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. 7383 _____ Тян Е.

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).

Задание.

1. Ознакомиться со сверточными нейронными сетями
2. Изучить построение модели в Keras в функциональном виде
3. Изучить работу слоя разреживания (Dropout)

Ход работы.

В качестве практической части была построена глубокая сверхточная нейронная сеть, которая была применена к классификации изображений из набора CIFAR-10.

Импорты:

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

Были заданы следующие параметры:

batch_size — количество обучающих образцов, обрабатываемых одновременно за одну итерацию алгоритма градиентного спуска;

num_epochs — количество итераций обучающего алгоритма по всему обучающему множеству;

kernel_size — размер ядра в сверточных слоях;

pool_size — размер подвыборки в слоях подвыборки;

conv_depth — количество ядер в сверточных слоях;

drop_prob (dropout probability) — dropout было применено после каждого слоя подвыборки, а также после полносвязного слоя;

hidden_size — количество нейронов в полносвязном слое MLP.

```
batch_size = 75 # in each iteration, we consider 32 training examples at
once
num_epochs = 20 # 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
```

Загрузка и первичная обработка CIFAR-10 осуществляется ровно так же, как и загрузка и обработка MNIST, где Keras выполняет все автоматически. Единственное отличие состоит в том, что теперь не рассматривается каждый пиксель как независимое входное значение, и поэтому изображение не переносится в одномерное пространство. Интенсивность пикселей была преобразована так, чтобы она попадала в отрезок [0,1] и было использовано прямое кодирование для выходных значений.

Тем не менее, в этот раз этот этап был выполнен для более общего случая, что позволило проще приспосабливаться к новым наборам данных: размер был не жестко задан, а вычислен из размера набора данных, количество классов было определено по количеству уникальных меток в обучающем множестве, а нормализация была выполнена путем деления всех элементов на максимальное значение обучающего множества.

```
(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
```

Была составлена сеть из четырех слоев Convolution_2D и слоев MaxPooling2D после второй и четвертой сверток. После первого слоя подвыборки было удвоено количество ядер. После этого выходное изображение

слоя подвыборки было трансформировано в одномерный вектор (слоем Flatten) и проходит два полносвязных слоя (Dense). На всех слоях, кроме выходного полносвязного слоя, была использована функция активации ReLU, последний же слой использовал softmax.

Для регуляризации модели после каждого слоя подвыборки и первого полносвязного слоя был применен слой Dropout.

```
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), padding
= 'same', activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, (kernel_size, kernel_size),
padding='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),
padding='same', activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, (kernel_size, kernel_size),
padding='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(inp, 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
H = 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
result = model.evaluate(X_test, Y_test, verbose=1) # Evaluate the
trained model on the test set!
```

Были построены графики ошибки и точности, изображенные на рис. 1 и 2. Точность составила: 0.6279.

Была исследована работа сети без слоя Dropout, представленная на рис. 3 и 4. Точность составила: 0.6195.

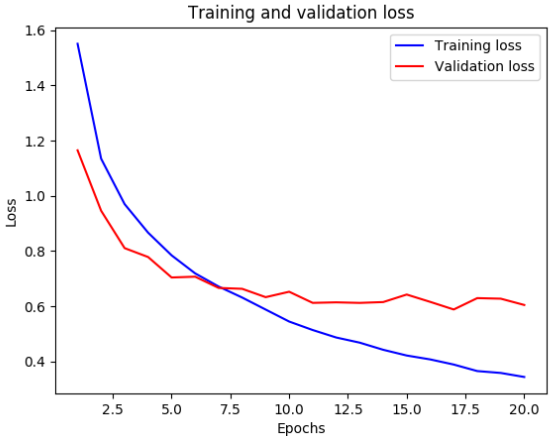

Рисунок 1 — График ошибки kernel_size=3

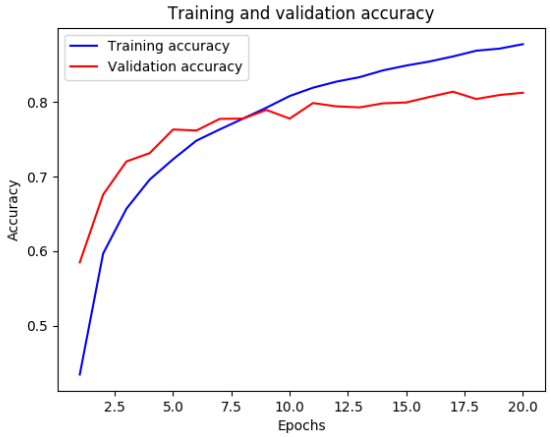

Рисунок 2 — График точности kernel_size=3

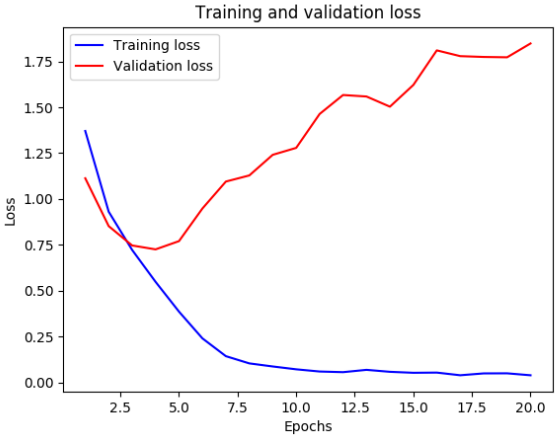

Рисунок 3 — График ошибки без слоя Dropout

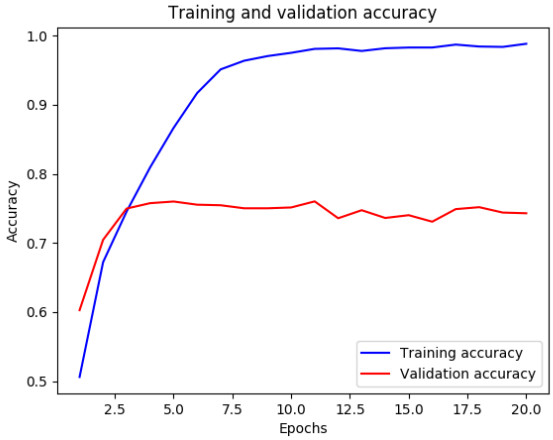

Рисунок 4 — График точности без слоя Dropout

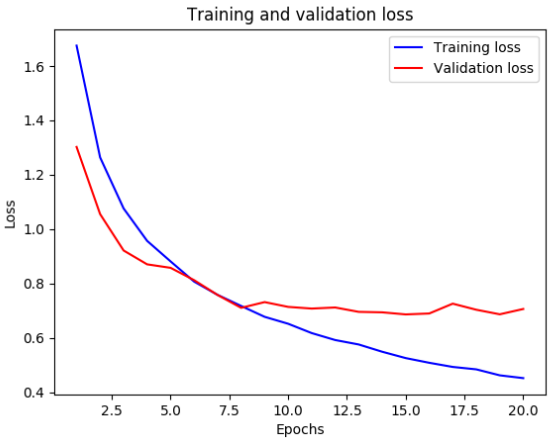

Рисунок 5 — График ошибки kernel_size=5

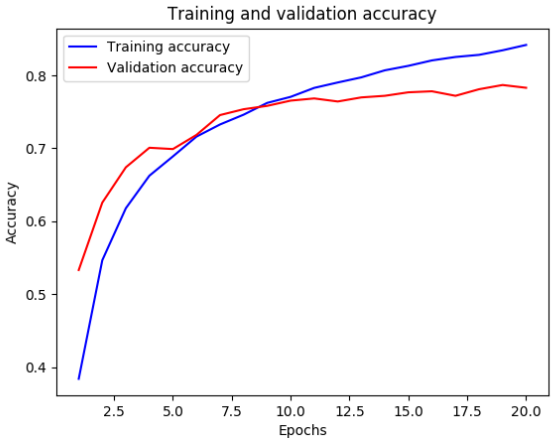

Рисунок 6 — График точности kernel_size=5

Была исследована работа сети при разных размерах ядер свертки: 3x3, 5x5 и 7x7. Результаты представлены на рис. 5, 6, 7 и 8. В случае 5x5 точность составила 0.6065. В случае 7x7 – 0.5059.

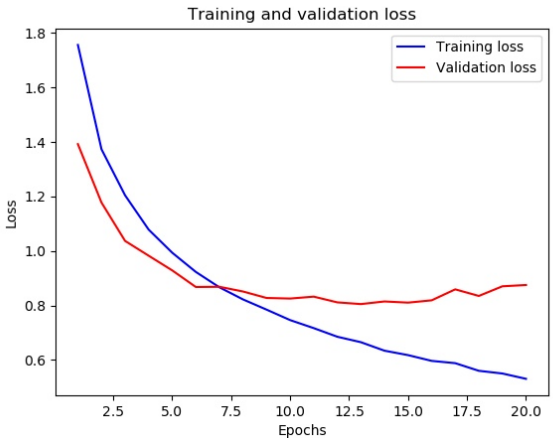

Рисунок 7 — График ошибки kernel_size=7

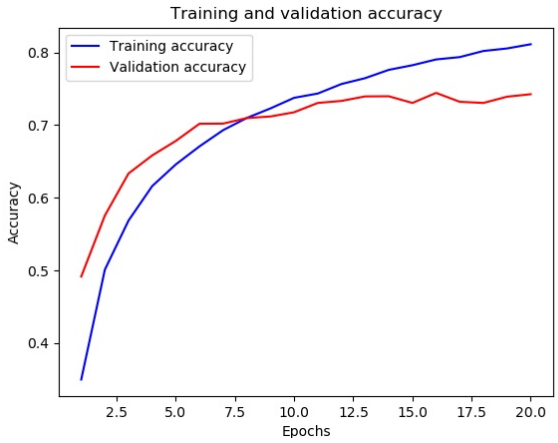

Рисунок 8 — График точности kernel_size=7

Выводы.

В ходе выполнения лабораторной работы была построена и обучена сверхточная нейронная сеть. Была исследована работа сети без слоя Dropout. Можно заметить, что переобучение началось на 3 эпохе. Это подтверждает тот факт, что Dropout – метод-решение проблемы переобучения. Исследована работа сети при разных размерах ядра свертки: 3x3, 5x5, 7x7. С увеличением размера ядра свертки, ухудшалась точность. Такое явление можно наблюдать, т.к. при увеличении размера ядра свертки, растет число связей, сеть начинает переобучение раньше.