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 Процветкина А.В.

Преподаватель Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

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).

Постановка задачи.

- Ознакомиться со сверточными нейронными сетями
- Изучить построение модели в Keras в функциональном виде
- Изучить работу слоя разреживания (Dropout)

Теоретические положения.

Сверточные сети.

Оператор свертки составляет основу сверточного слоя (convolutional layer) в CNN. Слой состоит из определенного количества ядер (с аддитивными составляющими смещения для каждого ядра) и вычисляет свертку выходного изображения предыдущего слоя с помощью каждого из ядер, каждый раз прибавляя составляющую смещения. В конце концов ко всему выходному изображению может быть применена функция активации σ. Обычно входной поток для сверточного слоя состоит из d каналов, например, red/green/blue для входного слоя, и в этом случае ядра тоже расширяют таким образом, чтобы они также состояли из d каналов; получается следующая формула для одного канала выходного изображения сверточного слоя, где K — ядро, a b — составляющая смещения:

$$\mathrm{conv}(I,K)_{x,y} = \sigma(b + \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k=1}^{d} K_{ijk} \times I_{x+i-1,y+j-1,k})$$

Рисунок 1 -- Формула свертки.

На самом деле MLP мог бы в совершенстве справиться с функциями сверточного слоя, но времени на обучение (как и обучающих данных) потребовалось бы намного больше.

Стоит также отметить, что хотя сверточный слой сокращает количество параметров по сравнению с полносвязным слоем, он использует больше гиперпараметров — параметров, выбираемых до начала обучения.

В частности, выбираются следующие гиперпараметры:

Глубина (depth) — сколько ядер и коэффициентов смещения будет задействовано в одном слое;

Высота (height) и *ширина (width)* каждого ядра;

Шаг (stride) — на сколько смещается ядро на каждом шаге при вычислении следующего пикселя результирующего изображения. Обычно его принимают равным 1, и чем больше его значение, тем меньше размер выходного изображения;

Отступ (padding): заметим, что свертка любым ядром размерности более, чем 1x1 уменьшит размер выходного изображения. Так как в общем случае желательно сохранять размер исходного изображения, рисунок дополняется нулями по краям.

Слой разреживания Dropout.

Чтобы помочь сети не утратить способности к обобщению, мы вводим приемы регуляризации: вместо сокращения количества параметров, мы накладываем ограничения на параметры модели во время обучения, не позволяя нейронам изучать шум обучающих данных. Dropout с параметром p за одну итерацию обучения проходит по всем нейронам определенного слоя и с вероятностью p полностью исключает их из сети на время итерации. Это заставит сеть обрабатывать ошибки и не полагаться на существование определенного нейрона (или группы нейронов), а полагаться на “единое

мнение” (consensus) нейронов внутри одного слоя. Это довольно простой метод, который эффективно борется с проблемой переобучения сам, без необходимости вводить другие регуляризаторы. Схема ниже иллюстрирует данный метод.

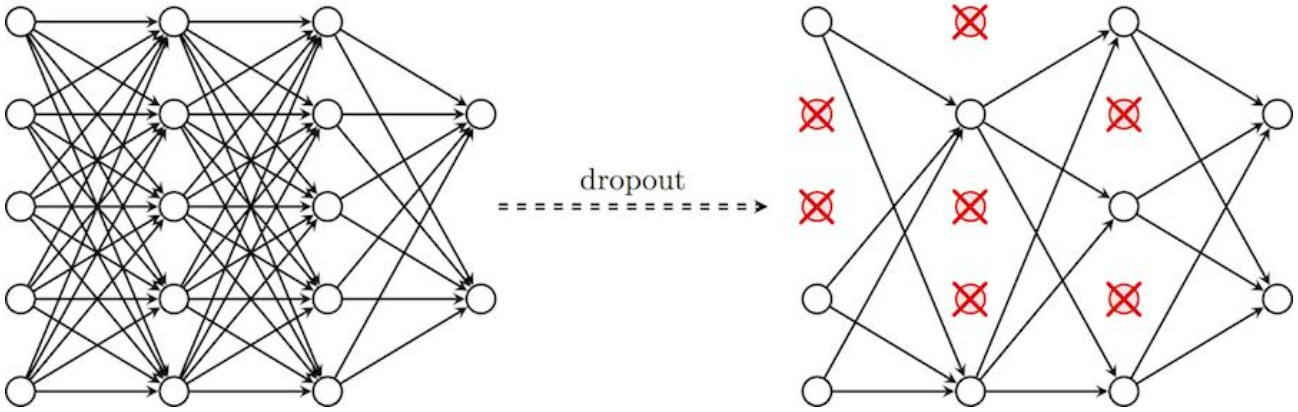

Рисунок 2 -- Схема работы Dropout.

Ход работы.

Построенная сеть состоит из четырех слоев Convolution_2D и слоев MaxPooling2D после второй и четвертой сверток. После первого слоя подвыборки мы удваиваем количество ядер. После этого выходное изображение слоя подвыборки трансформируется в одномерный вектор (слоем Flatten) и проходит два полносвязных слоя (Dense). На всех слоях, кроме выходного полносвязного слоя, используется функция активации ReLU, последний же слой использует softmax.

Количество параметров модели вместе с размером обучающей выборки очень сильно влияют на время обучения сети, поэтому вместо предложенных 200 эпох в работе использовалось только 10 (что безусловно очень мало). Все выводы сделанные в данной работе являются скорее предположением о том, как будет вести себя сеть при увеличении числа эпох.

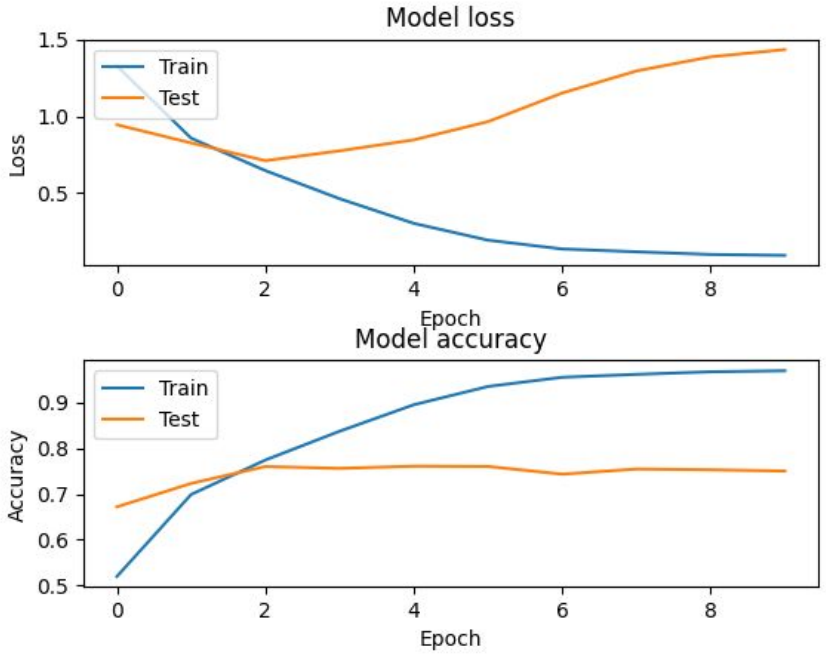

Рисунок 3 -- Процесс обучения сети **без** Dropout и ядром равным 3.

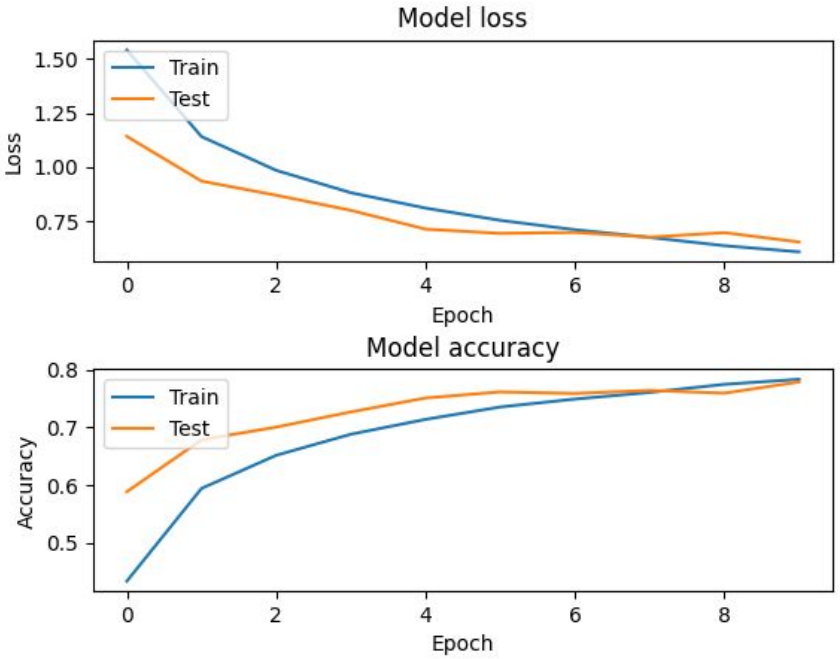

Рисунок 4 -- Процесс обучения сети **с** Dropout и ядром равным 3.

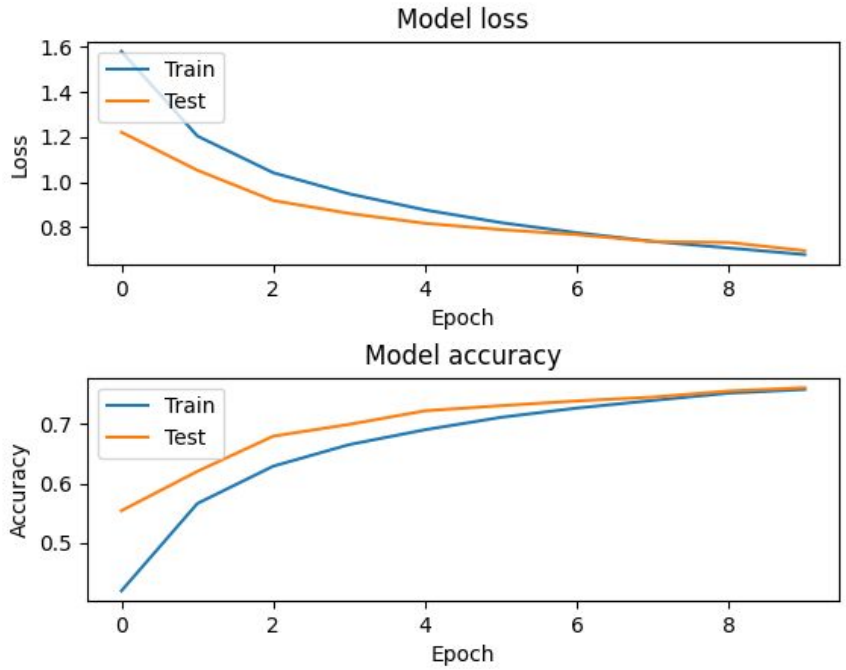

Рисунок 5 -- Процесс обучения сети с Dropout и ядром равным 4.

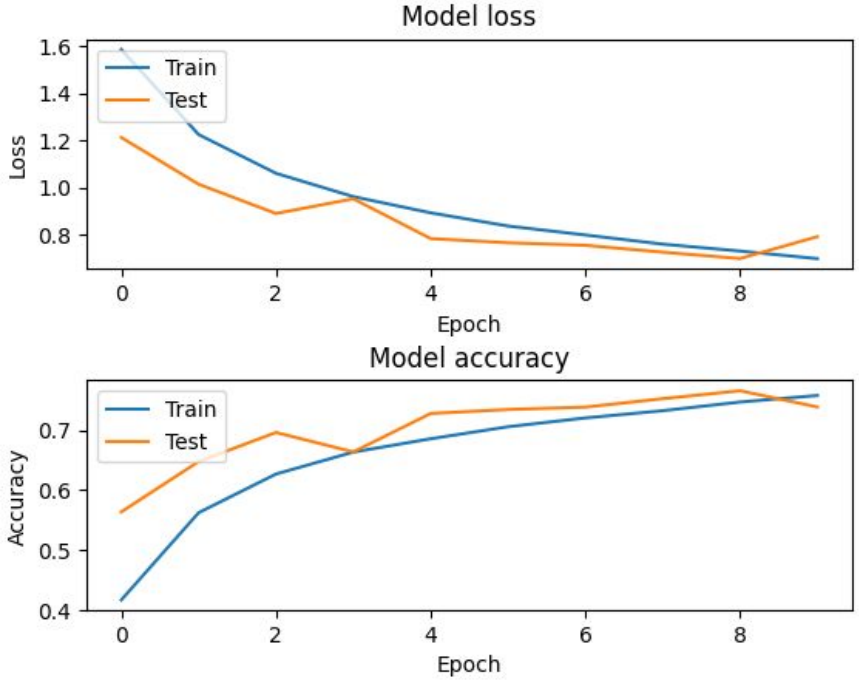

Рисунок 6 -- Процесс обучения сети с Dropout и ядром равным 5.

Таблица 1 -- Сравнение конфигурация модели

Конфигурация сети	Test loss	Test accuracy
kernel_size = 3	450.7131	0.6074
kernel_size = 3, dropout	207.7863	0.5378
kernel_size = 4, dropout	180.6414	0.4803
kernel_size = 5, dropout	242.2317	0.5293

Исходя из полученных результатов можно сказать, что вероятно фильтр размера 3x3 подходит лучше всего.

Замечание: Интересно, что нечетный размер ядра считается наиболее удачным в CNN. Более того, размер ядра выше чем 7 обычно не рассматривают. Общего правила для выбора размера ядра как обычно не существует, что не удивительно.

Dropout, как было указано в теор. положениях, используется для избежания переобучения модели. На рис.3 видно, что буквально со 2-й эпохи происходит переобучение, чего не видно на всех последующих графиках.

Выводы.

В ходе лабораторной работы были изучены основы сверточных нейронных сетей, их параметры, преимущества перед полносвязными сетями. Также был изучен слой Dropout.

Была построена и обучена сверточная нейронная сеть для решения задачи классификации цветных изображений на основе датасета CIFER-10. Работа сети была исследована при разных размерах ядра свертки (3, 4, 5) и при наличии и отсутствии слоя Dropout. Использование Dropout и вправду позволяет избежать

переобучения модели, а размер фильтра равный трем оказался наиболее удачным в данной задаче.

Размер модели и обучающей выборки не позволили в полном объеме выполнить исследование построенной модели, поэтому результаты являются скорее предположениями о том, как будет вести себя сеть если увеличить число эпох.

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 numpy as np
    import matplotlib.pyplot as plt


    batch_size = 32
    num_epochs = 10     # FUCK, I'LL DIE BEFORE 200 EPOCHS END. ARE U
KIDDING?
    kernel_size = 5     # 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

    (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
```

```
    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)
    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)
    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=inp, output=out)           # To define a model,
just specify its input and output layers
    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)
    score = model.evaluate(X_test, Y_test, verbose=1)

    plt.subplot(211)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.subplot(212)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
```

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```