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. 7382 _____ ДроздА.С.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

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)

Требования.

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. С помощью предложенного кода была построена и обучена нейронная сеть.

В целях скорейшего изучения работы сети было решено уменьшить количество эпох до 15.

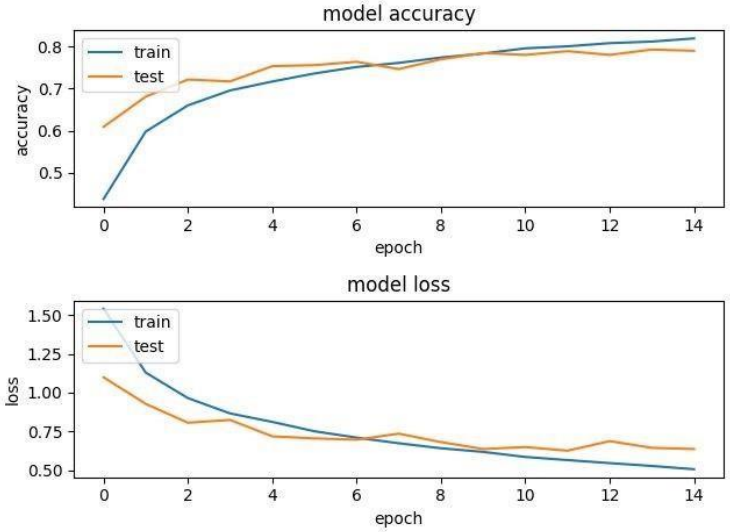

Рисунок 1 - График точности и потерь нейронной сети при 15 эпохах

Уберем слой Dropout и оценим работу сети.

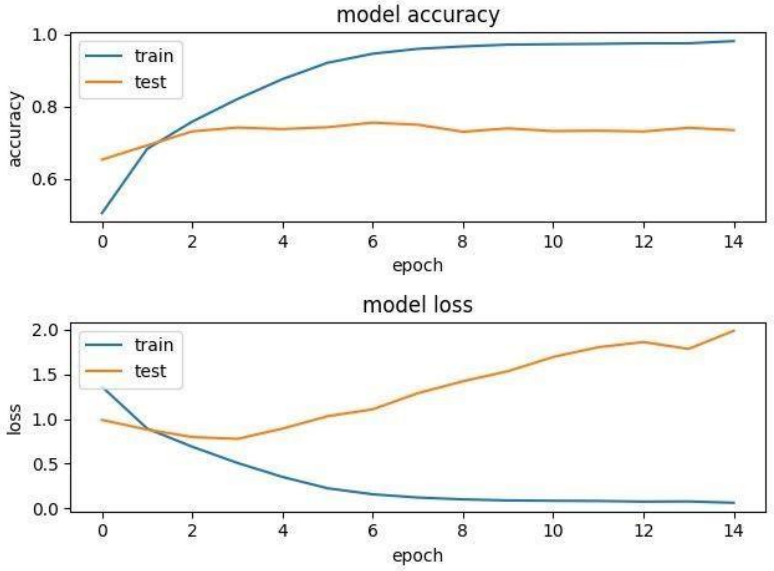

Рисунок 2 - График точности и потерь нейронной сети при 15 эпохах без слоя Dropout

Можно заметить падение точности и увеличение ошибки на тестовых данных, а вот на тренировочных точность стремится к 1, а потери - к 0. Это говорит нам о переобучении модели.Так как у нас задействованы все нейроны, сеть полагается не на “единое мнение”, а на отдельные нейроны. Таким образом, она запомнила ответы к тренировочным данным, но не выявила никакой закономерности среди них и не смогла справиться с тестовыми.

Посмотрим что будет, если изменить размер ядра свертки.

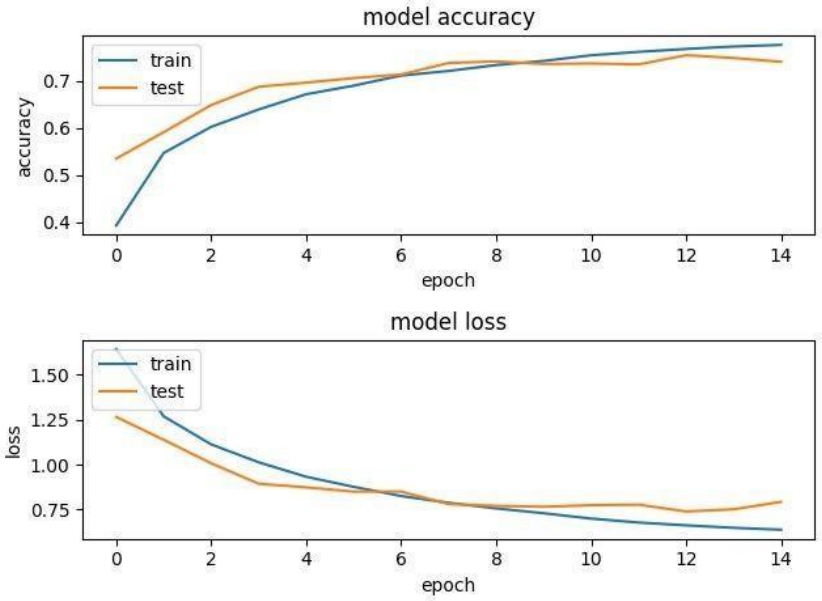

Рисунок 3 - График точности и потерь при размере ядра 5*5

Можно сделать вывод, что с увеличением размера ядра свертки падает точность и возрастают потери. Это связано с тем, что проходясь ядром большего размера, мы уловили меньше отличительных признаков объекта, которые легко выявить ядром меньшего размера.

Выводы.

Таким образом, была изучена задача распознавания объектов на фотографии. Установлено, что слой разрежения позволяет избежать переобучения сети.

ПРИЛОЖЕНИЕ А

```
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 = 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', # using the cross-entropy loss function
          optimizer='adam', # using the Adam optimiser
          metrics=['accuracy']) # reporting the accuracy

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!

plt.subplot(211)
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.subplot(212)
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.show()
```