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. 7383 _____ Александров Р.А.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель работы.

Распознавание объектов на фотографиях (Object Recognition in Photographs).

Постановка задачи.

1. Ознакомиться со сверточными нейронными сетями
2. Изучить построение модели в Keras в функциональном виде
3. 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

Выполнение работы.

В ходе работы была создана и обучена модель нейронной сети, весь код представлен в приложении А.

Из рис. 1-2 видим, представлены графики ошибок и точности для сверточной сети, с размером ядра 3x3. Видим, что точность составляет 79%.

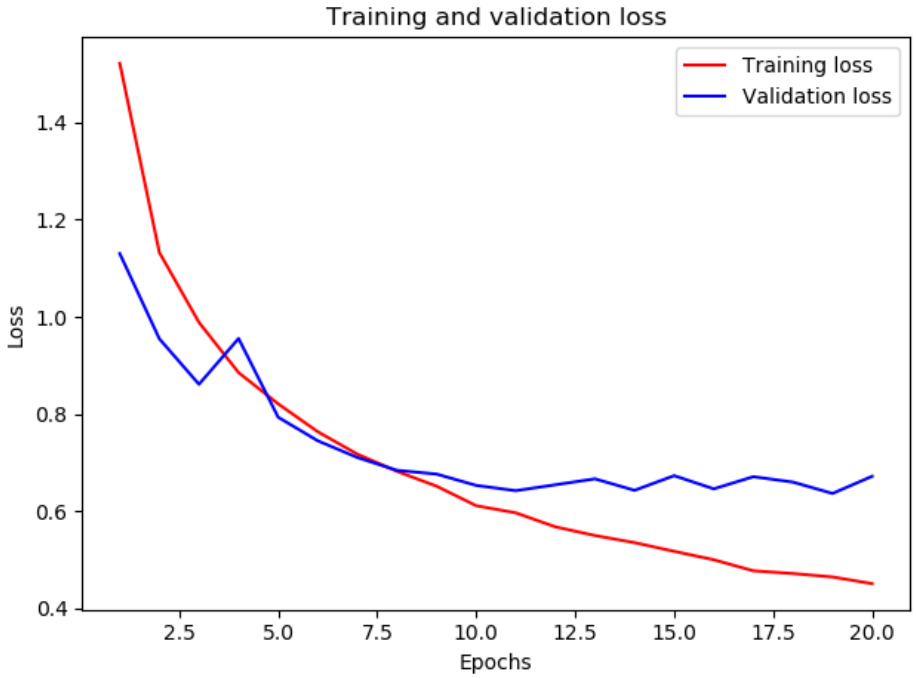

Рисунок 1 – Ошибки для CNN с ядром 3x3

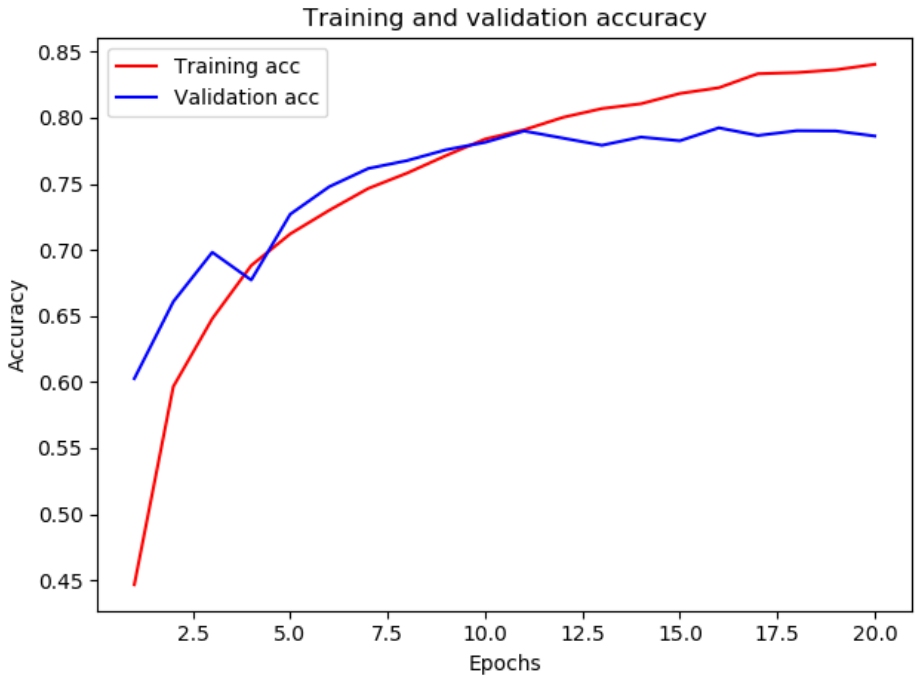

Рисунок 2 – Точность для CNN с ядром 3x3

Для этой же сети уберем слой dropout, результаты представлены на рис. 3-4. Видим, что точность уменьшилась до 71%, а также увеличились ошибки.

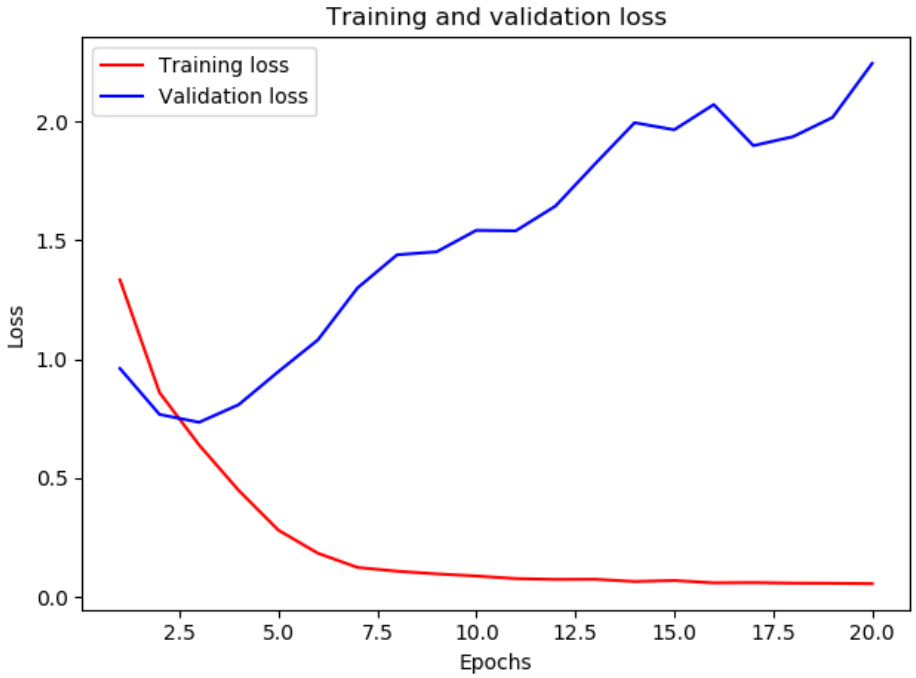

Рисунок 3 – Ошибки для CNN с ядром 3x3 без слоя dropout

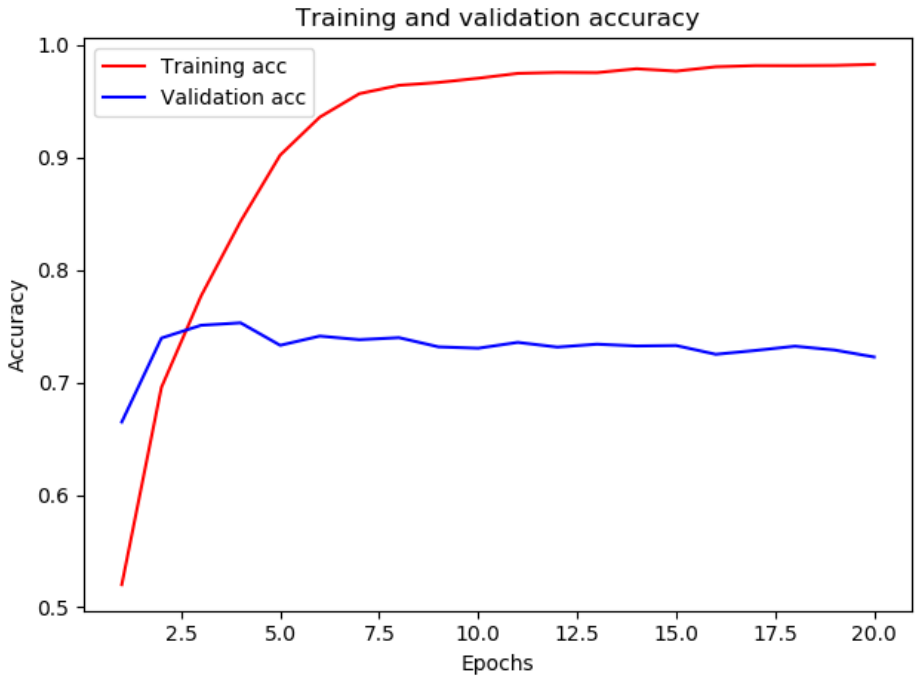

Рисунок 4 – Точность для CNN с ядром 3x3 без слоя dropout

Теперь изменим размер ядра свертки на 6x6, результаты представлены на рис. 5-6. Видим, что точность по сравнению с ядром 3x3 уменьшилась и составляет 75%, и снова увеличились ошибки.

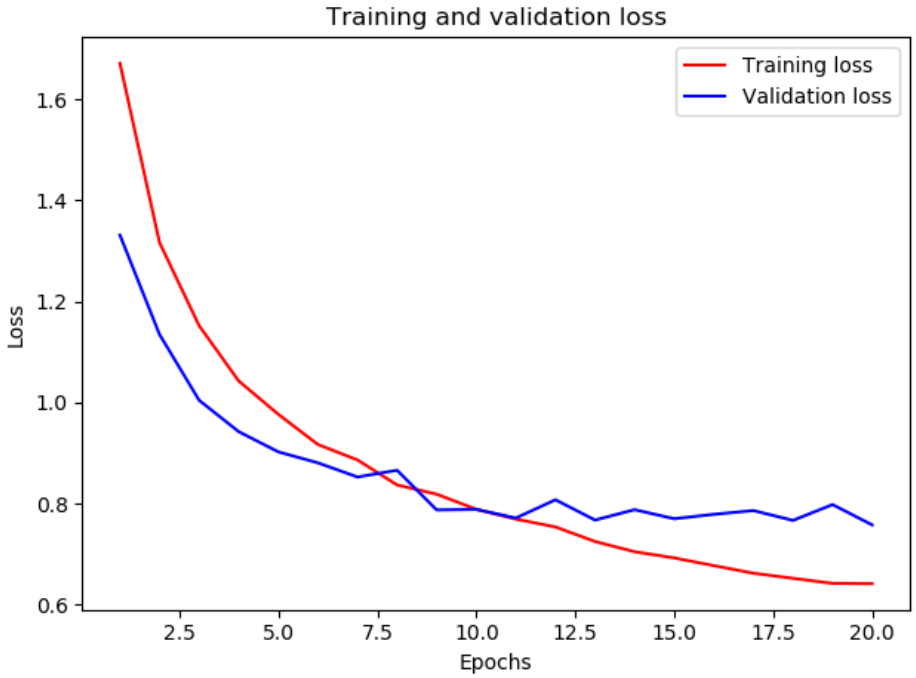

Рисунок 5 – Ошибки для CNN с ядром 6x6

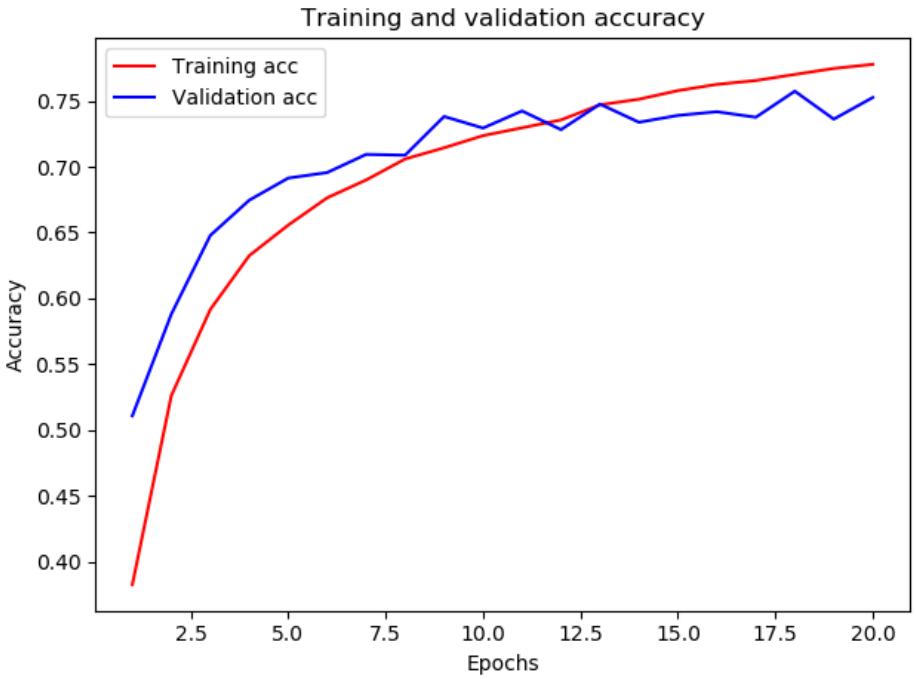

Рисунок 6 – Точность для CNN с ядром 6x6

Выводы.

В ходе работы были изучены принципы сверточных нейронных сетей, слоя разреживания. Была построена и обучена нейронная сеть, которая исследовалась на различных размерах ядра свертки и с влиянием слоя разреживания.

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 # in each iteration, we consider 32 training examples
at once
num_epochs = 20 # we iterate 200 times over the entire training set
kernel_size = 6 # 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!

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
epochs = range(1, num_epochs + 1)
```

```
plt.plot(epochs, loss_values, 'r', label='Training loss')
plt.plot(epochs, val_loss_values, '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, 'r', 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()
```