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

ОТЧЕТ

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

Студент гр. 8382 _____ Облизанов А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель.

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).

Задание.

- 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

Выполнение работы.

Работа выполнялась на базе операционной системы Windows 10 в среде разработки PyCharm и в онлайн сервисе Google Colab.

Была создана функция load_data (см. Приложение А) для загрузки датасета cifar10 из TensorFlow. В функции также проводится нормализация данных, сохраняются размеры тензоров, классы изображений приводятся к категориальному виду.

Был создан класс CNN, который отвечает за создание и обучение модели: с слоями Dropout и без, а также с различными размерами ядер свертки.

Определим переменные, которые будут изменяться при вызовах создания модели:

- input_shape – число входных параметров
- ksize – размер ядра свертки

- num_classes – число классов

Модель нейронной сети представлена в табл. 1.

Таблица 1

Слой	Ф-ция акт.	Параметры	Примечание
Input	-	Shape = input_shape	
Conv2D	Relu	Kernel_size = (ksize, ksize), Filters = 32	
Conv2D	Relu	Kernel_size = (ksize, ksize) Filters = 32	
MaxPool	-	Pool_size = (2,2)	
Dropout	-	Rate = 0.25	Опционально
Conv2D	Relu	Kernel_size = (ksize, ksize), Filters = 64	
Conv2D	Relu	Kernel_size = (ksize, ksize) Filters = 64	
MaxPool	-	Pool_size = (2,2)	
Dropout	-	Rate = 0.25	Опционально
Flatten	-	-	
Dense	Relu	Units = 512	
Dropout	-	Rate = 0.5	Опционально
Dense	Softmax	Units = num_classes	

Параметр k_size, а также флаг, отвечающий за включение в модель слоев Dropout, позволяют создавать различные модели.

Модель компилируется со следующими параметрами:

- Функция потерь – categorial crossentropy

$$CCE(p,t) = -\sum_{c=1}^{C} t_{o,c} \log\left(p_{o,c}\right)$$

- Оптимизатор – Adam
- Метрика – точность

Обучение происходит со следующими параметрами:

- Число эпох: 20
- Batch_size: 80

Результаты моделей

Модель №1 имеет следующие особенности:

- Слои Dropout включены
- Размер ядра свертки: 3x3

Графики точности и потерь представлены на рис. 1-2.

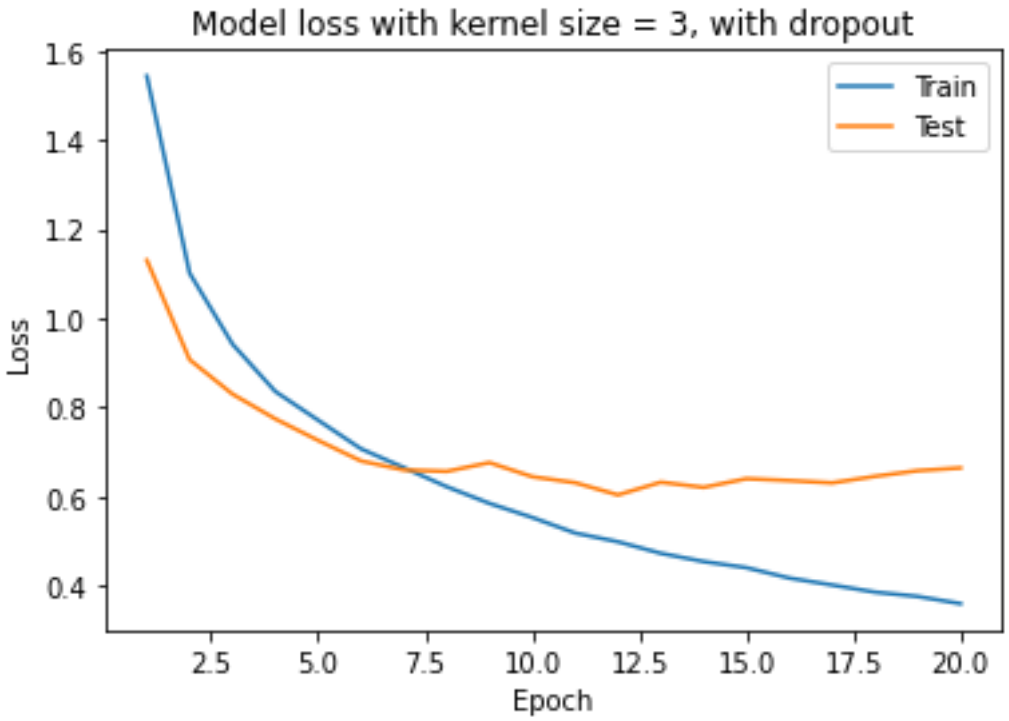

Рисунок 1 – Потери, модель №1

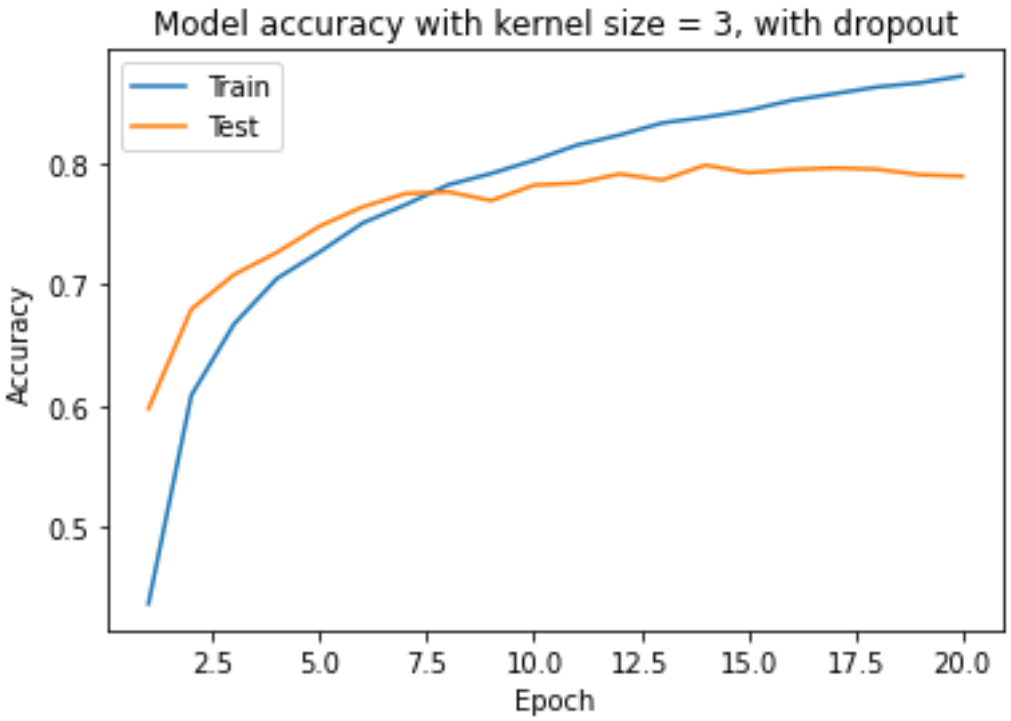

Рисунок 2 – Точность, модель №1

Как видно из графиков, модель достигла точки переобучения на 14 эпохе (после нее точность на данных для проверки падает, а потери увеличиваются). Для наглядности сравнения различных моделей обучение на 14-ти эпохах на данных момент проводиться не будет.

Показатели модели после обучения представлены в табл. 2.

Таблица 2

Loss	Accuracy	Val. loss	Val. accuracy	Step time
0.3495	0.8749	0.6649	0.7894	8 ms

Модель №2 имеет следующие особенности:

- Слои Dropout не включены
- Размер ядра свертки: 3x3

Графики точности и потерь представлены на рис. 3-4.

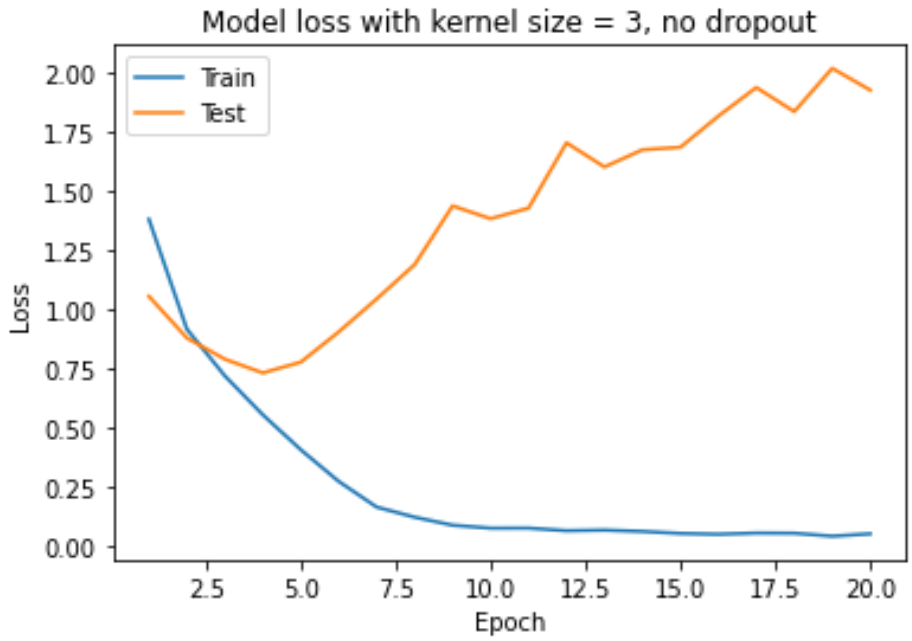

Рисунок 3 – Потери, модель №2

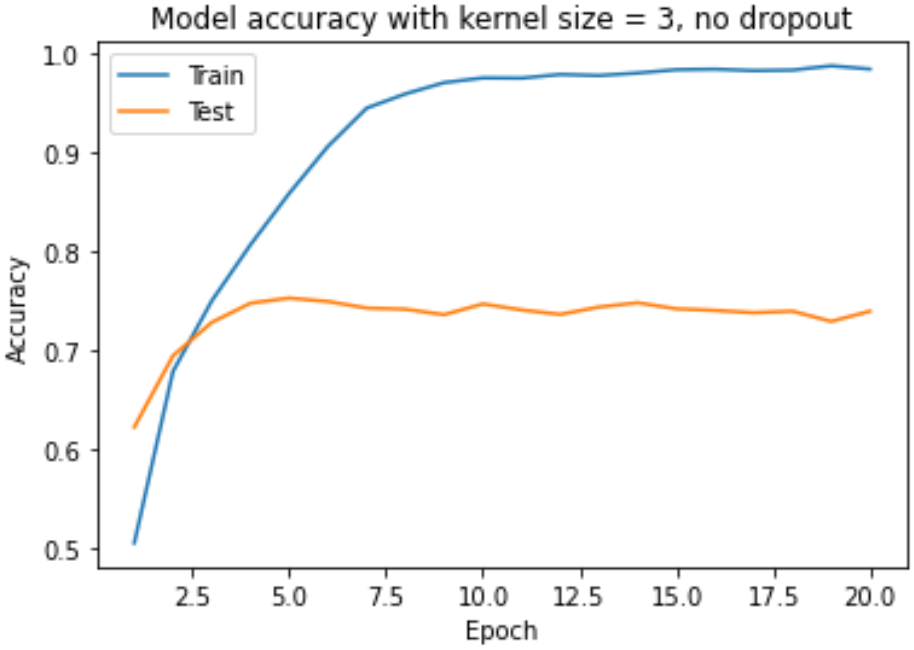

Рисунок 4 – Точность, модель №2

Показатели модели представлены в табл. 3.

Таблица 3

Loss	Accuracy	Val. loss	Val. accuracy	Step time
0.0499	0.9847	1.9231	0.7400	8 ms

Как видно из графиков, модель достигла точки переобучения уже на 4 эпохе, и переобучение выражается очень явно, так как значение ошибок на данных для проверки вырастает до значения 2, а точность колеблется около значений 0.7-0.75. Отсутствие слоев Dropout делает модель склонной к быстрому переобучению.

Слой Dropout за одну итерацию обучения проходит по всем нейронам определенного слоя и с вероятностью $rate$ полностью исключает их из сети на время итерации. Это позволяет уменьшить влияние определенных цепочек нейронов, которые могут отвечать в том числе за признаки, присущие только обучающей выборке. Исключение хотя бы одного из нейронов равносильно обучению новой нейронной сети, таким образом, слой Dropout позволяет создать ансамбль нейронных сетей, результаты которых потом усредняются.

Модель №3 имеет следующие особенности:

- Слои Dropout включены
- Размер ядра свертки: 2x2

Графики точности и потерь представлены на рис. 5-6.

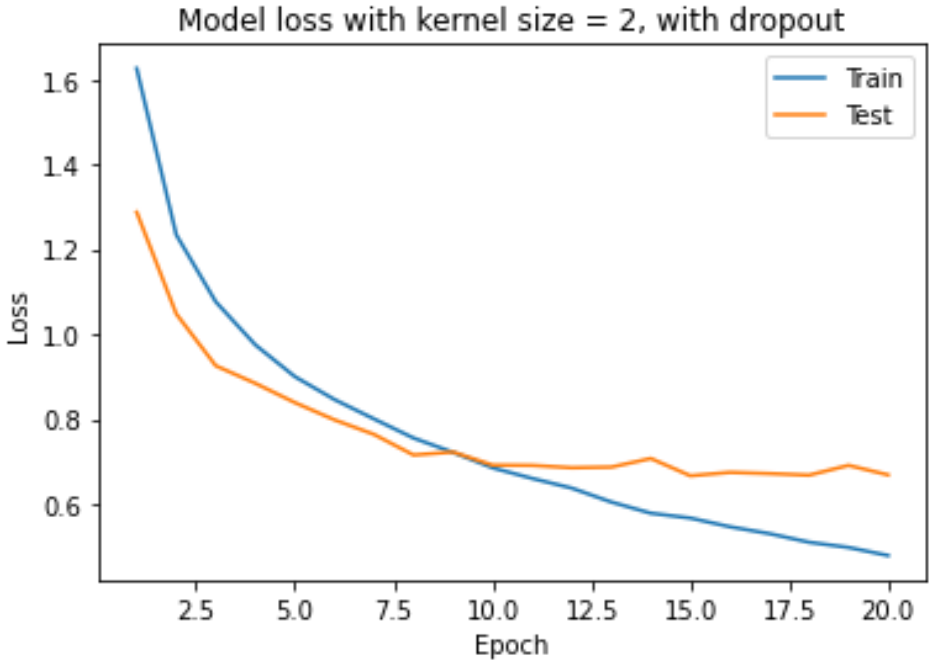

Рисунок 5 – Потери, модель №3

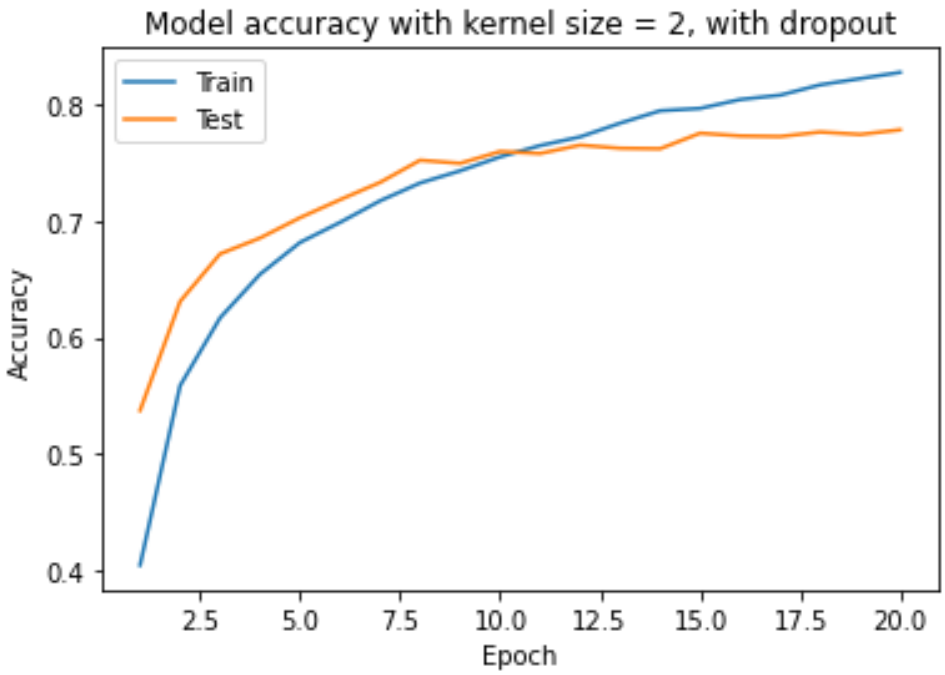

Рисунок 6 – Точность, модель №3

Показатели модели представлены в табл. 4.

Таблица 4

Loss	Accuracy	Val. loss	Val. accuracy	Step time
0.4707	0.8328	0.6689	0.7789	9 ms

При ядре свертки с размером (2,2) переобучения на 20-ти эпохах не возникает. Однако показатель точности на данных для проверки оказался ниже, чем при ядре свертки с размером (3,3). Вероятно, при малом ядре свертки сети сложнее выявить признаки для классификации изображений.

Модель №4 имеет следующие особенности:

- Слои Dropout включены
- Размер ядра свертки: 4x4

Графики точности и потерь представлены на рис. 7-8.

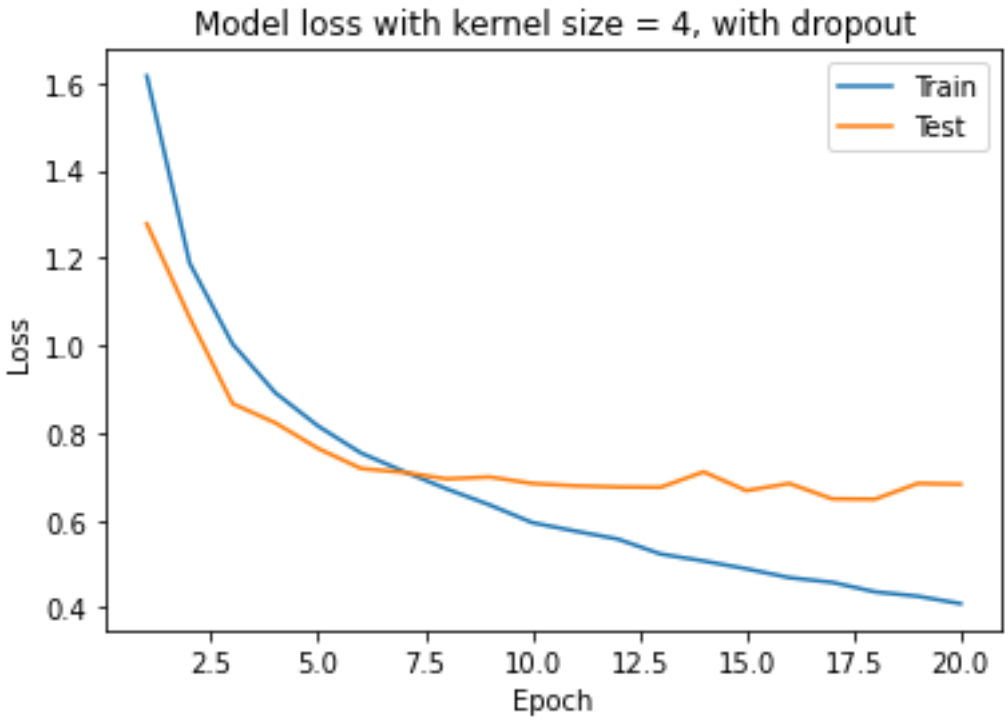

Рисунок 7 – Потери, модель №4

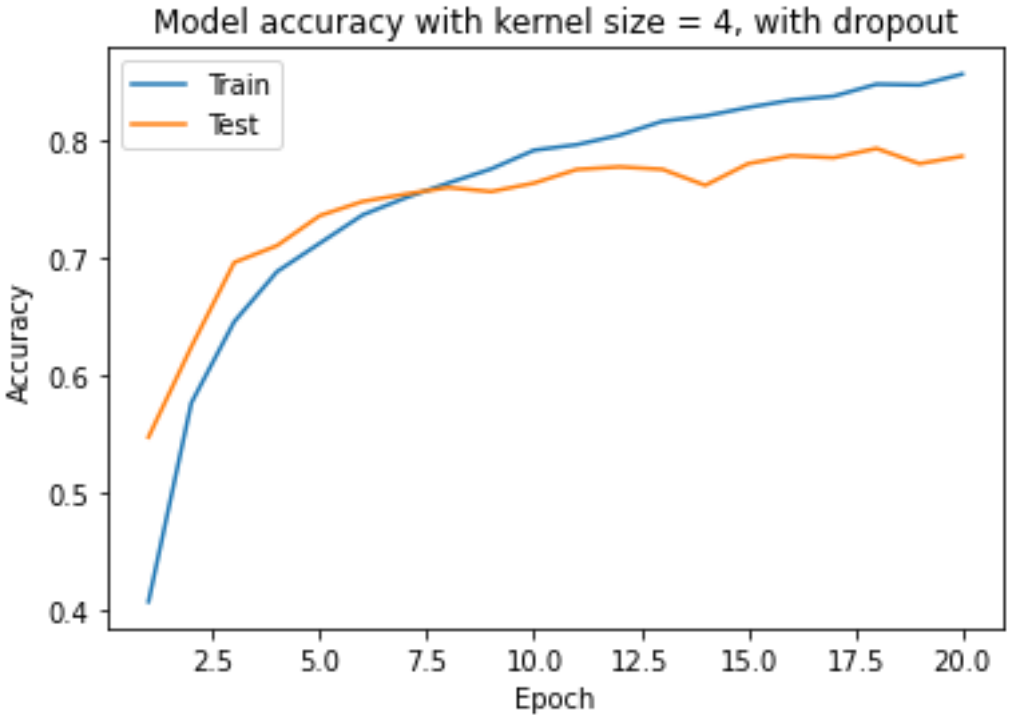

Рисунок 8 – Точность, модель №4

Показатели модели представлены в табл. 5.

Таблица 5

Loss	Accuracy	Val. loss	Val. accuracy	Step time
0.4027	0.8558	0.6824	0.7859	15 ms

Точка переобучения модели – 18-я эпоха. Графики довольно схожи с графиками модели №1 с размером ядра свертки (3,3), показатели точности и потерь также схожи. Стоит заметить, что значительно возросло время на каждый шаг, почти в два раза. Модель №1 достигает высокой точности на более ранних эпохах, а также обучается быстрее (с точки зрения времени, затраченного на 1 шаг).

Модель №5 имеет следующие особенности:

- Слои Dropout включены
- Размер ядра свертки: 5x5

Графики точности и потерь представлены на рис. 9-10.

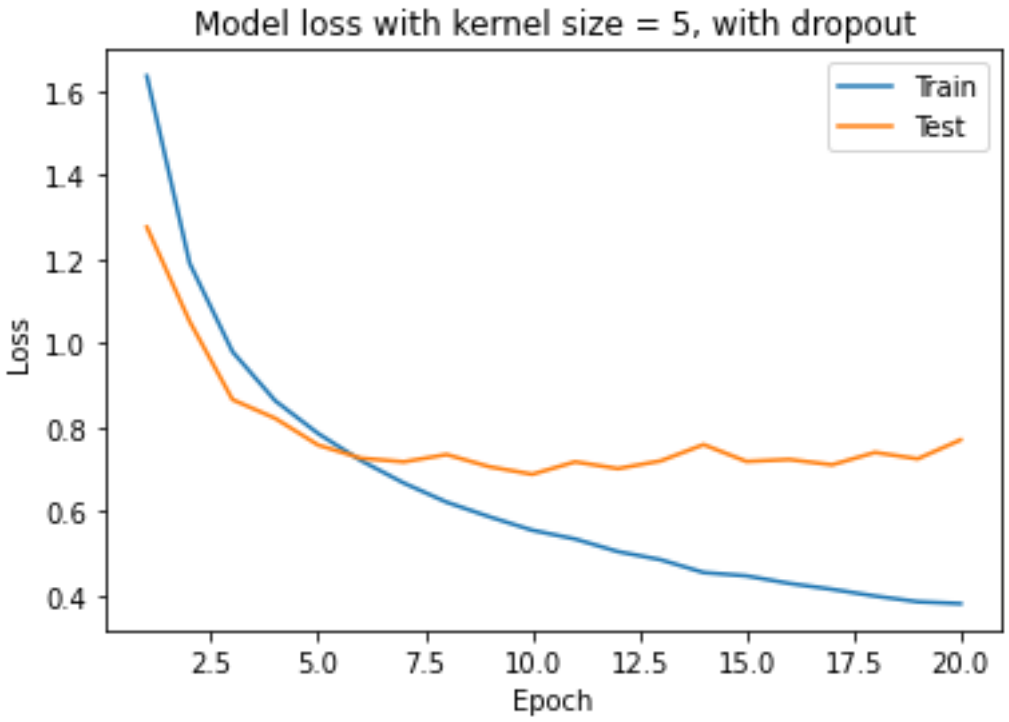

Рисунок 9 – Потери, модель №5

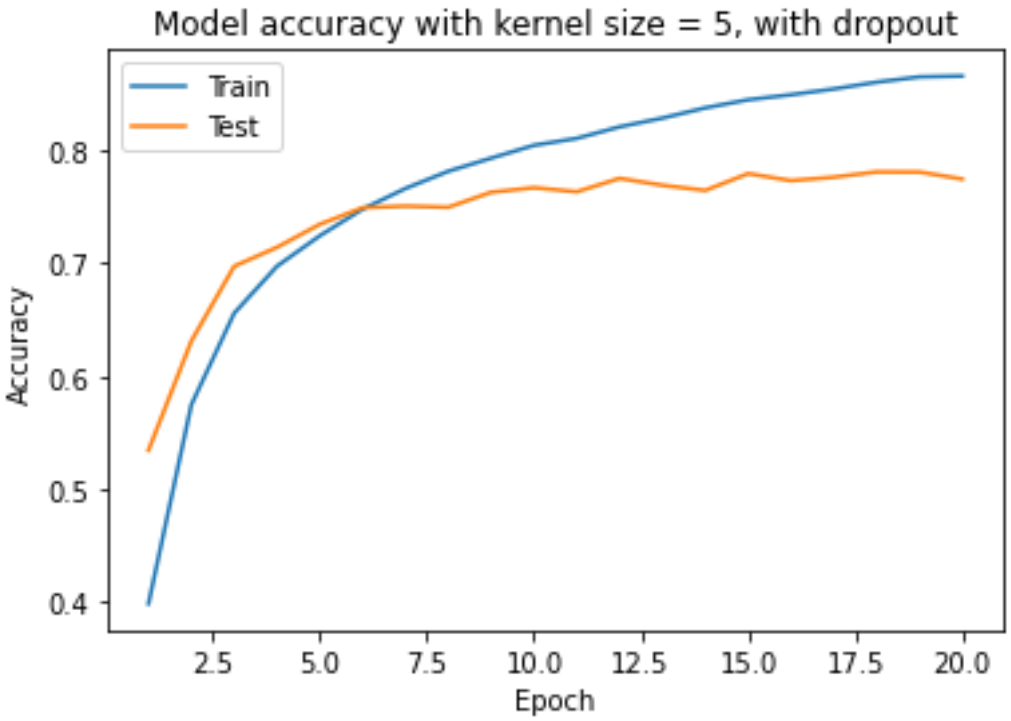

Рисунок 10 – Точность, модель №5

Показатели модели представлены в табл. 6.

Таблица 6

Loss	Accuracy	Val. loss	Val. accuracy	Step time
0.3626	0.8723	0.7697	0.7747	13 ms

Точка переобучения модели – 12 эпоха. При этом, точность сети после обучения на данных для проверки ниже, чем у моделей 1, 3, 4.

Основные выводы из сравнения моделей:

- Модель №2 без слоев Dropout достигла переобучения уже на 4 эпохе и показала большие ошибки и малую точность на данных для проверки
- Модель №1 со слоями Dropout и размером ядра свертки 3x3 показала лучшую точность, однако достигает переобучения на 14 эпохе.
- Модели №4 и №5 с размером ядра свертки 4x4 и 5x5 соответственно требуют большей вычислительной мощности на каждый шаг, при этом не дают преимущества по сравнению с моделью №1.

- Модель №3 с размером ядра свертки 2x2 имеет более низкую точность, чем модель №1, вероятно, потому что размера ядра свертки не хватает для корректного выявления признаков на изображениях.

Выводы.

В ходе выполнения лабораторной работы было изучено решение задачи классификации цветных изображений с помощью сверточной нейронной сети. Было изучено влияние слоев Dropout на показатели сети во время обучения, а также влияние размера ядра свертки. В результате было выявлено, что применение слоев Dropout позволяет решить проблему быстрого переобучения, а размер ядра свертки 3x3 наиболее подходящий под предложенный в задании датасет.

ПРИЛОЖЕНИЕ А

Исходный код программы. Файл lr5.py

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


def load_data():
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()
    num_train, depth, height, width = x_train.shape
    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, x_test, y_test, num_classes, input_shape, num_train


class CNN:
    def __init__(self):
        self.__set_params()
        self.x_train, self.y_train, self.x_test, self.y_test, self.num_classes,
self.input_shape, self.num_train = load_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, dropout=True):
        inp = Input(shape=self.input_shape)
        conv_1 = Convolution2D(self.conv_depth_1, (kernel_size, kernel_size),
                               padding='same', 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 dropout:
            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', activation='relu')(drop_1)
        conv_4 = Convolution2D(self.conv_depth_2, (kernel_size, kernel_size),
                            padding='same', activation='relu')(conv_3)
        pool_2 = MaxPooling2D(pool_size=self.pool_size)(conv_4)
        if dropout:
            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 dropout:
            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, dropout, kernel_size):
        if dropout:
            dropout = ', with dropout'
        else:
            dropout = ', no 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) + dropout)
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'])
        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) +
dropout)
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
```

```
        plt.legend(['Train', 'Test'])
        plt.show()

    def launch(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_data=(self.x_test,
self.y_test))
        self.plot(history, True, 3)
        model = self.build_model(dropout=False)
        history = model.fit(self.x_train, self.y_train,
                            batch_size=self.batch_size, epochs=self.num_epochs,
                            verbose=1,             validation_data=(self.x_test,
self.y_test))
        self.plot(history, False, 3)
        for i in [2, 4, 5]:
            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_data=(self.x_test,
self.y_test))
            self.plot(history, True, i)


net = CNN()
net.launch()
```