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»

Студентка гр. 7381 _____ Машина Ю.Д.

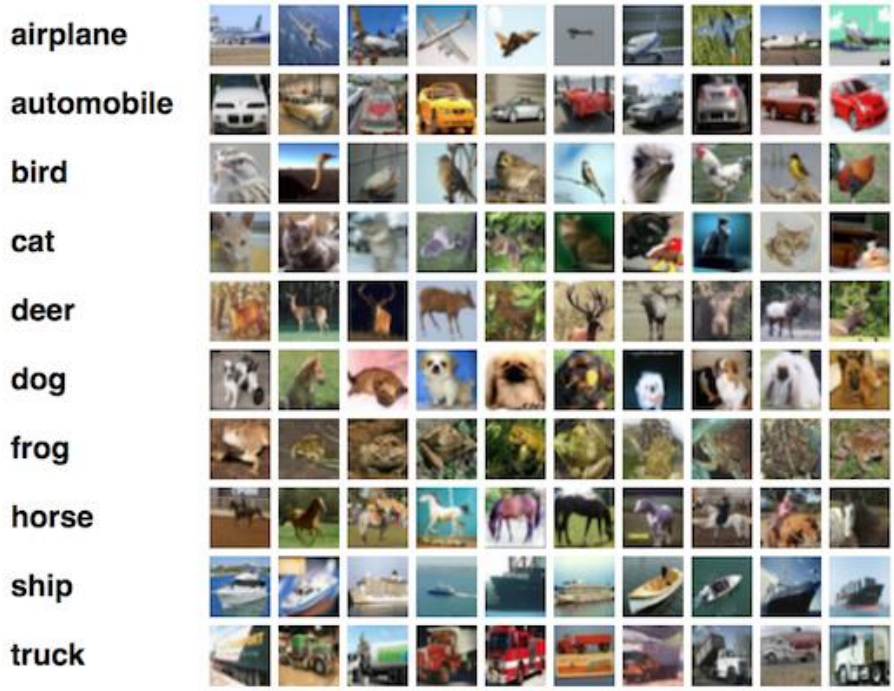
Преподаватель _____ Жукова Н. А.

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)

Выполнение работы.

1) Построить и обучить сверточную нейронную сеть.

Исходная (предложенная) сеть, а именно: Conv [32] -> Conv [32] -> Pool (with dropout on the pooling layer) —> Conv [64] -> Conv [64] -> Pool (with dropout on the pooling layer)—> Flatten+Dense -> ReLU (with dropout) -> softmax, со следующими параметрами:

```
      batch_size = 32 # in each iteration, we consider 32
training examples at once
      num_epochs = 200 # we iterate 200 times over the entire
training set
      kernel_size = 3 # 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
```

на Intel Core i7-6800K 3.4GHz обучалась 18867 секунд (5,24 часов) с 90%-ой загрузкой CPU.

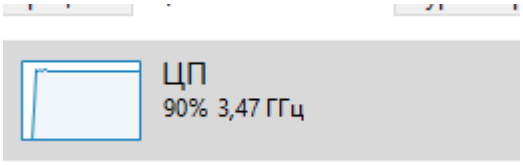
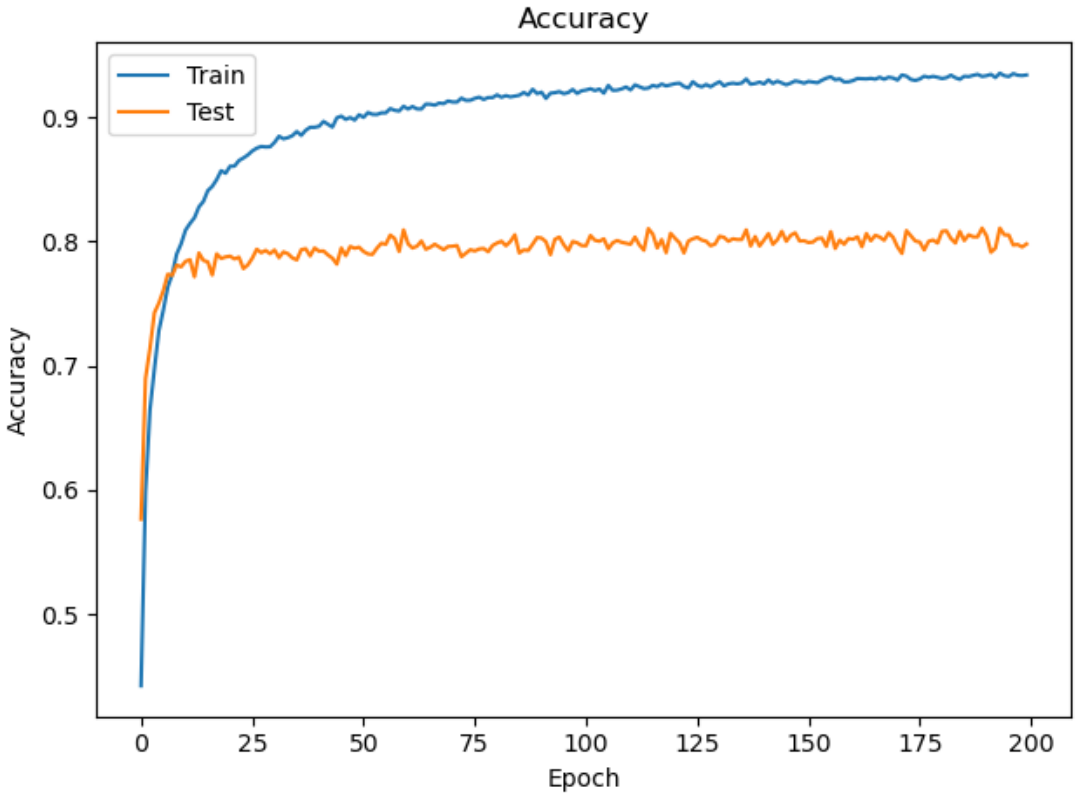


Рисунок 1 — Нагрузка на CPU на протяжении процесса обучения.



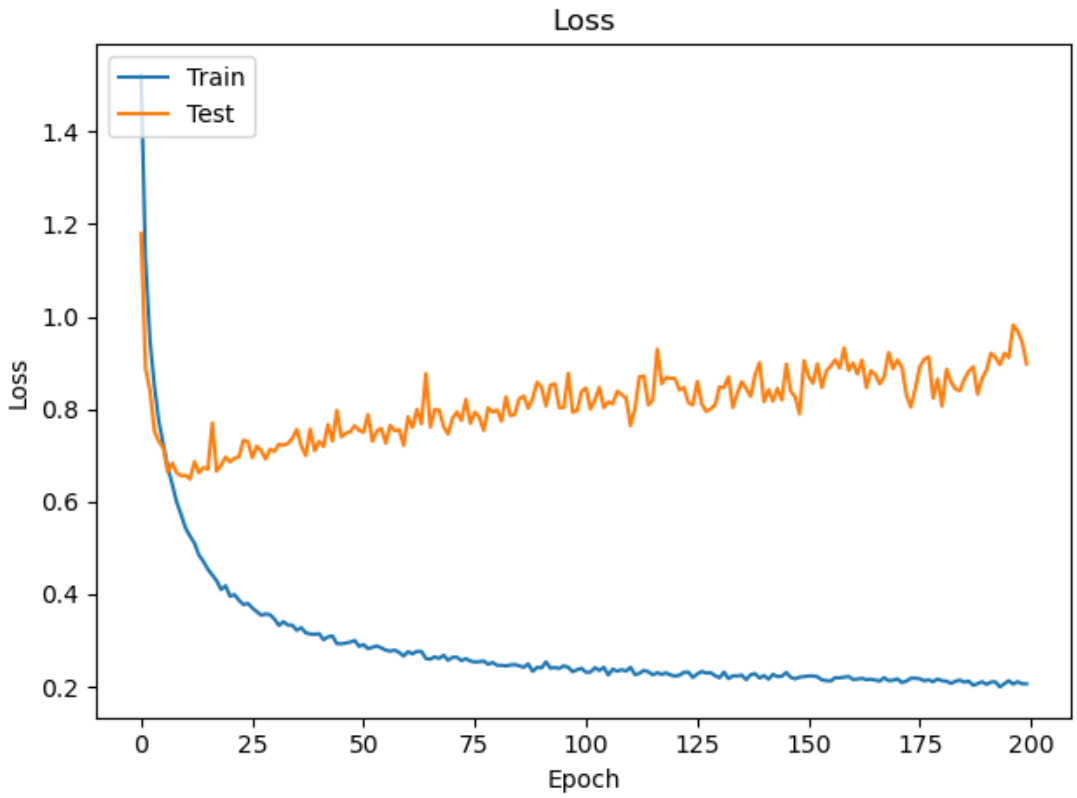
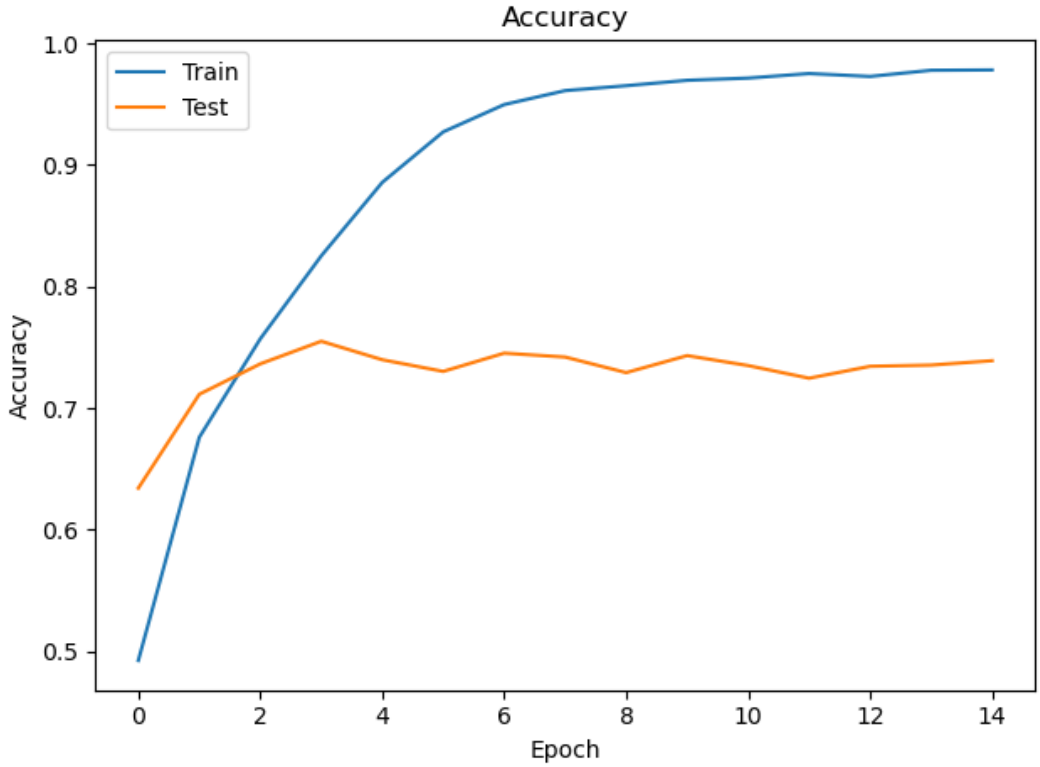


Рисунок 2 — Точность и потери построенной модели СНС.

Как видно по графикам, иллюстрирующим результаты обучения, точность на тестировании не поднимается выше 80%, а потери имеют

тенденцию увеличиваться на каждой эпохе, следующей за примерно 13 эпохой, поэтому я снижу кол-во эпох до 15, чтобы ускорить процесс изучения СНС, так как на эпоху в среднем уходит 95 секунд при 3 ядрах свертки. А из того, что видно по графикам, я делаю предположение о том, что после 15 эпох начинается переобучение: сеть повышает точность на обучении, а результат на тестах не улучшается, потери на тестах растут.

2) Исследовать работу сети без слоя Dropout.



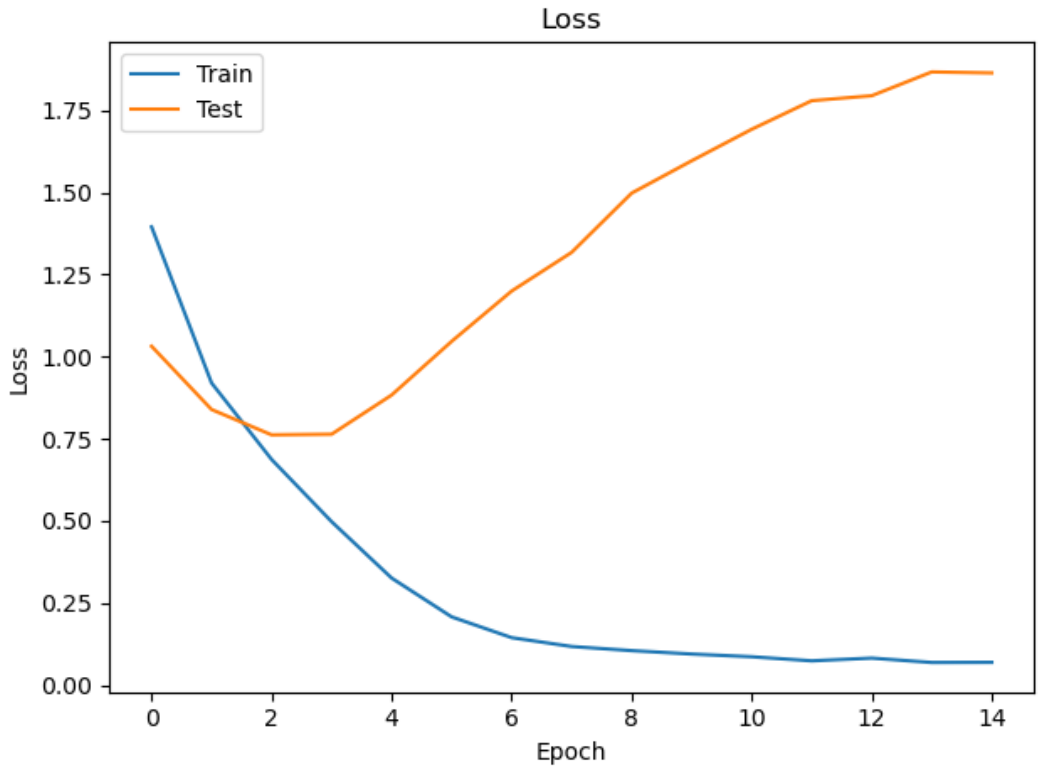
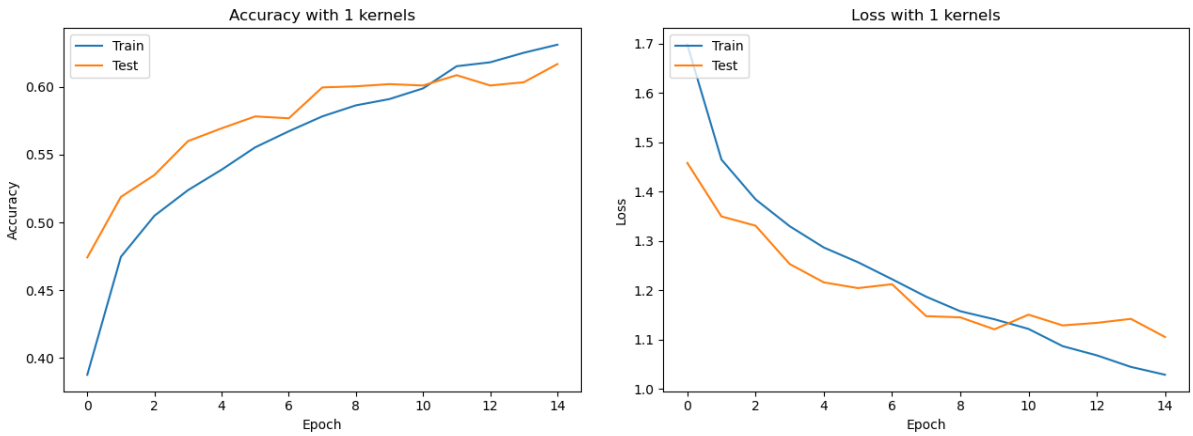


Рисунок 3 — Точность и потери сети без слоя Dropout.

Без этого приема регуляризации в сети наступило переобучение еще раньше, чем через 13 эпох: через 3.

3) Исследовать работу сети при разных размерах ядра свертки.



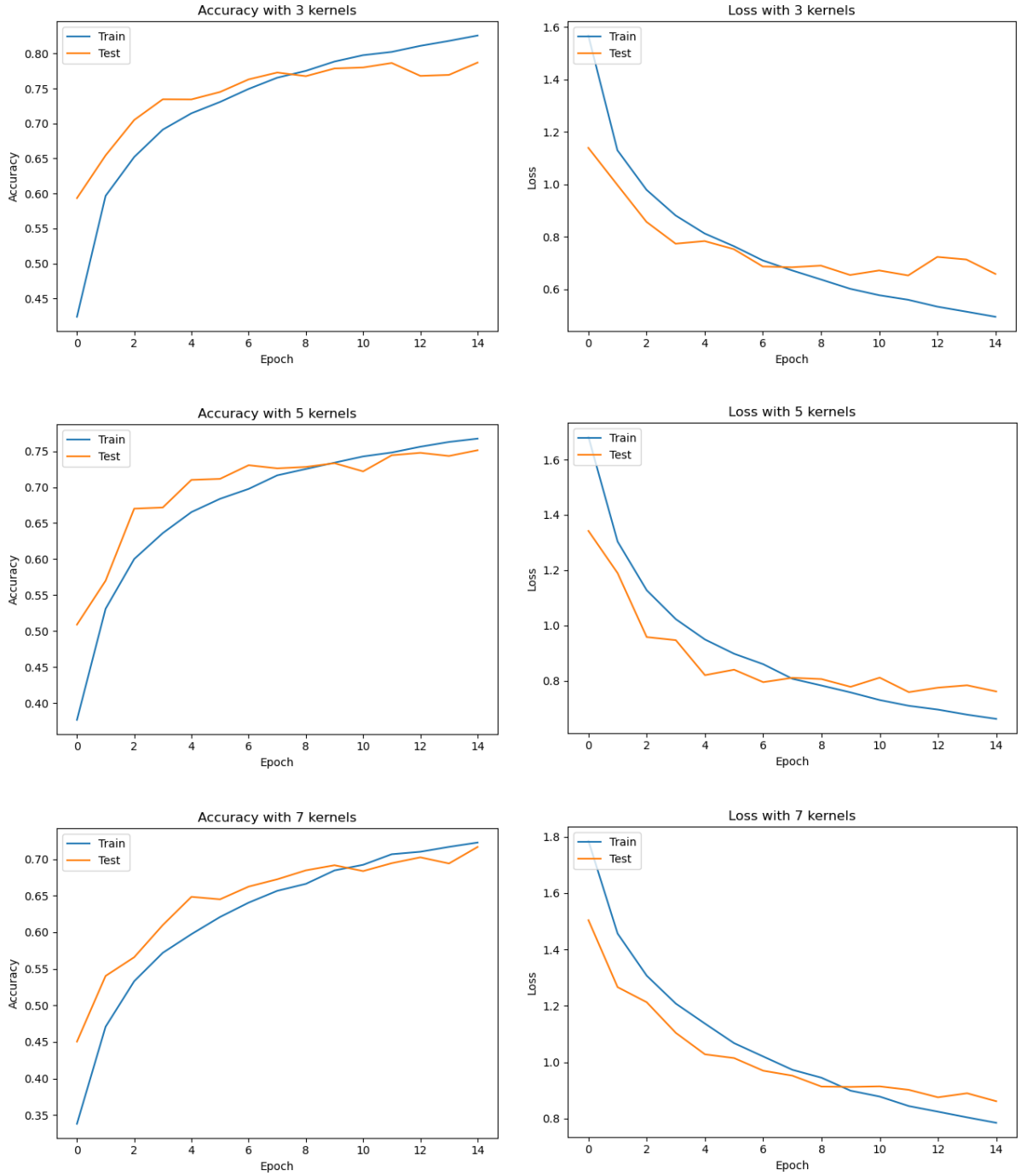


Рисунок 3 — Графики точности и потерь сети с разными размерами ядер: 1x1, 3x3, 5x5, 7x7. В названиях графиков опечатка, должно быть «… with XxX kernels», перезапуск программы бы занял много времени, поэтому я оставила так. А то на одну эпоху обучения заданной сети с ядрами 7x7 уходит 4,4 минуты.

Самой оптимальной размерностью матрицы свертки оказалась 3x3. Размерностью ядра можно контролировать количество feature maps (глубину тензора признаков). Например, изображение размером 200 x 200 с

50 признаками при свертке с 20 фильтрами (ядрами) 1x1 приведет к размеру 200 x 200 x 20.

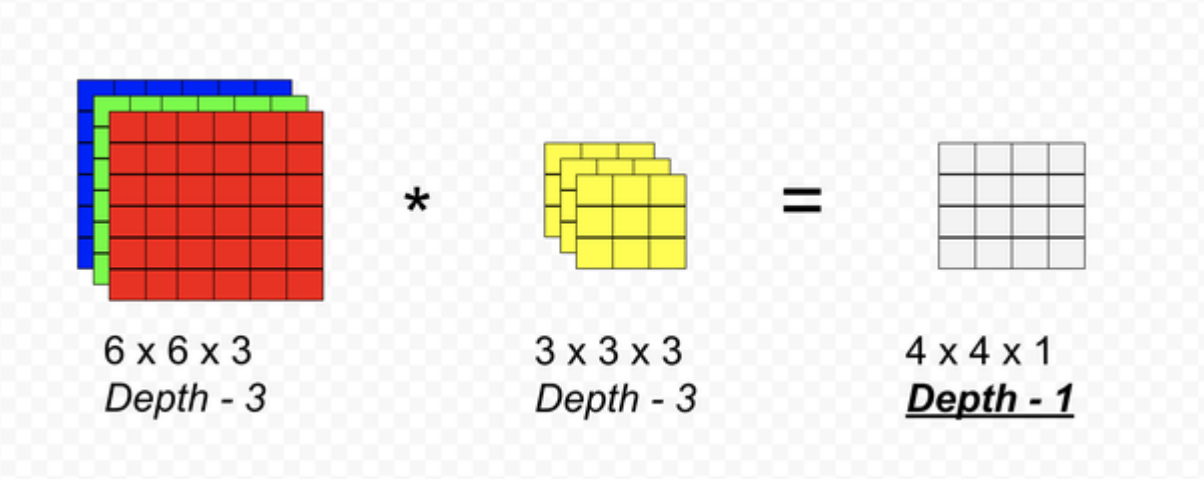


Рисунок 4 — На самом деле размерность ядра не 1x1, а 1 x 1 x K, где K – количество карт признаков (feature maps).

Каждый сверточный уровень преобразует предыдущую карту признаков в несколько карт последующих признаков, и их можно как раз представить в виде 3D массива. Ядра свертки разные и умножают цветовые коды пикселей на неодинаковые значения.

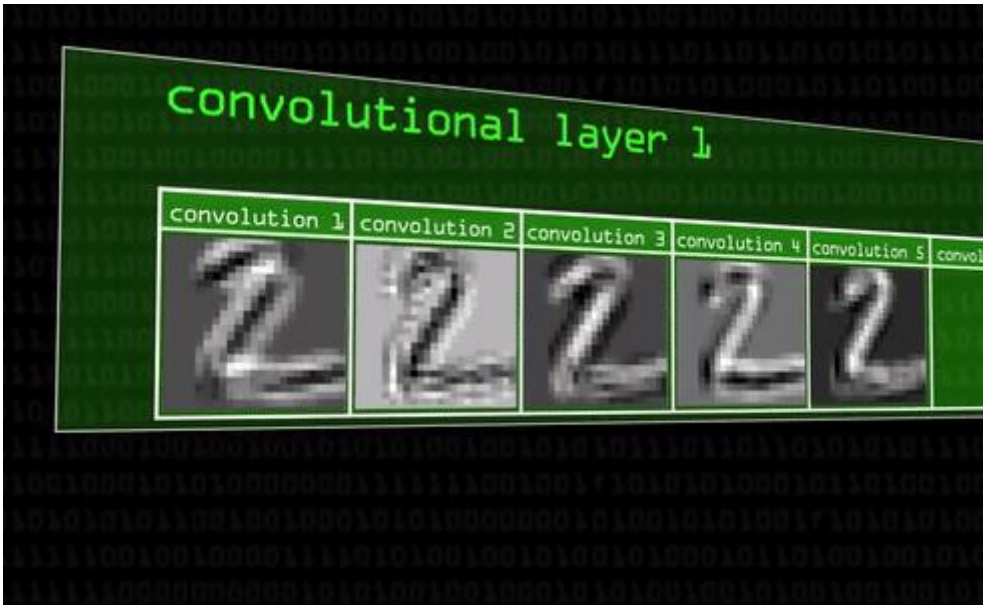


Рисунок 5 — В результате выходит набор визуально похожих изображений, и это выглядит как выделение уникальных черт объекта (например, вертикальных или диагональных краев) белыми пикселями.

Исходя из этой информации, предполагаю, что точность снизилась для 5x5 и 7x7 размерностей из-за того, что матрица 3x3 уловила больше мелких деталей, которые и помогли в итоге классифицировать картинку точнее. А ядро размерностью 1x1 уменьшило глубину тензора признаков (K), наверное это послужило уменьшению точности по сравнению с другими протестированными размерностями.

Вывод.

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

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

batch_size = 32 # in each iteration, we consider 32 training
examples at once
num_epochs = 15 # we iterate 200 times over the entire training set
kernel_size = 3 # 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

import time


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

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Loss with '+str(kernel)+"x"+str(kernel)+' kernel
size')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
```

```
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.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):
    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)
    drop_1 = Dropout(drop_prob_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)  # (pool_1)#(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)
    # Now flatten to 1D, apply Dense -> ReLU (with dropout) ->
softmax
    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)  #
(hidden)#(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
    # timing = time.time()
```

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

if __name__ == '__main__':

    (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

    #singleModelPlots(buildModel())
    test_kernels()

    #print(history.history)
```