**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. 7381 _____ Минуллин М.А.

Преподаватель _____ Жукова Н. А.

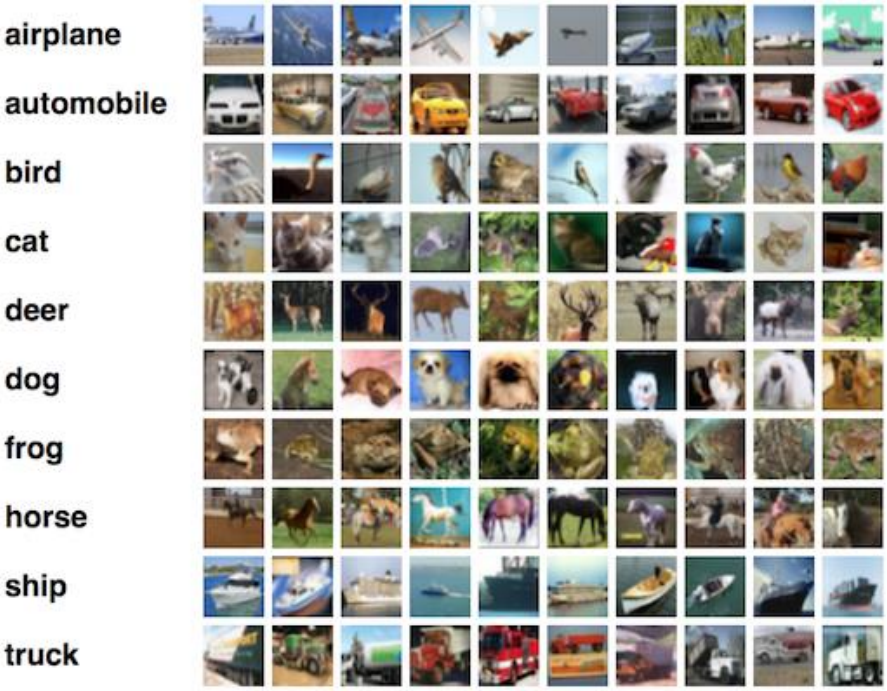
Санкт-Петербург

2020

**Цели.**

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)

**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:

```
batch_size = 32
num_epochs = 20
kernel_size = 3
```

```
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
drop_prob_1 = 0.25
drop_prob_2 = 0.5
hidden_size = 512
```

Обучение модели в течение 200 эпох заняло бы очень много времени. После обучения этой модели были получены следующие результаты:

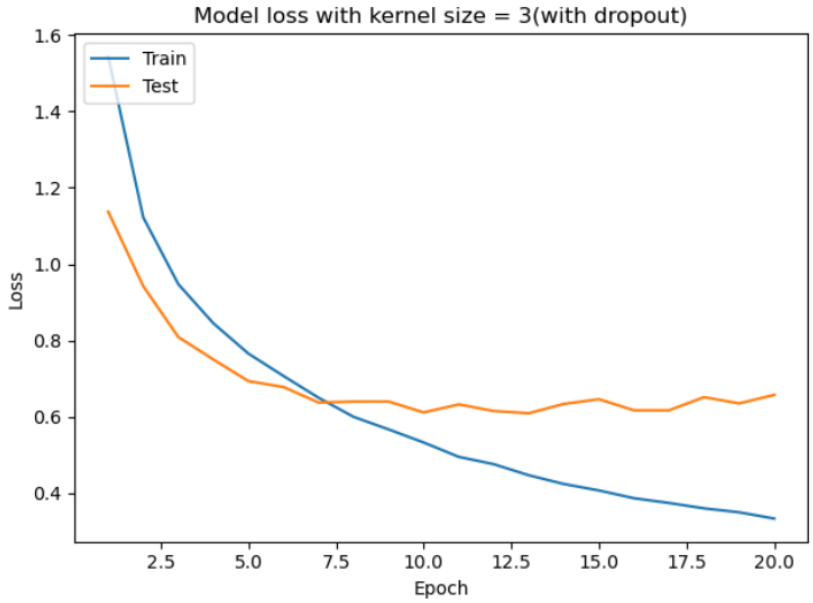



Рисунок 1 – Обучение сети при размерности ядра 3 на 3 с применением Dropout, точность.

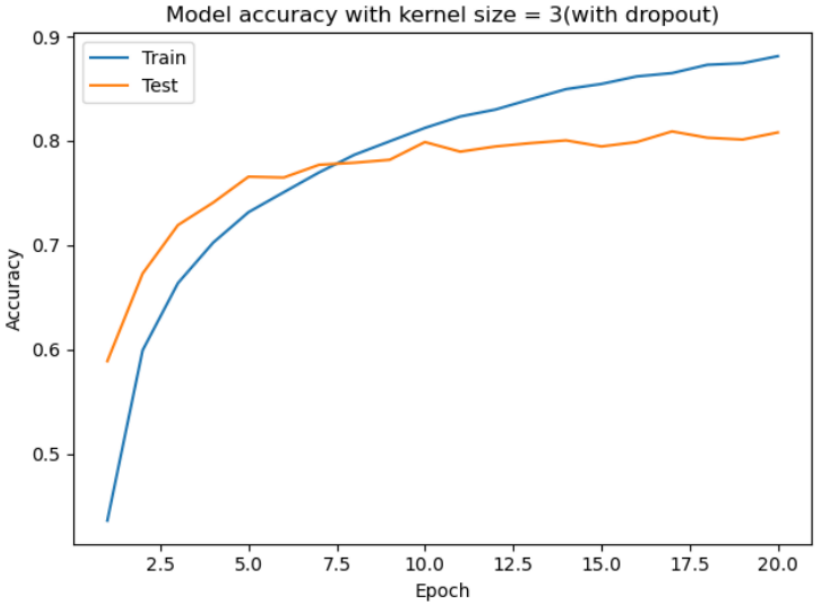



Рисунок 2 — Обучение сети при размерности ядра 3 на 3 с применением Dropout, потери.

Исследование работы сети без слоя Dropout:

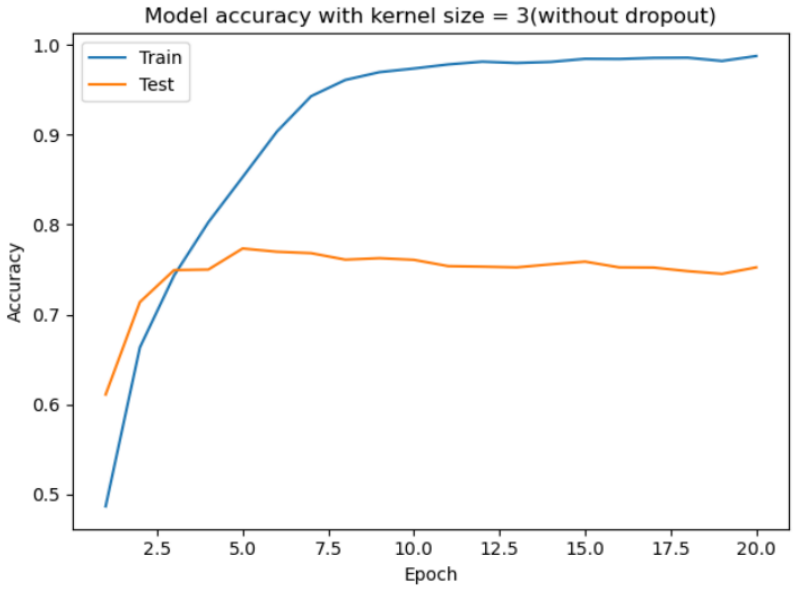



Рисунок 3 – Обучение сети при размерности ядра 3 на 3 без применения Dropout, точность.

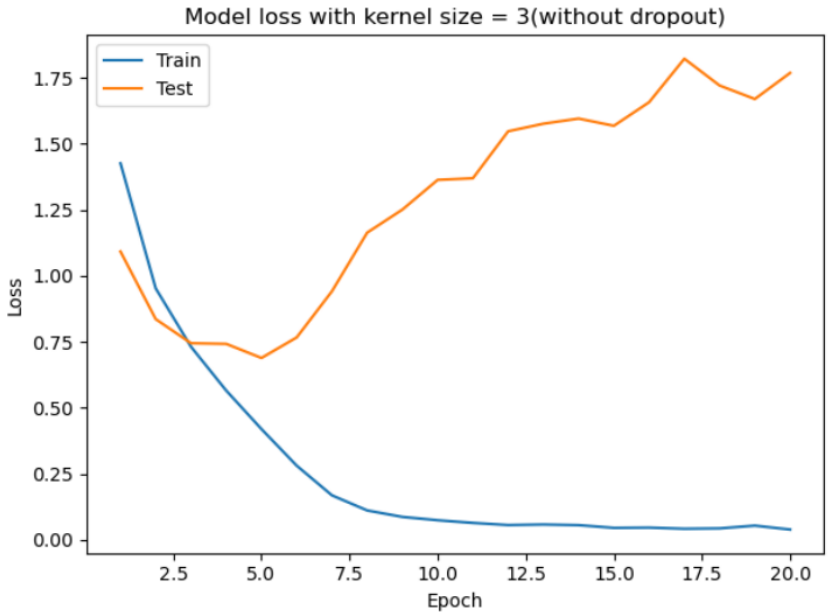
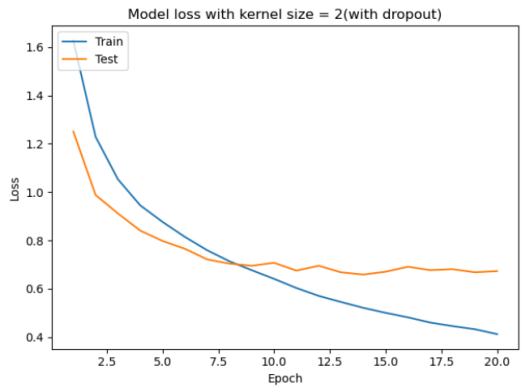


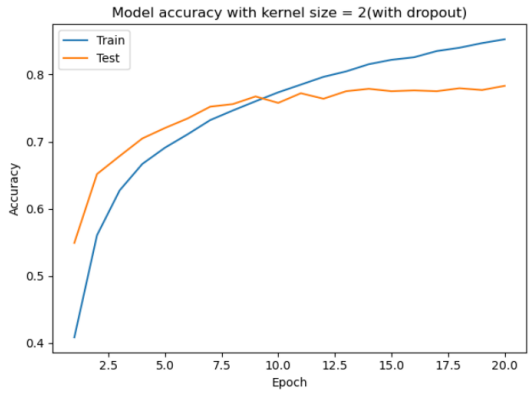

Рисунок 4 – Обучение сети при размерности ядра 3 на 3 без применения Dropout, потери.

Модель начала переобучаться примерно на 6 эпохе. По графикам можно убедиться в том, что сеть начинает переобучаться без использования метода dropout, исключающего нейроны с вероятностью p для вынуждения сети к обработке ошибок и ориентировки на консенсус нейронов. Это нужно, чтобы сеть не теряла способность к обобщению.

Исследование модели при различных размерностях ядра свертки.

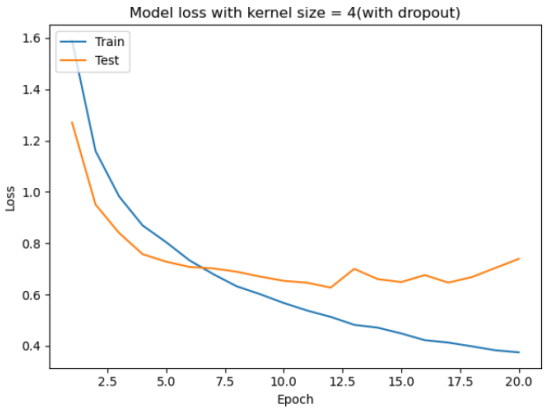



а

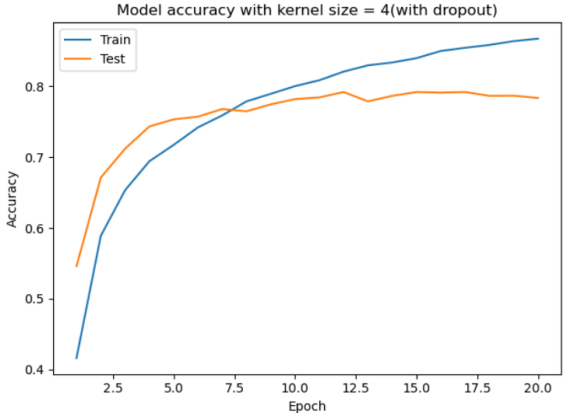



б

Рисунок 3 – Обучение сети при размерности ядра 2 на 2 с применением Dropout, потери (а) и точность (б).




а




б

Рисунок 4 – Обучение сети при размерности ядра 4 на 4 с применением Dropout, точность (а) и потери (б).

На точность и потери особо не повлияло изменение размера ядра свертки, протестировать более крупный размер матрицы свертки почти не представляется возможным на моей слабой машине, т. к. занимает очень много времени. Могу лишь сделать вывод о том, что самой оптимальной моделью представляется предложенная с ядром 3*3.

**Выводы.**

В ходе выполнения данной работы было произведено ознакомление со сверточными нейронными сетями: изучено построение модели в Keras в функциональном виде и изучена работа слоя разреживания (Dropout).

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

```
    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 plotter
    import numpy as np

    batch_size = 32
    num_epochs = 15
    kernel_size = 3
    pool_size = 2
    conv_depth_1 = 32
    conv_depth_2 = 64
    drop_prob_1 = 0.25
    drop_prob_2 = 0.5
    hidden_size = 512


    def singleModelPlots(kernel, history):
        plotter.plot(history.history['accuracy'])
        plotter.plot(history.history['val_accuracy'])
        plotter.title('Accuracy with ' + str(kernel) + "x" +
str(kernel) + ' kernel size')
        plotter.ylabel('Accuracy')
        plotter.xlabel('Epoch')
        plotter.legend(['Train', 'Test'], loc='upper left')
        plotter.show()

        plotter.plot(history.history['loss'])
        plotter.plot(history.history['val_loss'])
        plotter.title('Loss with ' + str(kernel) + "x" +
str(kernel) + ' kernel size')
        plotter.ylabel('Loss')
        plotter.xlabel('Epoch')
        plotter.legend(['Train', 'Test'], loc='upper left')
        plotter.show()
        return


    def test_kernels():
```

```
        kernels = [1, 3, 5, 7]
        for kernel in kernels:
            singleModelPlots(kernel=kernel,
history=buildModel(kernel))
        return


    def buildModel(kernel_size=3):
        main_input = Input(shape=(depth, height, width))

        conv_1 = Convolution2D(conv_depth_1, kernel_size,
kernel_size, border_mode='same', activation='relu')(main_input)
        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_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)

        flat = Flatten()(drop_2)  # (pool_2)#(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=main_input, output=out)
        model.compile(loss='categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])

        history = model.fit(X_train, Y_train,
                            batch_size=batch_size,
nb_epoch=num_epochs,
                            verbose=1, validation_split=0.1)
        model.evaluate(X_test, Y_test, verbose=1)
        return history


    if __name__ == '__main__':
        (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 /= np.max(X_train)
X_test /= np.max(X_train)

Y_train = np_utils.to_categorical(y_train, num_classes)
Y_test = np_utils.to_categorical(y_test, num_classes)

test_kernels()
```