МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе № 5

по дисциплине «Искусственные нейронные сети»

Тема: Распознавание объектов на фотографиях

Студент гр. 8383 _____ Костарев К.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

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)

Выполнение работы.

Для выполнения лабораторной работы был написан код программы на языке Python. Для начала в переменных были заданы гиперпараметры сети, представленные в табл. 1.

Таблица 1 – Гиперпараметры CNN

Название	Начальное значение	Определение
batch_size	32	количество обучающих образцов, обрабатываемых одновременно за одну итерацию алгоритма градиентного спуска
num_epoch	10	количество итераций обучающего алгоритма по всему обучающему множеству
kernel_size	3	размер ядра в сверточных слоях
pool_size	2	размер подвыборки в слоях подвыборки
conv_depth_1, conv_depth_2	32, 64	количество ядер в сверточных слоях
drop_prob_1, drop_prob_2	0.25, 0.5	мы будем применять dropout после каждого слоя подвыборки, а также после полносвязного слоя
hidden_size	512	количество нейронов в полносвязном слое MLP

Было выбрано 10 итераций обучения вместо предлагаемых 200 из-за ограничений в производительности ноутбука, на котором данная работа выполнялась.

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 мы не рассматриваем каждый пиксель как независимое входное значение, и поэтому мы не переносим изображение в одномерное пространство. Мы снова преобразуем интенсивность пикселей так, чтобы она попадала в отрезок [0,1] и используем прямое кодирование для выходных значений.

```
(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
```

Далее была настроена модель сети: из четырех слоев Convolution_2D и слоев MaxPooling2D после второй и четвертой сверток. После первого слоя подвыборки мы удваиваем количество ядер (вместе с описанным выше принципом принесения высоты и ширины в жертву глубине). 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. Для регуляризации нашей модели после каждого слоя подвыборки и первого полносвязного слоя применяется слой Dropout.

Результаты работы сети:

```
Epoch 10/10
1407/1407 - 142s - loss: 0.5577 - accuracy: 0.8037 - val_loss: 0.7154 - val_accuracy: 0.7688
313/313 - 8s - loss: 135.6227 - accuracy: 0.5634
```

Мы можем видеть, что точность сети на тренировочных данных составила 80%, на проверочных 77% и на тестовых 56%, время обучения в течение одной эпохи примерно 142 – 144 секунды.

Удалим из модели сети все слои Dropout. Результаты работы сети:

```
Epoch 10/10
1407/1407 - 173s - loss: 0.0896 - accuracy: 0.9696 - val_loss: 1.5969 - val_accuracy: 0.7452
313/313 - 7s - loss: 473.4882 - accuracy: 0.5365
```

Точность сети на тренировочных данных составила 97%, на проверочных 75% а на тестовых 54%. Это говорит о том, что с удалением слоев Dropout в какой-то момент началось переобучение сети, а они как раз позволяют избегать этого. Время обучения в пределах одной эпохи варьировалось от 122 до 173 секунд, что можно связать с тротлингом процессора и следовательно меньшей производительностью в некоторые промежутки времени.

Далее все слои были добавлены обратно, но размер ядра в сверточных слоях был уменьшен до 2x2. Результат работы сети:

```
Epoch 10/10
1407/1407 - 121s - loss: 0.7182 - accuracy: 0.7478 - val_loss: 0.7493 - val_accuracy: 0.7508
313/313 - 6s - loss: 282.4499 - accuracy: 0.4075
```

Сеть стала обучаться быстрее, но при этом стала гораздо менее точной. Точность сети на тренировочных данных составила 75%, на проверочных 75%, а на тестовых 41%.

Тогда увеличим размер ядра до 4x4. Результат составил:

```
Epoch 10/10
1407/1407 - 113s - loss: 0.6906 - accuracy: 0.7574 - val_loss: 0.7016 - val_accuracy: 0.7724
313/313 - 7s - loss: 168.3784 - accuracy: 0.5388
```

Сеть стала точнее, но далеко уступает точности сети при размере ядра свертки 3x3 (точность на тренировочных – 76%, на проверочных – 77%, на тестовых 54%).

Выводы.

В данной лабораторной работе была исследована зависимость точности сверточной НС при отсутствии/присутствии в модели слоя Dropout и различных размерах ядра свертки. Самой оптимальной моделью является с наличием Dropout (иначе происходит переобучение) и размером ядра 3x3.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"

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

batch_size = 32  # in each iteration, we consider 32 training examples at once
num_epochs = 10  # we iterate 10 times over the entire training set
kernel_size = 4  # 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), 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')(pool_1)
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()(pool_2)
flat = Flatten()(drop_2)
hidden = Dense(hidden_size, activation='relu')(flat)
drop_3 = Dropout(drop_prob_2)(hidden)
#out = Dense(num_classes, activation='softmax')(hidden)
out = Dense(num_classes, activation='softmax')(drop_3)

model = Model(inputs=inp, outputs=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

model.fit(X_train, Y_train,  # Train the model using the training set...
          batch_size=batch_size, epochs=num_epochs,
```

```
        verbose=2, validation_split=0.1)  # ...holding out 10% of the data for validation
evaluate = model.evaluate(X_test, Y_test, verbose=2)  # Evaluate the trained model on the test set!
```