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»

Студентка гр. 7381 _____ Алясова А.Н.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Написать программу для распознавания объектов на фотографиях (Object Recognition in Photographs) CIFAR-10, научиться классифицировать небольшие изображения 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.

- Построить и обучить сверточную нейронную сеть
- Исследовать работу сеть без слоя `Dropout`
- Исследовать работу сети при разных размерах ядра свертки

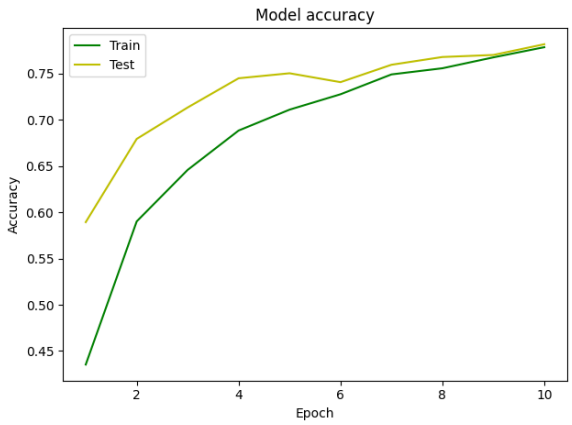
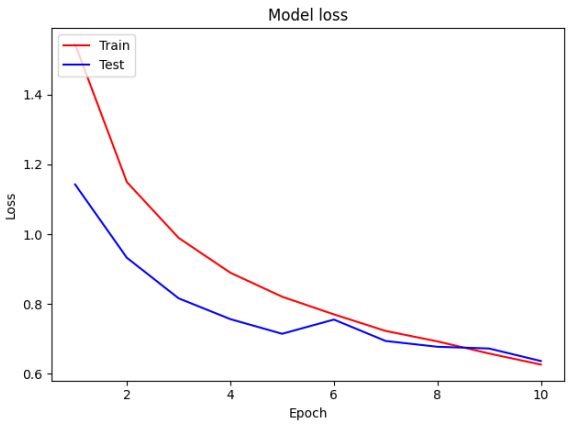
Ход работы.

В ходе работы была создана и обучена модель искусственной нейронной сети в соответствии с условиями (код представлен в приложении).

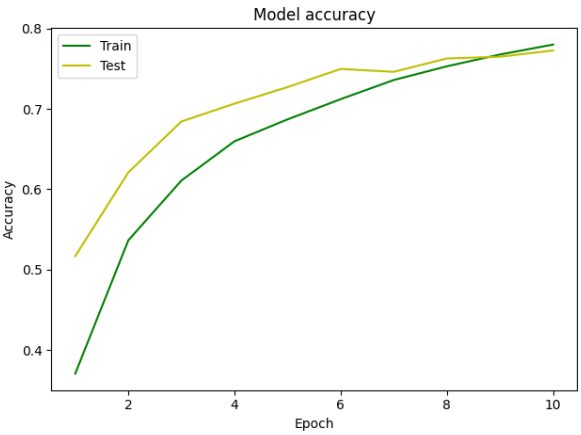
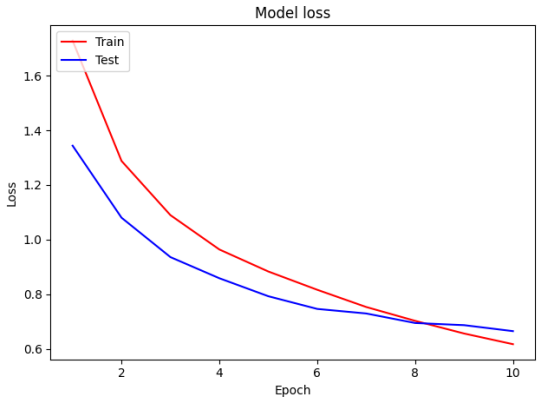
1. ***Выберем модель сети.***

Изначально была создана сверточная сеть использующая сверточные слои, слои `maxpooling` и слои разреживания `dropout`. Протестировали модель с разным количеством параметра `batch_size`.

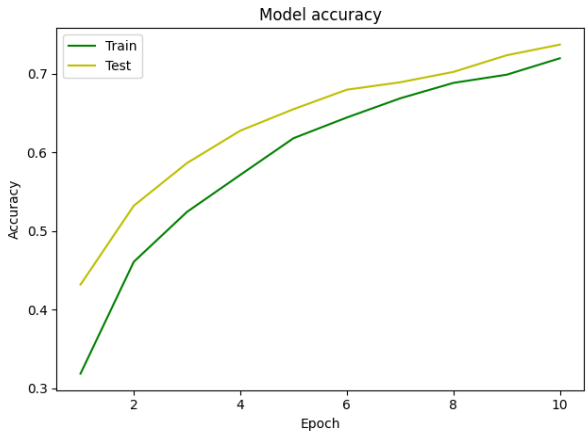
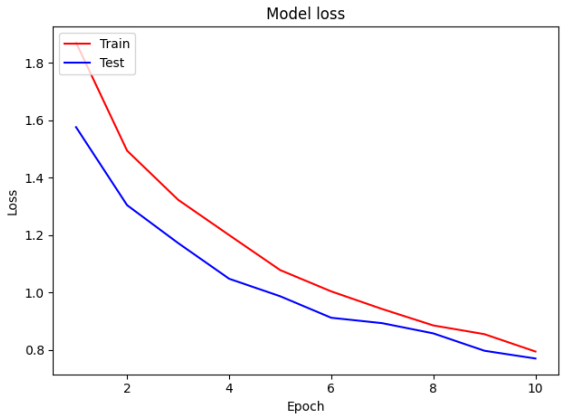
`batch_size` = 32

График точности	График потерь
	

`batch_size` = 256

График точности	График потерь
	

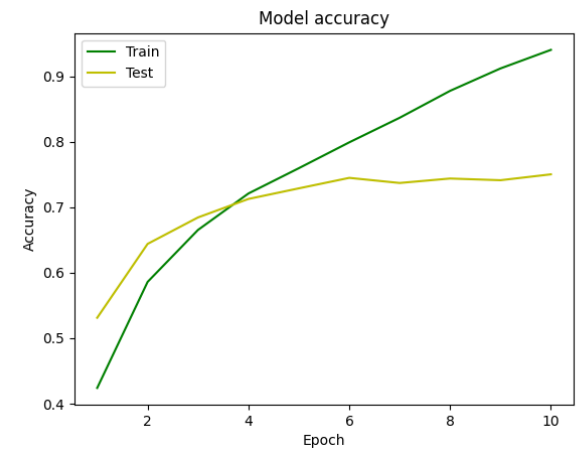
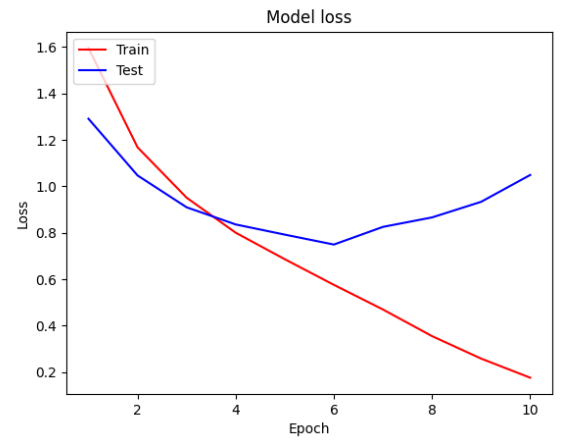
`batch_size` = 712

График точности	График потерь
	

Как видно из графиков выше точность у модели с `batch_size` = 256, поэтому будем в дальнейшем рассматривать ее.

2. Исследуем работу сети без слоя Dropout

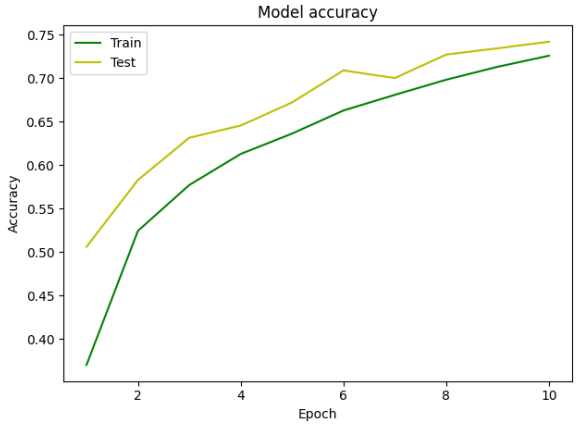
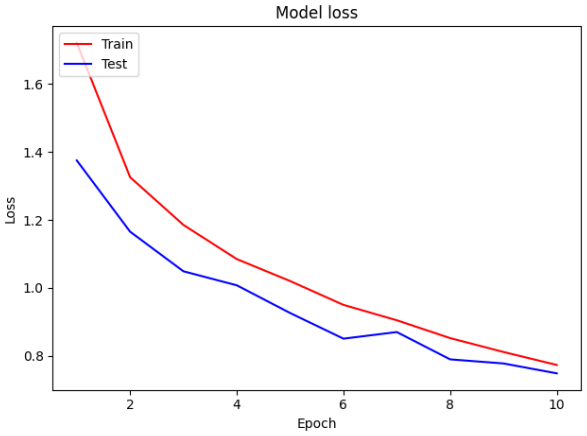
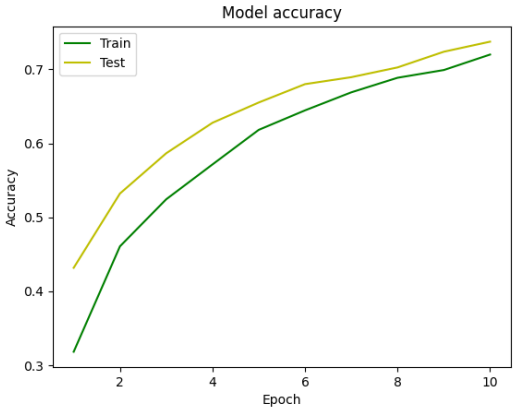
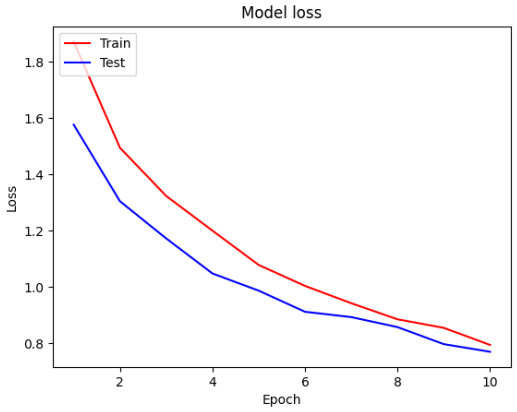
Исследовали выбранную модель без слоя `Dropout`.

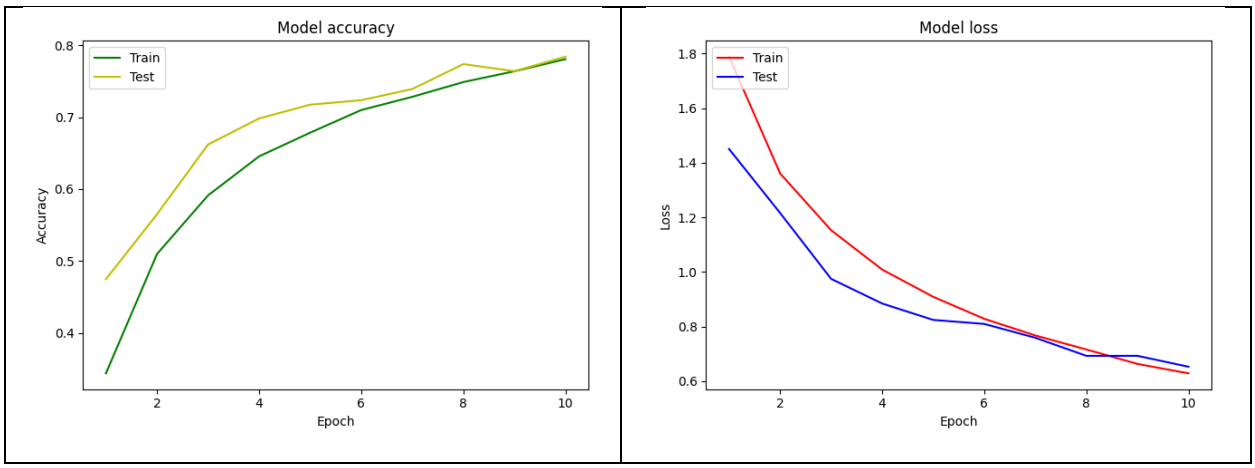
График точности	График потерь
	

Из графиков видно, что после 4 эпохи потери начали расти, а точность перестала повышаться, из чего можно сделать вывод о необходимости слоев прореживания.

3. Исследуем работу сети при разных размерах ядра свертки

Были изучены архитектуры, у которых в сверточных слоях размер ядра свертки имеет форму (2,2), (3,3) и (5,5) соответственно. Результаты представлены в следующей таблице.

<table>
<tr><td colspan="2">Размер ядра свертки 2*2</td></tr>
<tr><td>График точности</td><td>График потерь</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Размер ядра свертки 3*3</td></tr>
<tr><td>График точности</td><td>График потерь</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Размер ядра свертки 5*5</td></tr>
<tr><td>График точности</td><td>График потерь</td></tr>
</table>



По графикам видно, что при для данной модели точность наибольшая при размере ядра 5*5, потери тестовых данных тоже меньше, чем при размере ядра 3*3, поэтому для данной модели ядро размером 5*5 подходит больше.

Выводы.

В ходе выполнения данной работы была создана сеть для классификации изображений, были более подробно изучены сверточные сети, влияние слоев разреживания и ядра свертки на результаты обучения.

ПРИЛОЖЕНИЕ

Исходный код

```
import numpy as np
import matplotlib.pyplot as plt
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


batch_size = 256 # в каждой итерации одновременно рассматриваем 32
обучающих примера
num_epochs = 10 # повторяем 200 раз по всему тренировочному набору
#kernel_size = 3 # будем использовать ядра 3x3

pool_size = 2 # мы будем использовать пул (объединение во всем) 2x2
conv_depth_1 = 32 # у нас изначально будет 32 ядра на conv. layer...
conv_depth_2 = 64 # ...меняем на 64 после первого уровня пула
drop_prob_1 = 0.25 # отсев после pooling с вероятностью 0.25
drop_prob_2 = 0.5 # выпадение in the dense layer with probability 0.5
hidden_size = 512 # the dense layer will have 512 neurons

##########################################################################
########################
def load_data():
    (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')

    # Нормализация
    X_train /= np.max(X_train)  # Normalise data to [0, 1] range
    X_test /= np.max(X_train)  # Normalise data to [0, 1] range

    # Горячее кодирование меток
```

```
    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

    return X_train, Y_train, X_test, Y_test, num_train, depth, height,
width, num_test, num_classes
##############################################################################
########################

##############################################################################
########################
def build_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 = 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)

    if dropout:
        drop_2 = Dropout(drop_prob_1)(pool_2)
    else:
        drop_2 = pool_2
```

```
    # 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)  # Чтобы определить модель,
просто укажите ее 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
    return model
########################################################################
########################

########################################################################
########################
def create_graphics(history, label):
    # графики потерь
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, 'r')
    plt.plot(epochs, val_loss, 'b')
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.savefig(label + '_loss.png')
    plt.show()

    # графики точности
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    plt.plot(epochs, acc, 'g')
    plt.plot(epochs, val_acc, 'y')
    plt.title('Model accuracy')
```

```
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.savefig(label + '_acc.png')
    plt.show()
########################################################################
########################

########################################################################
########################
if __name__ == '__main__':
    X_train, Y_train, X_test, Y_test, num_train, depth, height, width,
num_test, num_classes = load_data()
    print("Введите, что вы хотите сделать:\n"
          "1 - Исходная сеть\n"
          "2 - Сеть без слоя Dropout\n"
          "3 - Исследование сети при разных размерах ядра свертки\n")
    num = input()

    if num == '1':
        model = build_model()
        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
        res = model.evaluate(X_test, Y_test, verbose=1)
        print(res)
        create_graphics(history, 'best')

    if num == '2':
        model = build_model(dropout=False)
        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
        create_graphics(history, 'drop')

    if num == '3':
        for kernels in [2, 5, 7]:
            model = build_model(kernels, True)
```

```
            history = model.fit(X_train, Y_train,
                                batch_size=batch_size,
epochs=num_epochs,
                                verbose=1, validation_split=0.1)
            create_graphics(history, str(kernels))

    res = model.evaluate(X_test, Y_test, verbose=1)  # Evaluate the
trained model on the test set!
    print(res)
```