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. 7383 _____ Бергалиев М.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы: Реализовать классификацию небольших изображений по десяти классам: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль и грузовик

Порядок выполнения работы.

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.

0. Исходный код программы представлен в приложении.

1. Выберем модель сети. Возьмем два сверточных слоя с 32 фильтрами с размером ядра свертки 3x3 с активацией relu, слой субдискретизации maxpooling с размером 2x2, слой разреживания dropout с вероятностью 0.2, два сверточных слоя с 64 фильтрами с размером ядра свертки 3x3 с активацией relu, слой субдискретизации maxpooling с размером 2x2, слой разреживания dropout с вероятностью 0.2, два сверточных слоя с 128 фильтрами с размером ядра свертки 3x3 с активацией relu, слой субдискретизации maxpooling с размером 2x2, слой разреживания dropout с вероятностью 0.2, слой с 512 нейронами с активацией relu, слой разреживания dropout с вероятностью 0.5 и выходной слой с 10 нейронами с активацией softmax. В качестве оптимизатора будем использовать SGD со скоростью обучения 0.1. Будем обучать 25 эпох с размером батча 100. Ошибка и точность во время обучения показаны на рис. 1, 2.

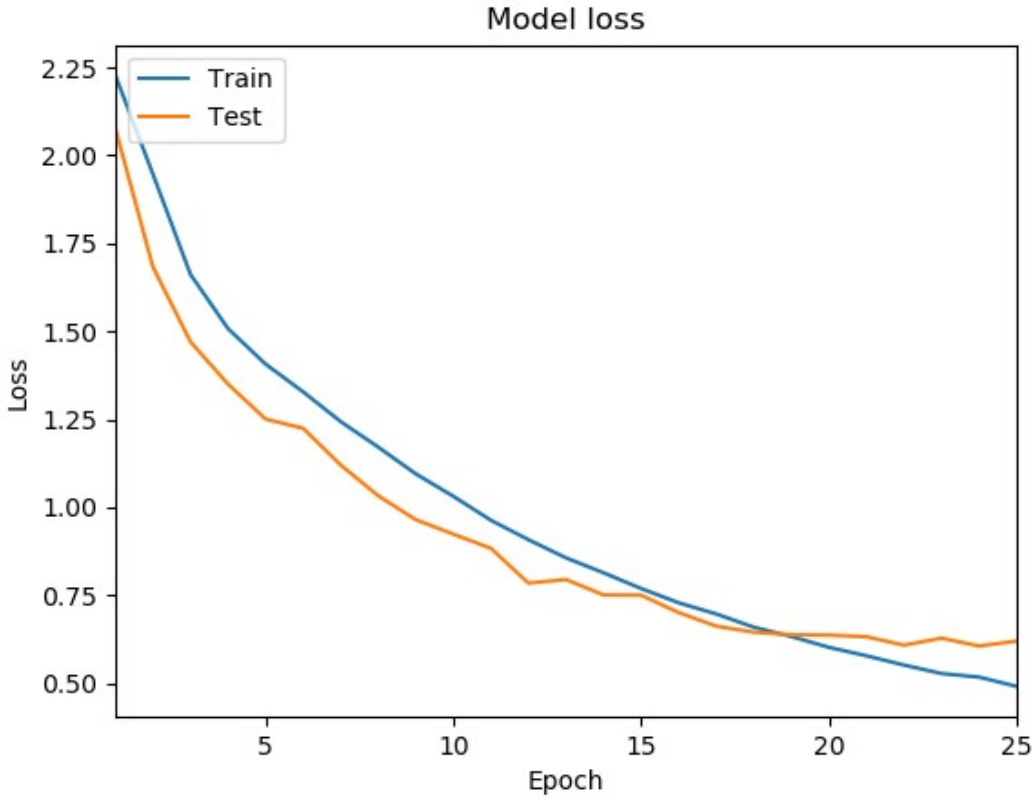

Рисунок 1 — График ошибок во время обучения модели

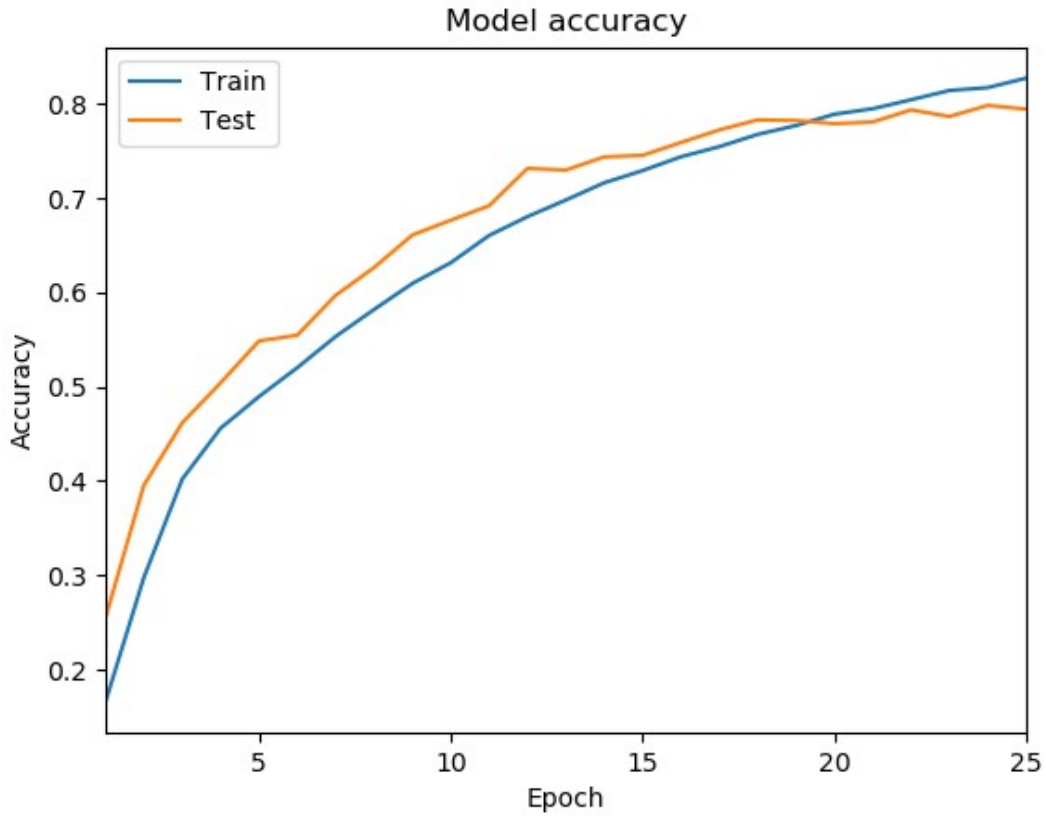

Рисунок 2 — График точности во время обучения модели

На проверочном наборе точность составила 80%.

2. Обучим модель без слоя разреживания. Результаты показаны на рис. 3-4.

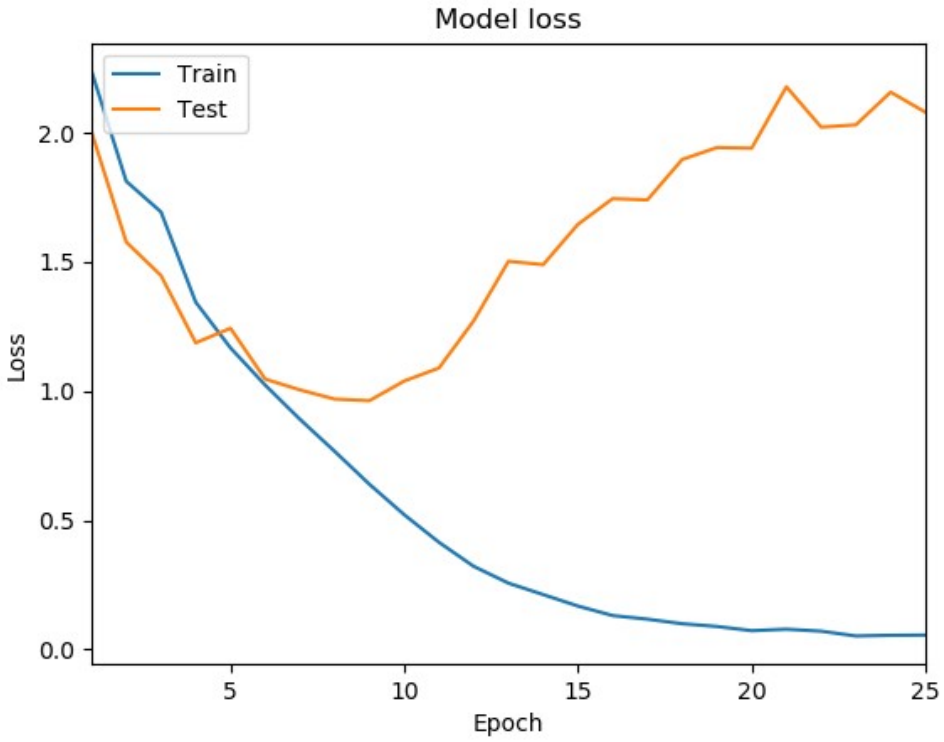

Рисунок 3 — График ошибок во время обучения модели без dropout

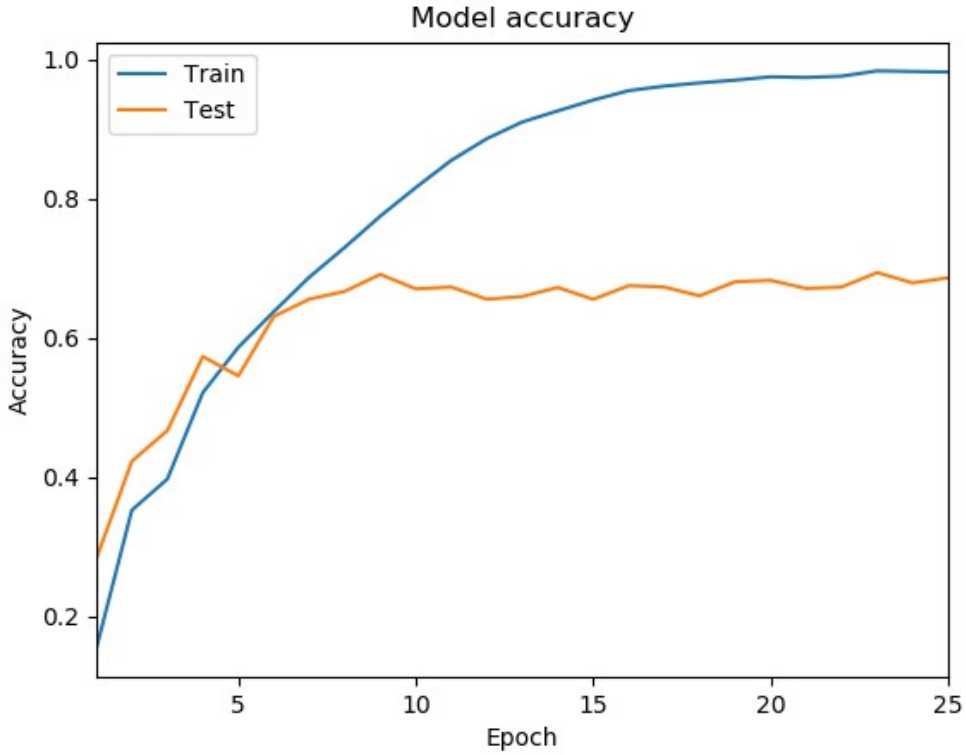

Рисунок 4 — График точности во время обучения модели без dropout

Как видно, после 9-ой эпохи начинается переобучение. Поэтому необходимо использование слоя разреживания. На проверочном наборе точность составила 67%.

3. Обучим модель с различными размерами ядра свертки. Результаты для размеров 2x2, 5x5, 7x7 показаны на рис. 5-10.

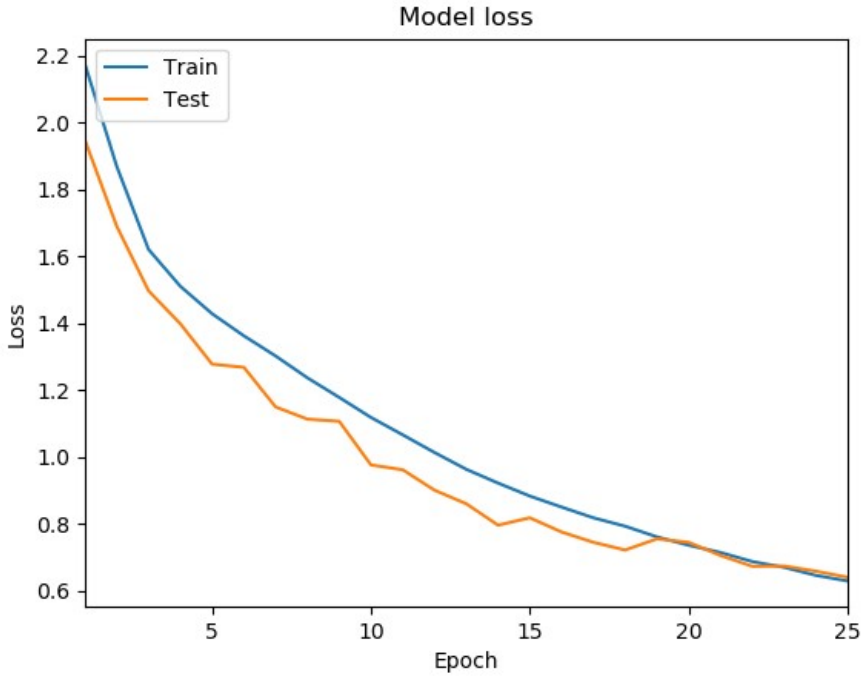

Рисунок 5 — График ошибок при размере ядра свертки 2x2

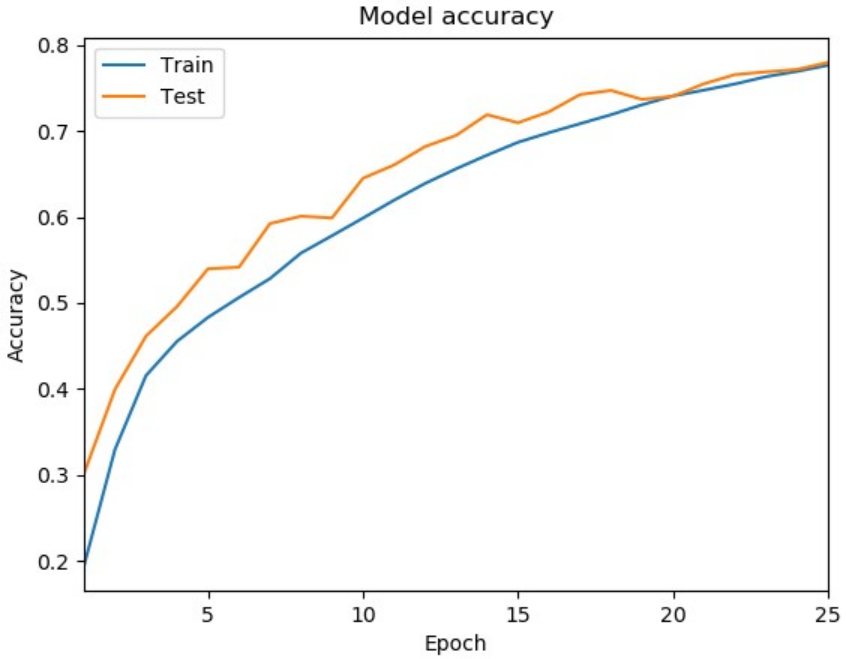

Рисунок 6 — График точности при размере ядра свертки 2x2

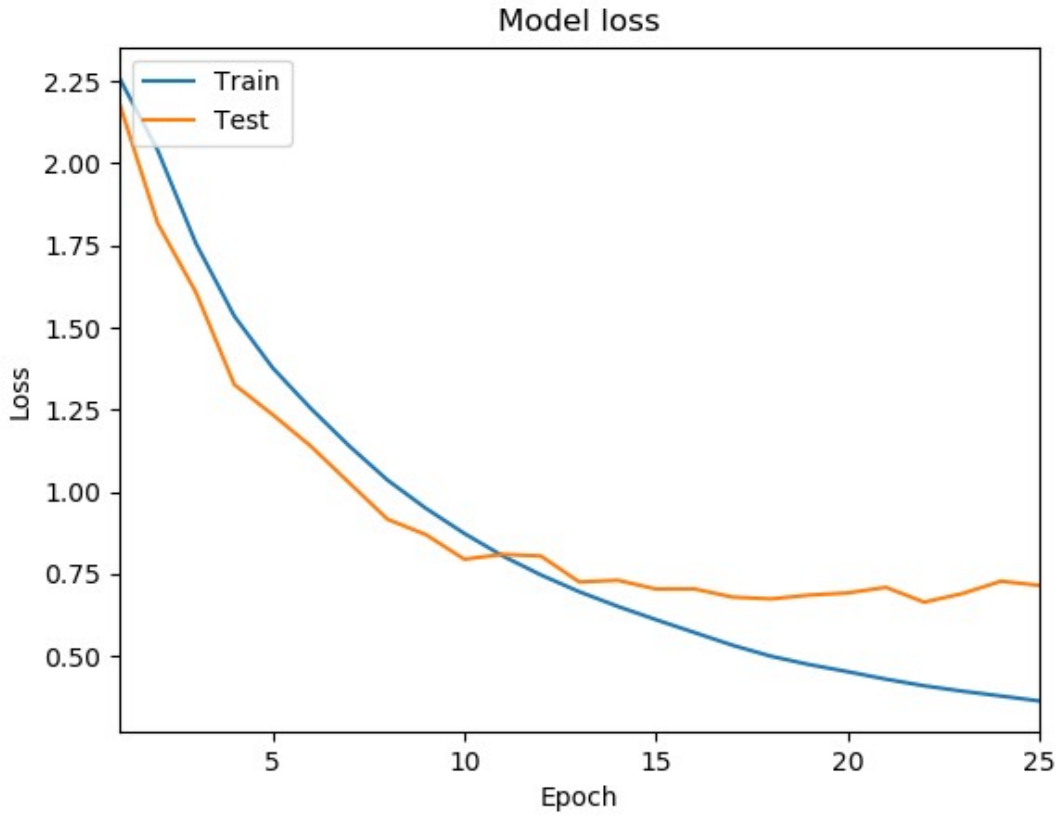

Рисунок 7 — График ошибок при размере ядра свертки 5x5

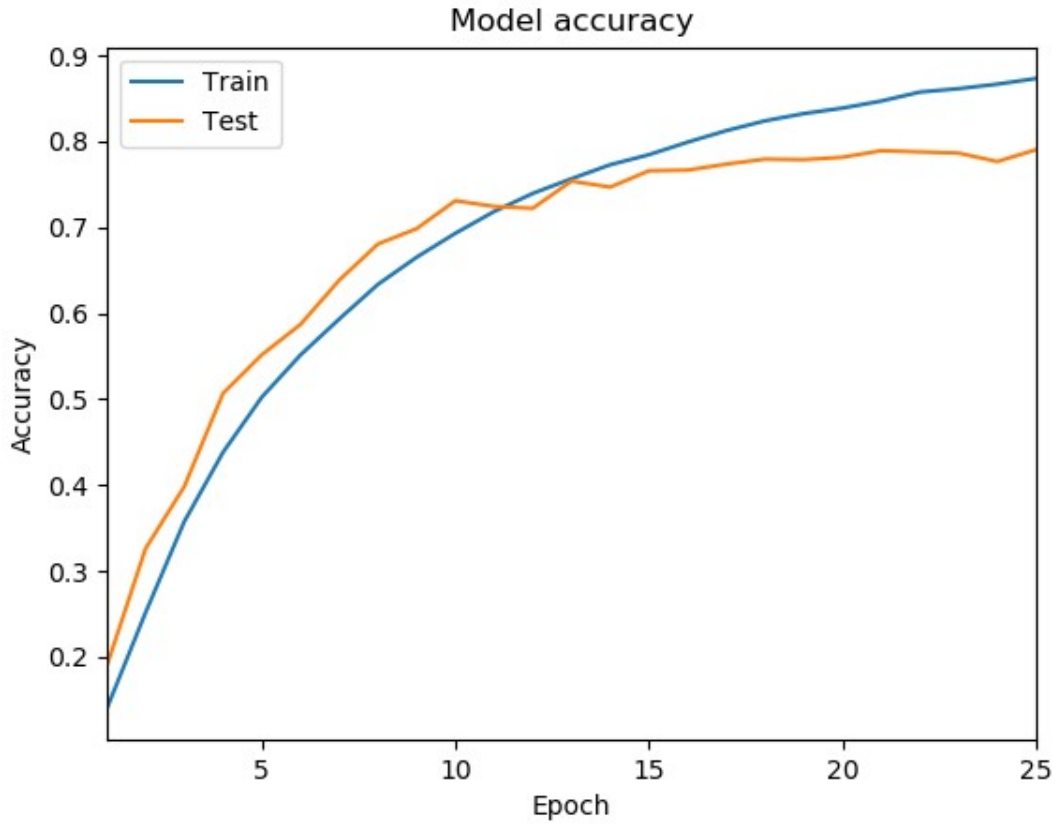

Рисунок 8 — График точности при размере ядра свертки 5x5

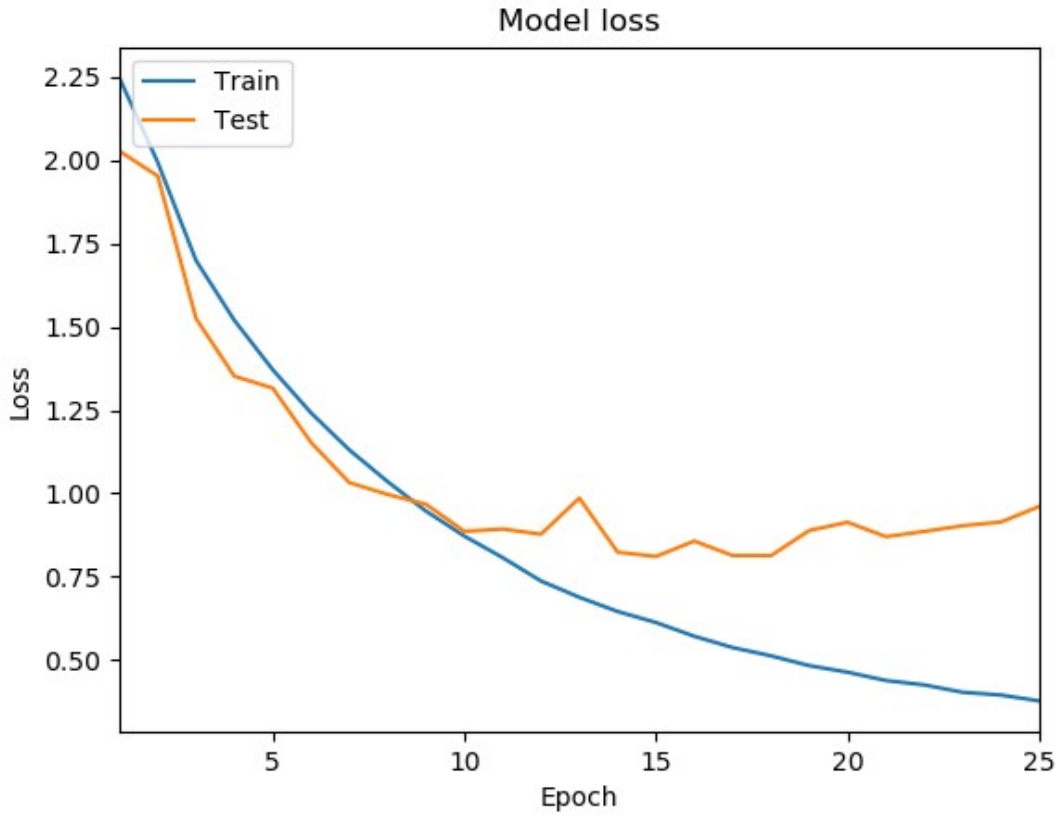

Рисунок 9 — График ошибок при размере ядра свертки 7x7

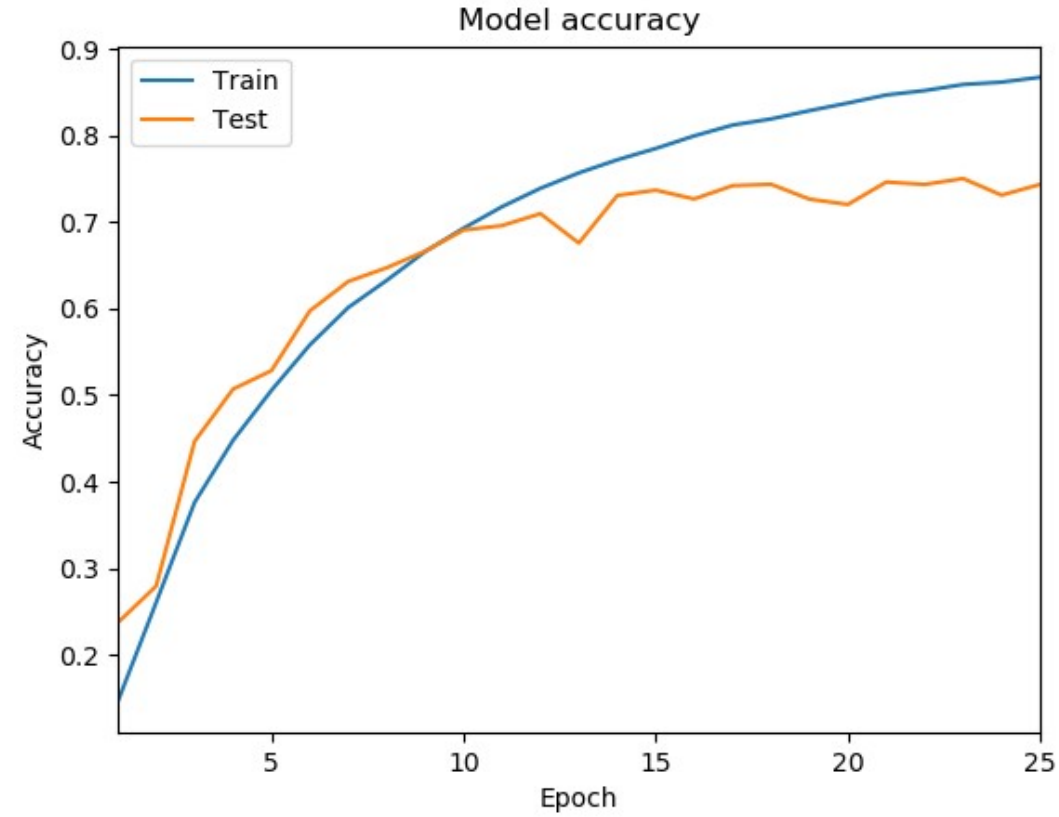

Рисунок 10 — График точности при размере ядра свертки 7x7

Как видно, с увеличением размера ядра свертки при неизменных остальных параметров сети переобучение возникает раньше, а точность падает. Для достижения лучших результатов нужно менять и другие параметры модели. На проверочном наборе точность составила 76%, 77% и 72% соотвественно.

Выводы: Было изучено влияние слоя разреживания на результат обучения. Данный слой помогает бороться с проблемой переобучения. При изменении размера ядра свертки необходимо корректировать всю модель.

ПРИЛОЖЕНИЕ

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.layers import Input, Convolution2D,
MaxPooling2D, Dense, Dropout, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
from time import sleep

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
num_train, width, height, depth = 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)

batch_size = 100
num_epochs = 25
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
conv_depth_3 = 128
drop_prob_1 = 0.2
drop_prob_2 = 0.5
hidden_size = 512

def build_model(kernel_size=3, with_dropout=True):
    inp = Input(shape=(width, height, depth))
          conv_1 = Convolution2D(conv_depth_1, (kernel_size,
kernel_size),
                                 padding='same', strides=(1,1),
activation='relu')(inp)
          conv_2 = Convolution2D(conv_depth_1, (kernel_size,
kernel_size),
                                 padding='same', activation='relu')
(conv_1)
        pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))
(conv_2)
    if with_dropout:
        drop_1 = Dropout(drop_prob_1)(pool_1)
```

```
    else:
        drop_1 = pool_1
           conv_3   =   Convolution2D(conv_depth_2,   (kernel_size,
kernel_size),
                                      padding='same', strides=(1,1),
activation='relu')(drop_1)
           conv_4   =   Convolution2D(conv_depth_2,   (kernel_size,
kernel_size),
                                      padding='same', strides=(1,1),
activation='relu')(conv_3)
        pool_2  =  MaxPooling2D(pool_size=(pool_size,  pool_size))
(conv_4)
    if with_dropout:
        drop_2 = Dropout(drop_prob_1)(pool_2)
    else:
        drop_2 = pool_2
           conv_5   =   Convolution2D(conv_depth_3,   (kernel_size,
kernel_size),
                                      padding='same', strides=(1,1),
activation='relu')(drop_2)
           conv_6   =   Convolution2D(conv_depth_3,   (kernel_size,
kernel_size),
                                      padding='same', strides=(1,1),
activation='relu')(conv_5)
        pool_3  =  MaxPooling2D(pool_size=(pool_size,  pool_size))
(conv_6)
    if with_dropout:
        drop_3 = Dropout(drop_prob_1)(pool_3)
    else:
        drop_3 = pool_3
    flat = Flatten()(drop_3)
    hidden = Dense(hidden_size, activation='relu')(flat)
    if with_dropout:
        drop_4 = Dropout(drop_prob_2)(hidden)
    else:
        drop_4 = hidden
    out = Dense(num_classes, activation='softmax')(drop_4)
    model = Model(inp, out)
    model.compile(loss='categorical_crossentropy',
                  optimizer=SGD(0.1),
                  metrics=['accuracy'])
    return model

def plot_res(history, label):
    x = range(1, num_epochs+1)
    plt.plot(x, history.history['loss'])
```

```
    plt.plot(x, history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.xlim(x[0], x[-1])
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.savefig(label+'_loss.png')
    plt.clf()
    plt.plot(x, history.history['accuracy'])
    plt.plot(x, history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.xlim(x[0], x[-1])
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.savefig(label+'_acc.png')
    plt.clf()

def test_model():
    model = build_model()
    history = model.fit(X_train, Y_train,
              batch_size=batch_size, epochs=num_epochs,
              verbose=1, validation_split=0.1)
    res = model.evaluate(X_test, Y_test, verbose=0)
    print(res)
    plot_res(history, 'best')

def test_without_dropout():
    model = build_model(with_dropout=False)
    history = model.fit(X_train, Y_train,
              batch_size=batch_size, epochs=num_epochs,
              verbose=1, validation_split=0.1)
    res = model.evaluate(X_test, Y_test, verbose=1)
    print(res)
    plot_res(history, 'without_dropout')

def test_kernel_size():
    for k in [2, 5, 7]:
        model = build_model(k)
        history = model.fit(X_train, Y_train,
              batch_size=batch_size, epochs=num_epochs,
              verbose=1, validation_split=0.1)
        res = model.evaluate(X_test, Y_test, verbose=1)
        print(res)
        plot_res(history, str(k))
```

```
def main():
    test_model()
    test_without_dropout()
    test_kernel_size()

if __name__ == '__main__':
    main()
```