[0] # there are 10000 test examples in
CIFAR-10
num_classes = np.unique(y_train).shape[0] # there are 10 image
classes
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= np.max(X_train) # Normalise data to [0, 1] range
X_test /= np.max(X_train) # Normalise data to [0, 1] range
Y_train = np_utils.to_categorical(y_train, num_classes) # One-hot
encode the labels
```

```
Y_test = np_utils.to_categorical(y_test, num_classes) # One-hot
encode the labels

inp = Input(shape=(depth, height, width)) # N.B. depth goes first
in Keras
# Conv [32] -> Conv [32] -> Pool (with dropout on the pooling
layer)
conv_1 = Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='same', activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='same', activation='relu')(conv_1)
pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_2)
drop_1 = Dropout(drop_prob_1)(pool_1)
# Conv [64] -> Conv [64] -> Pool (with dropout on the pooling
layer)
conv_3 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, kernel_size, kernel_size,
border_mode='same', activation='relu')(conv_3)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_4)
drop_2 = Dropout(drop_prob_1)(pool_2)
# Now flatten to 1D, apply Dense -> ReLU (with dropout) -> softmax
flat = Flatten()(drop_2)
hidden = Dense(hidden_size, activation='relu')(flat)
drop_3 = Dropout(drop_prob_2)(hidden)
out = Dense(num_classes, activation='softmax')(drop_3)
model = Model(input=inp, output=out) # To define a model, just
specify its input and output layers

model.compile(loss='categorical_crossentropy', # using the cross-
entropy loss function
              optimizer='adam', # using the Adam optimiser
              metrics=['accuracy']) # reporting the accuracy
H = model.fit(X_test, Y_test, # Train the model using the test
set...
          batch_size=batch_size, nb_epoch=num_epochs,
          verbose=1, validation_split=0.1) # ...holding out 10% of
the data for validation
save_plot('2x2_accuracy', H.history)
model.save('2x2.h5')
# the sound of dantage to wake me up as the model got trained
os.system('play --no-show-progress --null --channels 1 synth 1
sine 440')
```