МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

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

Студент гр. 7381 _____ Тарасенко Е.А.

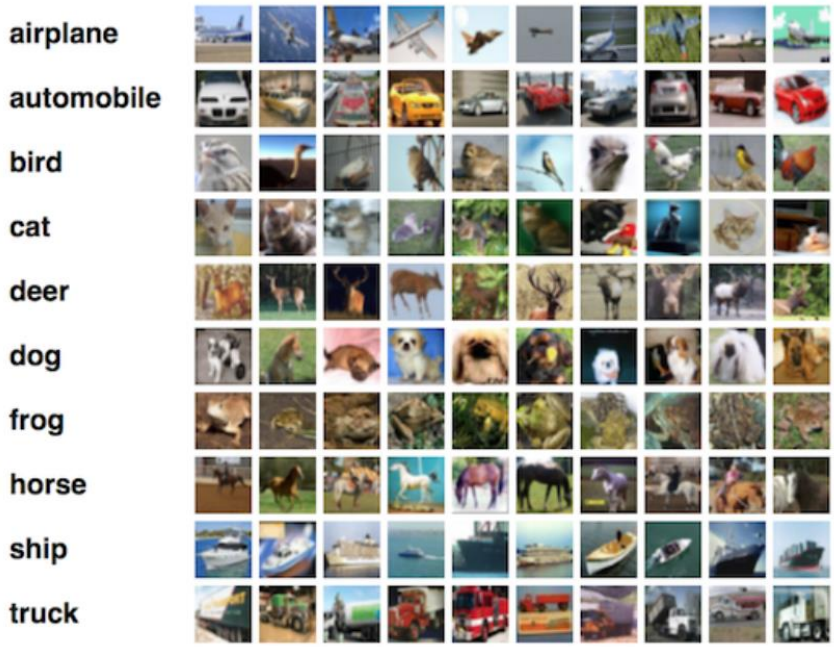
Преподаватель _____ Жукова Н.А.

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).



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 сети без слоя Dropout;
3. Исследовать работу сети при разных размерах ядра свертки.

Ход работы.

Сначала протестируем модель с архитектурой, указанной в методических указаниях к данной лабораторной работе, предварительно снизив количество эпох обучения до 20-ти. Этого (как показали тесты) хватит,

чтобы изучить влияние слоя Dropout на ход обучения сети, что требуется в условии. Также в ходе тестирования не возникло особых проблем с иллюстрацией изменений результатов обучения модели в зависимости от размера ядра свертки. Большее количество эпох не рассматривалось еще и по причине того, что уже на 20-ти обучение могло занять пару часов, а это и так затрудняет исследования. Стартовая архитектура показывала точность около 80%. Графики потерь и точности приведены на рис. 1.

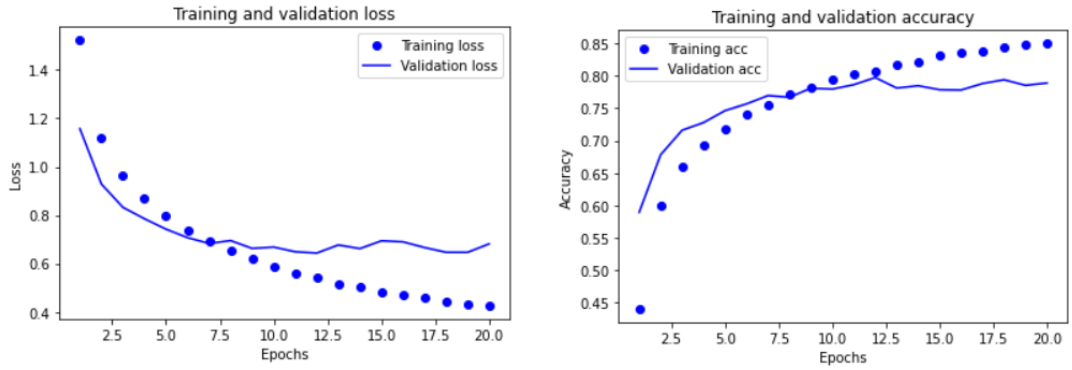


Рис. 1 – Графики ошибки и точности модели стартовой архитектуры

Теперь необходимо исследовать влияние слоев Dropout на ход обучения сети. Для этого они были отключены. Результаты см. на рис. 2.

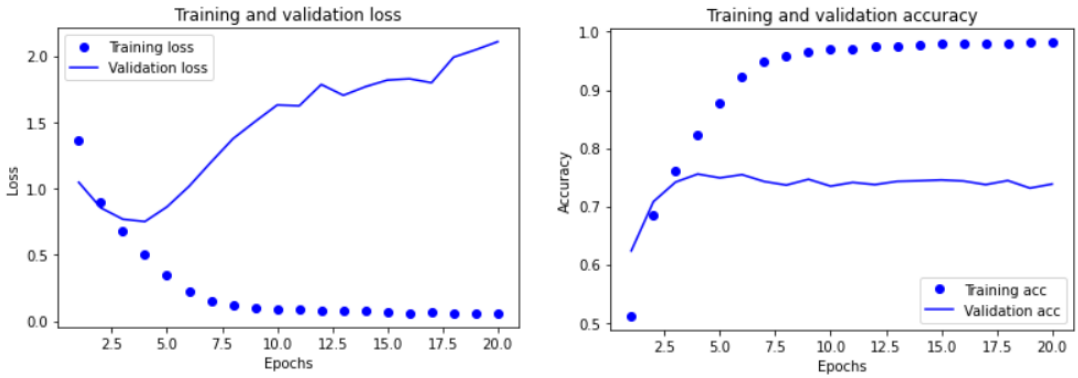


Рис. 2 – Графики ошибки и точности модели без слоев разреживания

Слои разреживания призваны избавиться от проблемы переобучения. В отсутствие Dropout-в потери сети очень быстро начинают расти, а точность на достаточно раннем этапе просто перестает возрастать. Все это неблагоприятно скажется на эксплуатации программы в дальнейшем.

Далее исследуем характер обучения в зависимости от размеров ядра свертки. Модель была обучена на размерах 3x3, 5x5 и 7x7. Результаты приведены на рис. 3.

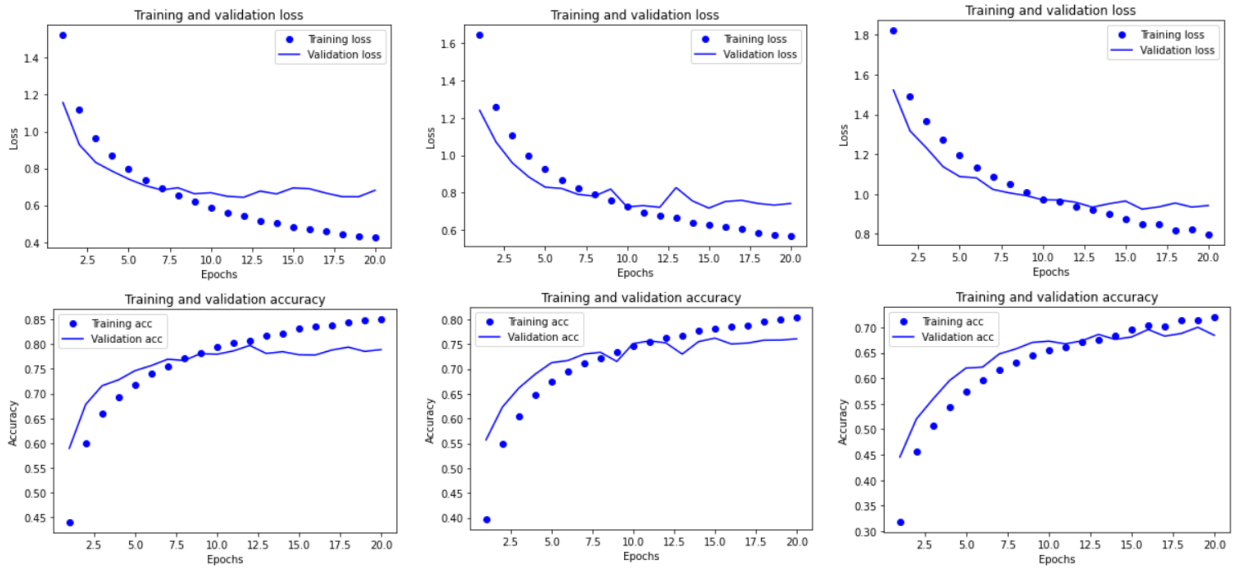


Рис. 1 – Графики ошибки и точности моделей с размерами сверток 3x3, 5x5 и 7x7 соответственно

При начальном ядре точность колебалась при разных тестах около 80%; при размерах 5x5 (уже больших) – около 75%; 7x7 – около 70%. Следовательно, увеличение размеров ядра свертки ухудшает точность модели; также стоит отметить, что оно значительно увеличивает и время обучения сети.

Выводы.

В ходе выполнения данной лабораторной работы была получена некоторая информация о сверточных нейронных сетях, построении модели в Keras в функциональном виде и работе слоев разреживания Dropout. В процессе исследования была построена и обучена модель, способная классифицировать объекты на фотографиях по 10-ти классам: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль и грузовик.

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 matplotlib.pyplot as plt
import numpy as np


def draw_plots(history):
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    epochs = range(1, len(loss) + 1)

    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    plt.clf()
    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()


batch_size = 32 # in each iteration, we consider 32 training examples at once
num_epochs = 20 # we iterate 20 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

(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 i
n 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', ac
tivation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, kernel_size, kernel_size, border_mode='same', ac
tivation='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', ac
tivation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, kernel_size, kernel_size, border_mode='same', ac
tivation='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 valid
ation
draw_plots(history)
model.evaluate(X_test, Y_test, verbose=1) # Evaluate the trained model on the test se
t!
```