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. 8382 _____ Нечепуренко Н.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цели работы.

Решить задачу распознавания 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).

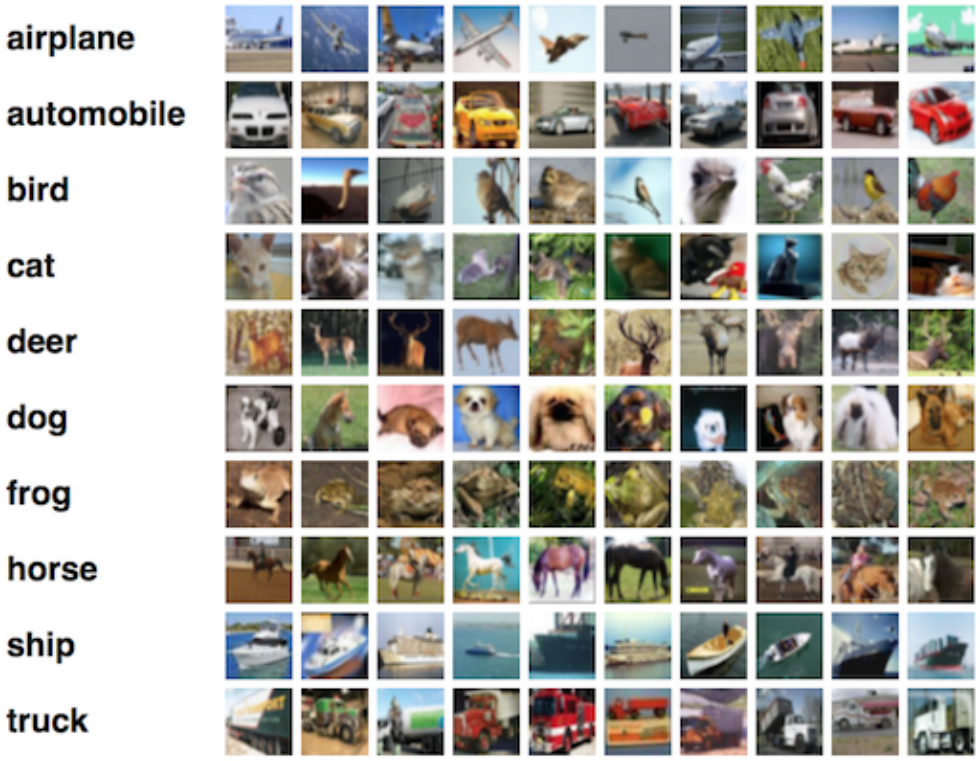

Рисунок 1 – Примеры изображений из датасета CIFAR10

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

Выполнение работы.

Импортируем необходимые для построения модели зависимости, а также набор данных CIFAR10.

```
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
```

Зададим константы конфигурации модели, среди которых как параметры слоев, так и параметры обучения.

```
batch_size = 32 # in each iteration, we consider 32 training
   examples at once
num_epochs = 200 # 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
```

Переведем значения каналов изображений (цветовых каналов) в отре-

зок [0,1], а также применим one-hot кодирование для вектора категорий.

```
(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
```

Построим модель согласно методическим указаниям.

```
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), activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, (kernel_size,
   kernel_size), 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), activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, (kernel_size,
   kernel_size), 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(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
```

Обучим модель и проверим ее на тестовых данных.

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

Результаты обучения модели следующие

```
    Epoch 200/200
1407/1407 [==============================] - 13s 10ms/step - loss
   : 0.3318 - accuracy: 0.8886 - val_loss: 0.7659 - val_accuracy:
    0.7958
313/313 [==============================] - 2s 5ms/step - loss:
   265.9335 - accuracy: 0.4758
[265.9334716796875, 0.47580000758171087]
```

Несмотря на удовлетворительные результаты в процессе обучения (точность 90% на тренировочных данных и 80 % на валидационных), точность на тестовых данных получилась порядка 50%. Модель обучалась в течение 200 эпох, даже с использованием графического ускорителя в сервисе Google Colab обучение заняло более 40 минут.

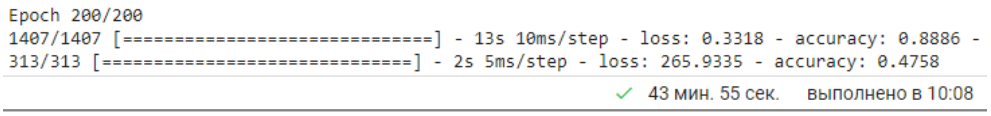

Рисунок 2 – Время обучения модели

При этом уже на 75 эпохе были получены следующие значения

```
    Epoch 75/200
1407/1407 [==============================] - 13s 9ms/step - loss:
    0.3947 - accuracy: 0.8648 - val_loss: 0.7165 - val_accuracy:
   0.7876
```

т.е. за остальные 125 эпох результаты улучшились всего на 2%, поэтому попробуем провести обучение в течение 75 эпох и сравним результаты.

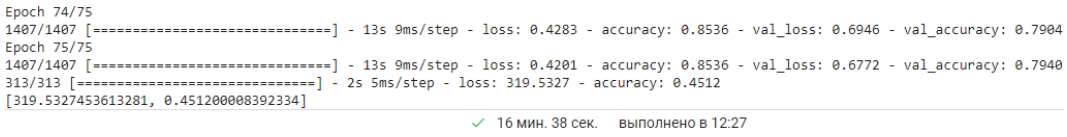

Рисунок 3 – Результат обучения модели в течение 75 эпох

потеряли 2% точности, зато модель стала учиться гораздо быстрее.

Исследуем работу сети без использования Dropout.

```
    Epoch 73/75
1407/1407 [==============================] - 13s 9ms/step - loss:
    0.1449 - accuracy: 0.9552 - val_loss: 1.5911 - val_accuracy:
   0.7538
Epoch 74/75
1407/1407 [==============================] - 13s 9ms/step - loss:
    0.1324 - accuracy: 0.9584 - val_loss: 1.6443 - val_accuracy:
   0.7468
Epoch 75/75
1407/1407 [==============================] - 13s 9ms/step - loss:
    0.1421 - accuracy: 0.9555 - val_loss: 1.4982 - val_accuracy:
   0.7520
313/313 [==============================] - 1s 4ms/step - loss:
   696.9914 - accuracy: 0.5380
[696.9913940429688, 0.5379999876022339]
```

Модель явно переобучилась, но на удивление показала лучшие результаты на тестовых данных. Несмотря на это, результаты остаются неудовлетворительными.

Вернем слои Dropout в модель. Изменим размер ядра свертки с 3 до 5.

```
    Epoch 73/75
1407/1407 [==============================] - 12s 9ms/step - loss:
    0.6919 - accuracy: 0.7694 - val_loss: 0.9628 - val_accuracy:
   0.6938
Epoch 74/75
1407/1407 [==============================] - 12s 9ms/step - loss:
    0.6847 - accuracy: 0.7680 - val_loss: 0.8491 - val_accuracy:
   0.7262
Epoch 75/75
1407/1407 [==============================] - 12s 9ms/step - loss:
```

```
    0.6774 - accuracy: 0.7667 - val_loss: 0.8908 - val_accuracy:
   0.7200
313/313 [==============================] - 1s 5ms/step - loss:
   196.6075 - accuracy: 0.5129
[196.60745239257812, 0.5128999948501587]
```

Результаты на тренировочном наборе оказались хуже, зато на тестовом множестве модель показала более удачные результаты.

Установим размер ядра свертки равным 2.

```
    Epoch 74/75
1407/1407 [==============================] - 12s 9ms/step - loss:
    0.2906 - accuracy: 0.8977 - val_loss: 0.7265 - val_accuracy:
   0.7878
Epoch 75/75
1407/1407 [==============================] - 12s 9ms/step - loss:
    0.3068 - accuracy: 0.8961 - val_loss: 0.7213 - val_accuracy:
   0.7938
313/313 [==============================] - 1s 4ms/step - loss:
   375.5798 - accuracy: 0.4165
[375.57977294921875, 0.4165000021457672]
```

Результаты на тестовом множестве оказались гораздо хуже.

Сравнивая результаты модели с размерами ядер 2,3,5, приходим к выводу, что ядро 3x3 наиболее предпочтительно для заданной конфигурации модели.

Выводы.

В результате выполнения лабораторной работы был реализован классификатор изображений по 10 категориям на основе набора данных CIFAR10. Был изучен принцип обработки изображений с помощью слоев свертки и пуллинга. Было исследовано использование слоя Dropout для уменьшения пере-

обучения модели. К сожалению, полученные результаты вряд ли можно применить в прикладных программах, так как для увеличения точности стоит больше экспериментировать с архитектурой и параметрами модели, что возможно при наличии соответствующего оборудования. Даже с использованием облачных средств и графического ускорителя, модели обучались довольно долго.

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

batch_size = 32 # in each iteration, we consider 32 training
   examples at once
num_epochs = 75 # we iterate 75 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 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),
   activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, (kernel_size, kernel_size),
   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),
   activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, (kernel_size, kernel_size),
   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(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=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!
```