**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Искусственные нейронные сети»**

**Тема: «Распознавание объектов на фотографиях»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7381 |  | Лукашев Р.С. |
| Преподаватель |  | Жукова Н.А. |

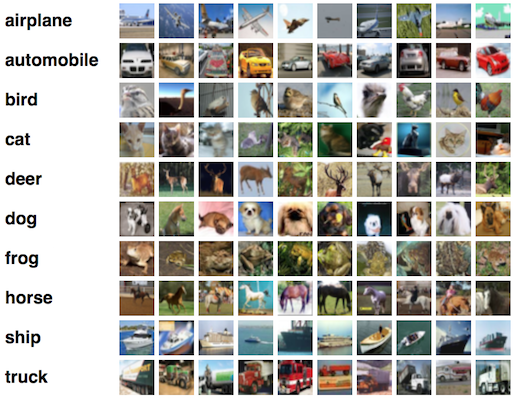
Санкт-Петербург

2020

**Цель работы.**

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);

**Требования.**

1. Построить и обучить сверточную нейронную сеть;
2. Исследовать работу сеть без слоя Dropout;
3. Исследовать работу сети при разных размерах ядра свертки.

**Ход работы.**

Была написана функция, создающая модель с заданным размером ядра свертки, и опционально с dropout слоями. Вместо 200 эпох было выбрано 15, поскольку такого количества эпох достаточно, чтобы делать какие-то выводы и сравнивать модели. Посмотрим на разницу между моделью, в которой есть слои разреживания, и в которой таковые отсутствуют (см. рис 1, 2).

Рис. 1 – Точность и потери при обучении модели без слоев разреживания.

Рис. 2 – Точность и потери при обучении модели со слоями разреживания.

Как можно заметить, у модели без слоев разреживания быстро происходит переобучение.

Далее посмотрим, что случится, если мы изменим размер ядра свертки (см. рис 3).

Рис. 3 – Точность и потери при обучении моделей с разным размером ядра.

Размерность ядра контролирует глубину тензора признаков. Результатом действия каждого сверточного уровня является карта признаков. Маленькие размерности (3x3) позволяют улавливать более мелкие детали, что в данном случае позволяет точнее классифицировать картинку. Ядро размерностью 1x1 улавливает слишком мелкие признаки, которые вполне могут принадлежать всем или нескольким классам одновременно, из-за чего уменьшилась точность классификации. Большие размерности не способны улавливать мелкие детали, что в данном случае привело к снижению точности классификации.

**Выводы.**

В ходе выполнения данной лабораторной работы было произведено ознакомление со сверточными нейронными сетями: изучено построение модели в Keras

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД**

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 matplotlib.pyplot as plt

import numpy as np

batch\_size = 32 # in each iteration, we consider 32 training examples at once

num\_epochs = 15 # we iterate 15 times over the entire training set

kernel\_size = [1, 3, 10] # 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

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

def fit\_model(kernel\_size=3, dropout=True):

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)

if dropout:

drop\_1 = Dropout(drop\_prob\_1)(pool\_1)

else:

drop\_1 = conv\_2

# 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)

if dropout:

drop\_2 = Dropout(drop\_prob\_1)(pool\_2)

else:

drop\_2 = conv\_4

# Now flatten to 1D, apply Dense -> ReLU (with dropout) -> softmax

flat = Flatten()(drop\_2)

hidden = Dense(hidden\_size, activation='relu')(flat)

if dropout:

drop\_3 = Dropout(drop\_prob\_2)(hidden)

else:

drop\_3 = 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

history = model.fit(X\_train, Y\_train, # Train the model using the training set...

batch\_size=batch\_size, nb\_epoch=num\_epochs,

verbose=1, validation\_split=0.1) # ...holding out 10% of the data for validation

return history

def graph(H, title):

loss = H.history['loss']

val\_loss = H.history['val\_loss']

acc = H.history['accuracy']

val\_acc = H.history['val\_accuracy']

epochs = range(1, len(loss) + 1)

plt.title(title)

# Построение графика ошибки

plt.plot(epochs, loss, 'r', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title(title + ' Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Построение графика точности

plt.clf()

plt.plot(epochs, acc, 'r', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title(title + ' Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

graph(fit\_model(kernel\_size=3, dropout=False), "Without dropout layers")

for k\_s in kernel\_size:

graph(fit\_model(kernel\_size=k\_s, dropout=True), "Kernel size = " + str(k\_s))