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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Искусственные нейронные сети»
Тема: Распознавание объектов на фотографиях

Студент гр. 7382 ____________________ Гаврилов А.В.

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).

Порядок выполнения работы.

1. Ознакомиться со сверточными нейронными сетями.
2. Изучить построение модели в Keras в функциональном виде.
3. Изучить работу слоя разреживания (Dropout).

Требования к выполнению задания.

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

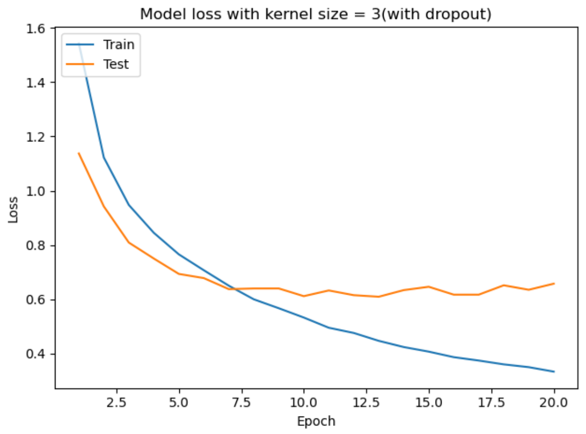
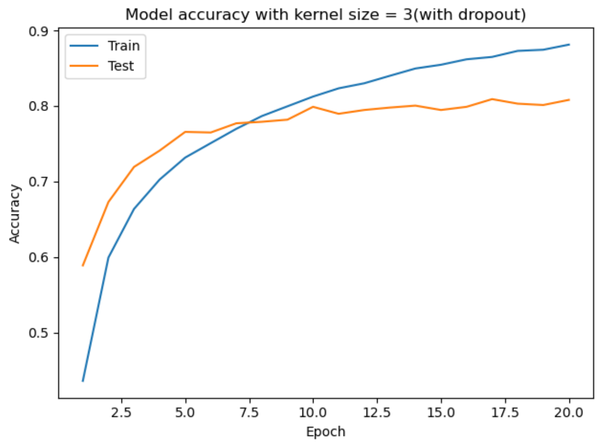
Ход работы.

Была создана и обучена модель искусственной нейронной сети. Код предоставлен в приложении А.

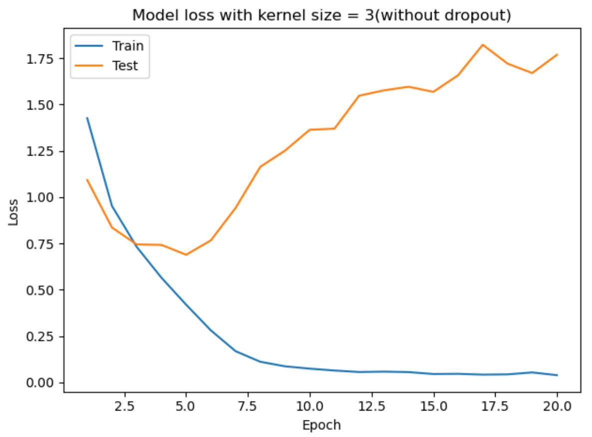
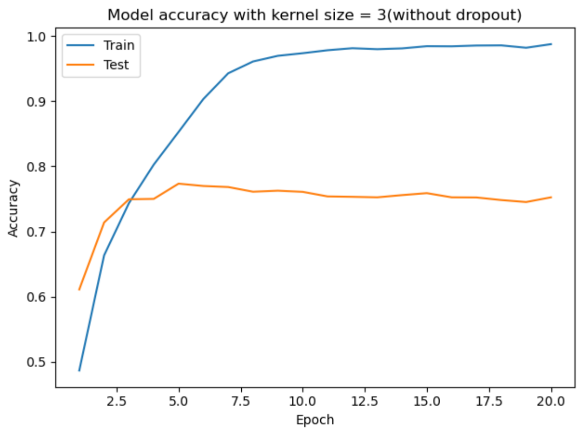
Архитектура созданной нейронной сети основана на модели из исходных данных к лабораторной работы и выставлены следующие параметры:

- Оптимизатор ‘adam’
- Количество эпох обучения – 20
- batch_size = 80
- pool_size = 2*2
- Ядро свертки: 3*3

После обучения этой модели получили следующие результаты:

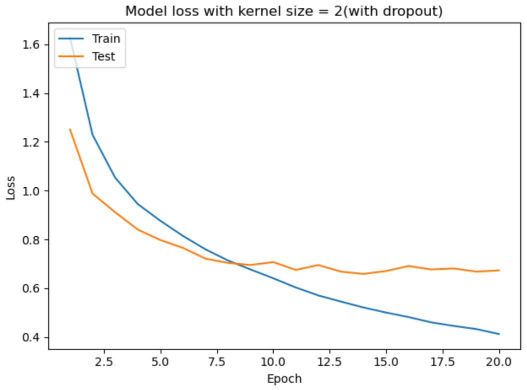
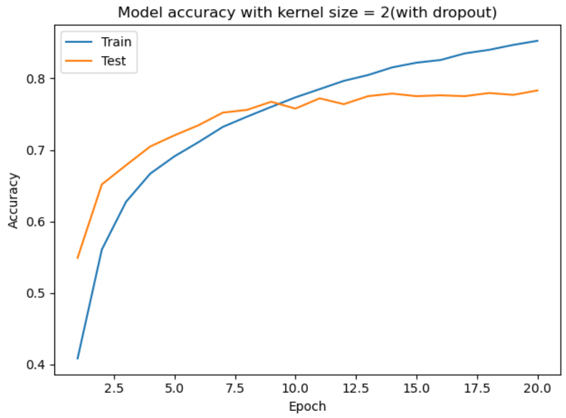
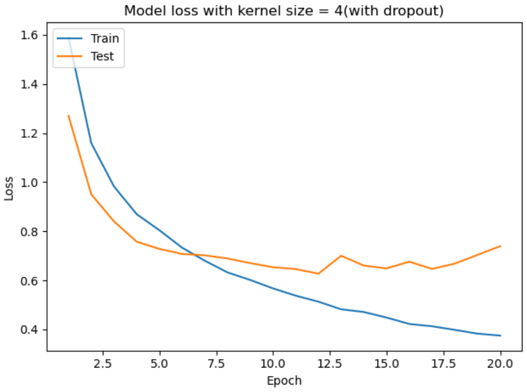
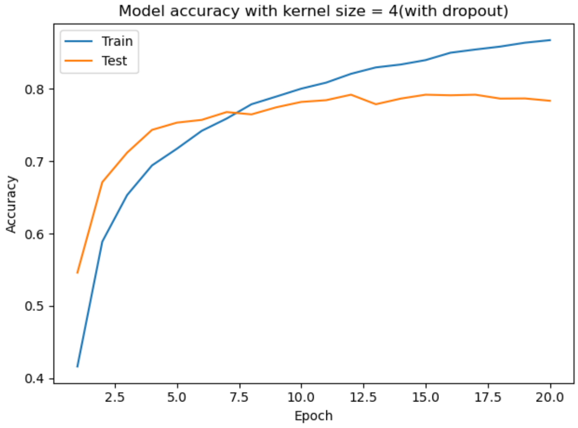
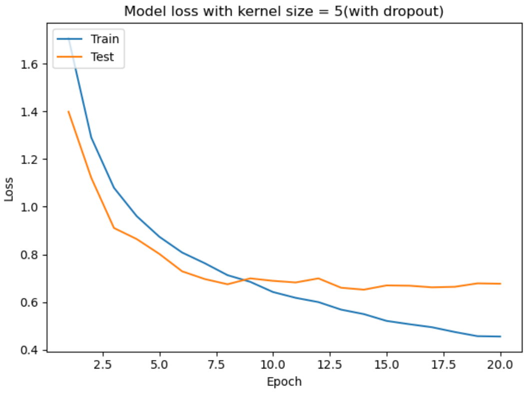
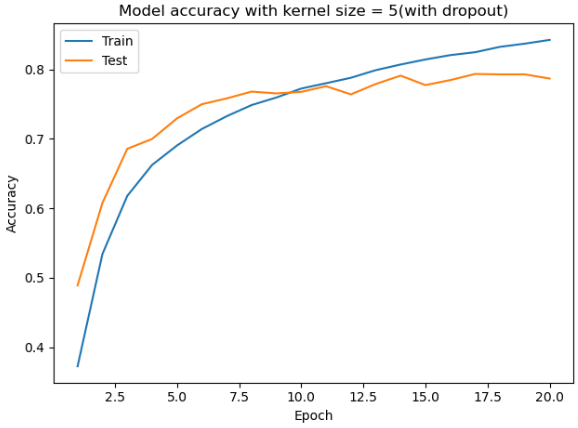
График потерь	График точности
	

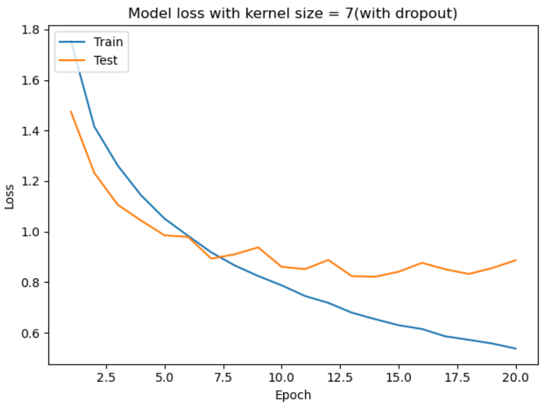
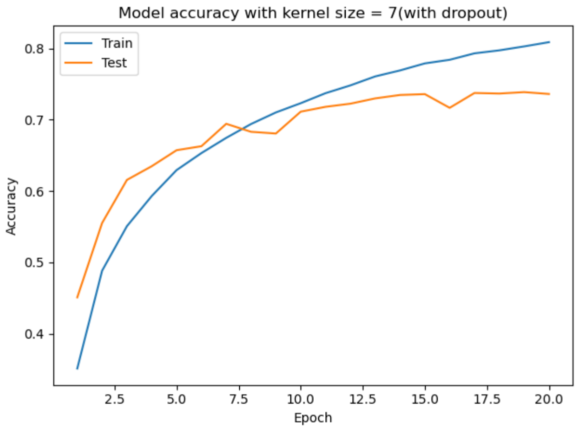
Проверим результаты, если убрать dropout слой:

График потерь	График точности
	

Как и ожидалось, модель стала переобучаться примерно на 6 эпохе. Таким образом справляются со своей задачей достаточно неплохо не давая сети переобучаться и сохраняя возможность сети к обобщению.

Исследуем модель при различных размерностях ядра свертки (с dropout слоями):

<table>
<tr><th>N</th><th>График потерь</th><th>График точности</th></tr>
<tr><td>2</td><td>
</td><td>
</td></tr>
<tr><td>4</td><td>
</td><td>
</td></tr>
<tr><td>5</td><td>
</td><td>
</td></tr>
</table>

7		

При изменении размера ядра свертки точность и потери сильно не изменяются и на процесс обучения это почти не повлияло. Но при размере ядра 7*7 ошибка была наибольшей ~0.7 в сравнении с ~0.4-0.6 на остальных вариантах. Лучшей моделью оказался изначальный вариант с ядром 3*3, точность которого ~0.85.

Выводы.

В ходе работы была изучена задача классификация изображений из датасета CIFAR-10. Подобрана архитектура, дающая точность 85%. При исследовании влияния dropout слоев на модель, было доказано, что эти слои помогают при переобучении, неточности подобранных параметров и оставляют модели возможность к обобщению.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.py:

```
import gen_data

from tensorflow.keras.layers import Input, Convolution2D,
MaxPooling2D, Dense, Dropout, Flatten
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt



class Net:
    def __init__(self):
        self.__set_params()
        self.x_train, self.y_train, self.num_classes,
self.input_shape, self.num_train = gen_data.get_data()


    def __set_params(self):
        self.num_epochs = 20
        self.batch_size = 80
        self.pool_size = (2, 2)
        self.conv_depth_1 = 32
        self.conv_depth_2 = 64
        self.drop_prob_1 = 0.25
        self.drop_prob_2 = 0.5
        self.hidden_size = 512

    def build_model(self, kernel_size=3, hasDropout=True):
        inp = Input(shape=self.input_shape)
        conv_1 = Convolution2D(self.conv_depth_1, (kernel_size,
kernel_size),
                               padding='same', strides=(1, 1),
activation='relu')(inp)
        conv_2 = Convolution2D(self.conv_depth_1, (kernel_size,
kernel_size),
                               padding='same',
activation='relu')(conv_1)
        pool_1 = MaxPooling2D(pool_size=self.pool_size)(conv_2)
        if hasDropout:
            drop_1 = Dropout(self.drop_prob_1)(pool_1)
        else:
            drop_1 = pool_1
        conv_3 = Convolution2D(self.conv_depth_2, (kernel_size,
kernel_size),
                               padding='same', strides=(1, 1),
activation='relu')(drop_1)
        conv_4 = Convolution2D(self.conv_depth_2, (kernel_size,
```

```
kernel_size),
                              padding='same', strides=(1, 1),
activation='relu')(conv_3)
        pool_2 = MaxPooling2D(pool_size=self.pool_size)(conv_4)
        if hasDropout:
            drop_2 = Dropout(self.drop_prob_1)(pool_2)
        else:
            drop_2 = pool_2
        flat = Flatten()(drop_2)
        hidden = Dense(self.hidden_size, activation='relu')(flat)
        if hasDropout:
            drop_4 = Dropout(self.drop_prob_2)(hidden)
        else:
            drop_4 = hidden
        out = Dense(self.num_classes, activation='softmax')(drop_4)
        model = Model(inp, out)
        model.compile(loss='categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])
        return model

    def plot(self, history, hasDropout, kernel_size):
        if hasDropout:
            hasDropout = '(with dropout)'
        else:
            hasDropout = '(without dropout)'
        x = range(1, self.num_epochs + 1)
        plt.plot(x, history.history['loss'])
        plt.plot(x, history.history['val_loss'])
        plt.title('Model loss with kernel size = ' + str(kernel_size)
+ hasDropout)
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()
        plt.plot(x, history.history['acc'])
        plt.plot(x, history.history['val_acc'])
        plt.title('Model accuracy with kernel size = ' +
str(kernel_size) + hasDropout)
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()

    def lab5(self):
        model = self.build_model()
        history = model.fit(self.x_train, self.y_train,
                            batch_size=self.batch_size,
epochs=self.num_epochs,
                            verbose=1, validation_split=0.1)
        self.plot(history, True, 3)
```

```
        model = self.build_model(hasDropout=False)
        history = model.fit(self.x_train, self.y_train,
                            batch_size=self.batch_size,
epochs=self.num_epochs,
                            verbose=1, validation_split=0.1)
        self.plot(history, False, 3)
        k_size = [2, 4, 5, 7]
        for i in k_size:
            model = self.build_model(i)
            history = model.fit(self.x_train, self.y_train,
                            batch_size=self.batch_size,
epochs=self.num_epochs,
                            verbose=1, validation_split=0.1)
            self.plot(history, True, i)


lab = Net()
lab.lab5()
```

Файл gen_data.py:

```
from tensorflow.keras.datasets import cifar10
from keras.utils import to_categorical
import numpy as np

def get_data(size = None):
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()
    num_train, depth, height, width = x_train.shape
    num_test = x_test.shape[0]
    num_classes = np.unique(y_train).shape[0]
    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255.0
    x_test /= 255.0
    y_train = to_categorical(y_train, num_classes)
    y_test = to_categorical(y_test, num_classes)
    input_shape = (depth, height, width)
    return x_train, y_train, num_classes, input_shape, num_train
```