УНИВЕРСИТЕТ ИТМО

# Практика. Сверточные нейронные сети

Олег А. Евстафьев[1] Михаил А. Каканов[1]
[1] Факультет систем управления и робототехники, Университет ИТМО
{oaevstafev, makakanov}@itmo.ru

Декабрь 2021
Курс «Прикладной искусственный интеллект»

# Слой свёртки на Python

УНИВЕРСИТЕТ ИТМО

```python
# импортировать необходимые пакеты
from skimage.exposure import rescale_intensity
import numpy as np
import argparse
import cv2

def convolve(image, K):
    # захватить пространственные размеры изображения и ядра
    (iH, iW) = image.shape[:2]
    (kH, kW) = K.shape[:2]

    # выделите память для выходного изображения,
    #позаботившись о "pad" -заполнении или отступе
    # границы входного изображения, поэтому пространственный размер (т. е,
    # ширина и высота) не уменьшаются
    pad = (kW - 1) // 2
    image = cv2.copyMakeBorder(image, pad, pad, pad, pad,
        cv2.BORDER_REPLICATE)
    output = np.zeros((iH, iW), dtype="float")

    # цикл по входному изображению, "скользя" ядром по поверхности
    # каждая (x, y)-координата слева направо и сверху вниз
    for y in np.arange(pad, iH + pad):
        for x in np.arange(pad, iW + pad):
            # извлечение ROI изображения путем извлечения
            # *центр* области текущих (x, y)-координат
            roi = image[y - pad:y + pad + 1, x - pad:x + pad + 1]

            # выполнить фактическую свертку, взяв
            # поэлементное умножение между ROI и
            # ядро, затем суммирование матрицы
            k = (roi * K).sum()

            # сохранить свернутое значение в выходном (x, y)-.
            # координата выходного изображения
            output[y - pad, x - pad] = k

    # изменить масштаб выходного изображения в диапазоне [0, 255]
    output = rescale_intensity(output, in_range=(0, 255))
    output = (output * 255).astype("uint8")

    # возвращает выходное изображение
    return output
```

ITMOre than a UNIVERSITY

# Слой свёртки на Python

УНИВЕРСИТЕТ ИТМО

```python
# построить разбор аргументов и разобрать аргументы(парсер-флаги)
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to the input image")
args = vars(ap.parse_args())

# построить средние ядра размытия, используемые для сглаживания изображения
smallBlur = np.ones((7, 7), dtype="float") * (1.0 / (7 * 7))
largeBlur = np.ones((21, 21), dtype="float") * (1.0 / (21 * 21))

# построить фильтр повышения резкости
sharpen = np.array((
    [0, -1, 0],
    [-1, 5, -1],
    [0, -1, 0]), dtype="int")

# построить ядро Лапласиана, используемое для обнаружения краеподобного
# области изображений
laplacian = np.array((
    [0, 1, 0],
    [1, -4, 1],
    [0, 1, 0]), dtype="int")

# построить ядро Собеля по оси x
sobelX = np.array((
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]), dtype="int")
# построить ядро Собеля по оси y

sobelY = np.array((
    [-1, -2, -1],
    [0, 0, 0],
    [1, 2, 1]), dtype="int")

# построить ядро для вытиснения (emboss kernel)
emboss = np.array((
    [-2, -1, 0],
    [-1, 1, 1],
    [0, 1, 2]), dtype="int")
```

ITMOre than a UNIVERSITY

# Слой свёртки на Python

УНИВЕРСИТЕТ ИТМО

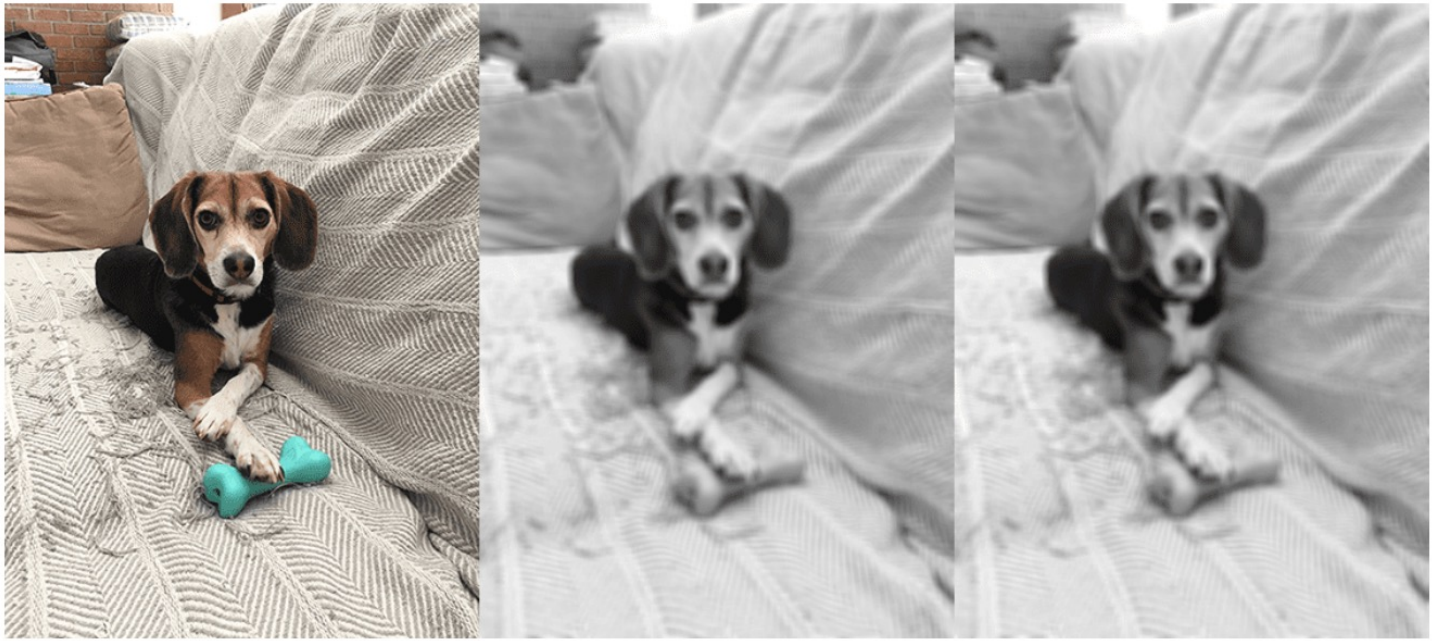
```
# создать банк ядер, список ядер, которые мы собираемся применить
# используя как нашу собственную функцию "convolve", так и OpenCV "filter2D".
# функция
kernelBank = (
    ("small_blur", smallBlur),
    ("large_blur", largeBlur),
    ("sharpen", sharpen),
    ("laplacian", laplacian),
    ("sobel_x", sobelX),
    ("sobel_y", sobelY),
    ("emboss", emboss))

# загрузить входное изображение и преобразовать его в градации серого цвета
image = cv2.imread(args["image"])
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# перебрать ядра в цикле
for (kernelName, K) in kernelBank:
    # применяем ядро к полутоновому изображению, используя как наши собственные
    # Функция 'convolve' и функция OpenCV 'filter2D'
    print("[INFO] applying {} kernel".format(kernelName))
    convolveOutput = convolve(gray, K)
    opencvOutput = cv2.filter2D(gray, -1, K)
    # показать выходные изображения
    cv2.imshow("Original", gray)
    cv2.imshow("{} - convolve".format(kernelName), convolveOutput)
    cv2.imshow("{} - opencv".format(kernelName), opencvOutput)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

ITMOre than a UNIVERSITY

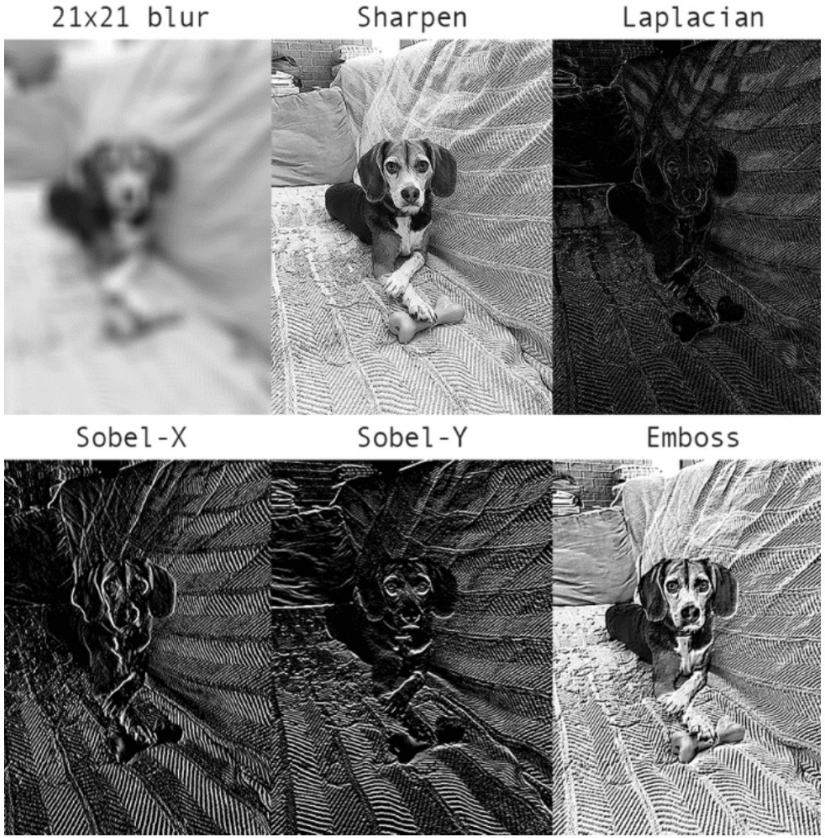
# Результат



Input Image | convolve | cv2.filter2D



# Результат



# MNIST-Recognition-CNN

УНИВЕРСИТЕТ ИТМО

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
%matplotlib inline
```

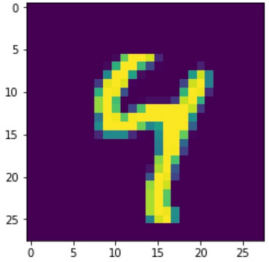
```
data = input_data.read_data_sets("./data/mnist", one_hot = True)
```

```
print("Training set images shape: {}".format(data.train.images.shape))
print("Training set labels shape: {}".format(data.train.labels.shape))
print("Test set images shape: {}".format(data.test.images.shape))
print("Test set labels shape: {}".format(data.test.labels.shape))
```

```
Training set images shape: (55000, 784)
Training set labels shape: (55000, 10)
Test set images shape: (10000, 784)
Test set labels shape: (10000, 10)
```

```
img = np.reshape(data.train.images[2], (28,28))
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x131d5e208>
```




```
print("Max Value: {}\nMin Value: {}".format(np.max(img), np.min(img)))
```

```
Max Value: 1.0
Min Value: 0.0
```

```
train_X = data.train.images.reshape(-1, 28, 28, 1)
test_X = data.test.images.reshape(-1, 28, 28, 1)
print("Train Shape: {}\nTest Shape: {}".format(train_X.shape, test_X.shape))
```

```
Train Shape: (55000, 28, 28, 1)
Test Shape: (10000, 28, 28, 1)
```

```
train_y = data.train.labels
test_y = data.test.labels
```

```
epochs = 10
learning_rate = 0.001
batch_size = 128
```

```
n_input = 28
n_classes = 10
```

```
x = tf.placeholder("float", [None, 28, 28, 1])
y = tf.placeholder("float", [None, n_classes])
```

```
def conv2d(x, W, b, stride = 1):
    x = tf.nn.conv2d(x, W, strides = [1, stride, stride, 1], padding = "SAME")
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)

def maxpool2d(x, k = 2):
    return tf.nn.max_pool(x, ksize = [1, k , k , 1], strides = [1, k , k, 1], padding = "SAME")
```

# MNIST-Recognition-CNN

УНИВЕРСИТЕТ ИТМО

```
weights = {
    'wc1' : tf.get_variable('W0', shape = (3, 3, 1, 32), initializer = tf.contrib.layers.xavier_initializer()),
    'wc2' : tf.get_variable('W1', shape = (3, 3, 32, 64), initializer = tf.contrib.layers.xavier_initializer()),
    'wc3' : tf.get_variable('W2', shape = (3, 3, 64, 128), initializer = tf.contrib.layers.xavier_initializer()),
    'wd1' : tf.get_variable('W3', shape = (4 * 4 * 128, 128), initializer = tf.contrib.layers.xavier_initializer()),
    'out' : tf.get_variable('W4', shape = (128, n_classes), initializer = tf.contrib.layers.xavier_initializer())
}

biases = {
    'bc1': tf.get_variable('B0', shape = (32), initializer = tf.contrib.layers.xavier_initializer()),
    'bc2': tf.get_variable('B1', shape = (64), initializer = tf.contrib.layers.xavier_initializer()),
    'bc3': tf.get_variable('B2', shape = (128), initializer = tf.contrib.layers.xavier_initializer()),
    'bd1': tf.get_variable('B3', shape = (128), initializer = tf.contrib.layers.xavier_initializer()),
    'out': tf.get_variable('B4', shape = (10), initializer = tf.contrib.layers.xavier_initializer()),
}
```

```
def conv_net(x, weights, biases):
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    conv1 = maxpool2d(conv1, k = 2)
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    conv2 = maxpool2d(conv2, k = 2)
    conv3 = conv2d(conv2, weights['wc3'], biases['bc3'])
    conv3 = maxpool2d(conv3, k = 2)
    fc1 = tf.reshape(conv3, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
    return out
```

```
pred = conv_net(x, weights, biases)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = pred, labels = y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

ITMOre than a UNIVERSITY

```
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

# MNIST-Recognition-CNN

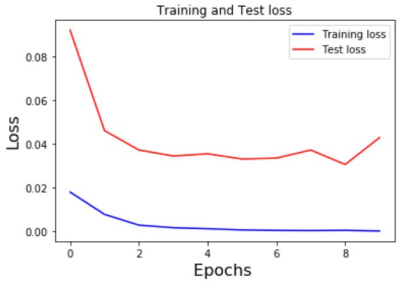
УНИВЕРСИТЕТ ИТМО

```
init = tf.global_variables_initializer()
```

```
with tf.Session() as sess:
    sess.run(init)
    train_loss = []
    test_loss = []
    train_accuracy = []
    test_accuracy = []
    summary_writer = tf.summary.FileWriter('./Output', sess.graph)
    for i in range(epochs):
        for batch in range(len(train_X)//batch_size):
            batch_x = train_X[batch*batch_size:min((batch+1)*batch_size,len(train_X))]
            batch_y = train_y[batch*batch_size:min((batch+1)*batch_size,len(train_y))]
            opt = sess.run(optimizer, feed_dict={x: batch_x,
                                                 y: batch_y})
            loss, acc = sess.run([cost, accuracy], feed_dict={x: batch_x,
                                                              y: batch_y})
        print("Iter " + str(i) + ", Loss= " + \
                      "{:.6f}".format(loss) + ", Training Accuracy= " + \
                      "{:.5f}".format(acc))
        print("Optimization Finished!")


        test_acc,valid_loss = sess.run([accuracy,cost], feed_dict={x: test_X,y : test_y})
        train_loss.append(loss)
        test_loss.append(valid_loss)
        train_accuracy.append(acc)
        test_accuracy.append(test_acc)
        print("Testing Accuracy:","{:.5f}".format(test_acc))
    summary_writer.close()
```

```
Iter 0, Loss= 0.017856, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.97040
Iter 1, Loss= 0.007680, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.98530
Iter 2, Loss= 0.002753, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.98720
Iter 3, Loss= 0.001578, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.98840
Iter 4, Loss= 0.001109, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.98910
Iter 5, Loss= 0.000578, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.99010
Iter 6, Loss= 0.000345, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.99040
Iter 7, Loss= 0.000277, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.98930
Iter 8, Loss= 0.000386, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.99140
Iter 9, Loss= 0.000063, Training Accuracy= 1.00000
Optimization Finished!
Testing Accuracy: 0.98830
```

```
plt.plot(range(len(train_loss)), train_loss, 'b', label='Training loss')
plt.plot(range(len(train_loss)), test_loss, 'r', label='Test loss')
plt.title('Training and Test loss')
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.legend()
plt.figure()
plt.show()
```




```
<matplotlib.figure.Figure at 0x1289de4a8>
```

```
plt.plot(range(len(train_loss)), train_accuracy, 'b', label='Training Accuracy')
plt.plot(range(len(train_loss)), test_accuracy, 'r', label='Test Accuracy')
plt.title('Training and Test Accuracy')
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.legend()
plt.figure()
plt.show()
```

# Cifar-10 - CNN

УНИВЕРСИТЕТ ИТМО

## Распознавание объектов на изображениях из набора данных CIFAR-10

```
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Dropout
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras.preprocessing import image
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Размер мини-выборки
batch_size = 128
# Количество классов изображений
nb_classes = 10
# Количество эпох для обучения
nb_epoch = 25
# Размер изображений
img_rows, img_cols = 32, 32
# Количество каналов в изображении: RGB
img_channels = 3
# Названия классов из набора данных CIFAR-10
classes=['самолет', 'автомобиль', 'птица', 'кот', 'олень', 'собака', 'лягушка', 'лошадь',
```

## Подготовка данных

**Загружаем данные**

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

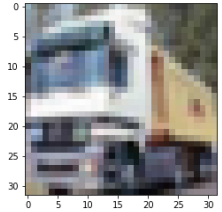
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
170508288/170498071 [==============================] - 3s 0us/step
```

**Просмотр примеров данных**

```
n = 1
plt.imshow(X_train[n])
plt.show()
print("Номер класса:", y_train[n])
print("Тип объекта:", classes[y_train[n][0]])
```

ITMO more than a UNIVERSITY

# Cifar-10 - CNN



```
Номер класса: [9]
Тип объекта: грузовик
```

**Нормализуем данные**

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

**Преобразуем правильные ответы в формат one hot encoding**

```
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)
```

## Создаем нейронную сеть

```
# Создаем последовательную модель
model = Sequential()
# Первый сверточный слой
model.add(Conv2D(32, (3, 3), padding='same',
                        input_shape=(32, 32, 3), activation='relu'))
# Второй сверточный слой
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
# Первый слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Слой регуляризации Dropout
model.add(Dropout(0.25))

# Третий сверточный слой
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
# Четвертый сверточный слой
model.add(Conv2D(64, (3, 3), activation='relu'))
# Второй слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Слой регуляризации Dropout
```

# Cifar-10 - CNN

```
model.add(Dropout(0.25))
# Слой преобразования данных из 2D представления в плоское
model.add(Flatten())
# Полносвязный слой для классификации
model.add(Dense(512, activation='relu'))
# Слой регуляризации Dropout
model.add(Dropout(0.5))
# Выходной полносвязный слой
model.add(Dense(nb_classes, activation='softmax'))
```

**Печатаем информацию о сети**

```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 14, 14, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 7, 7, 64)          0

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 512)               1606144

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 1,676,842
Trainable params: 1,676,842
Non-trainable params: 0
_________________________________________________________________
None
```

**Компилируем модель**

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
```

# Cifar-10 - CNN

УНИВЕРСИТЕТ ИТМО

```
3              metrics=['accuracy'])
```

Обучаем нейронную сеть

```
1 history = model.fit(X_train, Y_train,
2              batch_size=batch_size,
3              epochs=nb_epoch,
4              validation_split=0.1,
5              shuffle=True,
6              verbose=2)
```

```
Epoch 1/25
352/352 - 17s - loss: 1.6466 - accuracy: 0.3946 - val_loss: 1.2658 - val_accur
Epoch 2/25
352/352 - 8s - loss: 1.2304 - accuracy: 0.5587 - val_loss: 1.0323 - val_accura
Epoch 3/25
352/352 - 8s - loss: 1.0469 - accuracy: 0.6290 - val_loss: 0.8937 - val_accura
Epoch 4/25
352/352 - 8s - loss: 0.9300 - accuracy: 0.6713 - val_loss: 0.7968 - val_accura
Epoch 5/25
352/352 - 8s - loss: 0.8543 - accuracy: 0.7004 - val_loss: 0.7722 - val_accura
Epoch 6/25
352/352 - 8s - loss: 0.7911 - accuracy: 0.7223 - val_loss: 0.7342 - val_accura
Epoch 7/25
352/352 - 8s - loss: 0.7403 - accuracy: 0.7397 - val_loss: 0.7037 - val_accura
Epoch 8/25
352/352 - 8s - loss: 0.7025 - accuracy: 0.7514 - val_loss: 0.6814 - val_accura
Epoch 9/25
352/352 - 8s - loss: 0.6594 - accuracy: 0.7660 - val_loss: 0.6846 - val_accura
Epoch 10/25
352/352 - 8s - loss: 0.6282 - accuracy: 0.7794 - val_loss: 0.6393 - val_accura
Epoch 11/25
352/352 - 8s - loss: 0.5947 - accuracy: 0.7904 - val_loss: 0.6292 - val_accura
Epoch 12/25
352/352 - 8s - loss: 0.5692 - accuracy: 0.7981 - val_loss: 0.6084 - val_accura
Epoch 13/25
352/352 - 8s - loss: 0.5463 - accuracy: 0.8054 - val_loss: 0.6481 - val_accura
Epoch 14/25
352/352 - 8s - loss: 0.5207 - accuracy: 0.8156 - val_loss: 0.6224 - val_accura
Epoch 15/25
352/352 - 8s - loss: 0.5032 - accuracy: 0.8204 - val_loss: 0.6681 - val_accura
Epoch 16/25
352/352 - 8s - loss: 0.4817 - accuracy: 0.8264 - val_loss: 0.6006 - val_accura
Epoch 17/25
352/352 - 8s - loss: 0.4659 - accuracy: 0.8360 - val_loss: 0.6105 - val_accura
Epoch 18/25
352/352 - 8s - loss: 0.4574 - accuracy: 0.8353 - val_loss: 0.6123 - val_accura
Epoch 19/25
352/352 - 8s - loss: 0.4265 - accuracy: 0.8477 - val_loss: 0.6133 - val_accura
Epoch 20/25
352/352 - 8s - loss: 0.4217 - accuracy: 0.8493 - val_loss: 0.6169 - val_accura
Epoch 21/25
352/352 - 8s - loss: 0.4100 - accuracy: 0.8529 - val_loss: 0.6107 - val_accura
Epoch 22/25
352/352 - 8s - loss: 0.3884 - accuracy: 0.8581 - val_loss: 0.6587 - val_accura
Epoch 23/25
```

ITMOre than a UNIVERSITY

# Cifar-10 - CNN

```
352/352 - 8s - loss: 0.3907 - accuracy: 0.8616 - val_loss: 0.6107 - val_accura
Epoch 24/25
352/352 - 8s - loss: 0.3755 - accuracy: 0.8640 - val_loss: 0.6436 - val_accura
Epoch 25/25
352/352 - 8s - loss: 0.3611 - accuracy: 0.8703 - val_loss: 0.6303 - val_accura
```

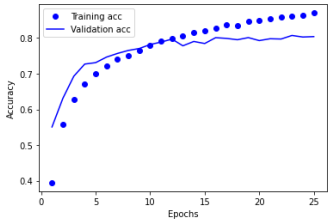
## Оцениваем качетсво обучения сети

```
# Оцениваем качество обучения модели на тестовых данных
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Точность работы на тестовых данных: %.2f%%" % (scores[1]*100))
```

```
Точность работы на тестовых данных: 79.79%
```

```
#@title Текст заголовка по умолчанию
history_dict = history.history
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
epochs = range(1, len(acc_values) + 1)
plt.plot(epochs, acc_values, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```

Текст заголовка по умолчанию




## Сохраняем обученную нейронную сеть

```
model_json = model.to_json()
json_file = open("cifar10_model.json", "w")
json_file.write(model_json)
json_file.close()
model.save_weights("cifar10_model.h5")
```

# Cifar-10 - CNN

```
1 !ls
```

```
cifar10_model.h5  cifar10_model.json  sample_data
```

```
1 from google.colab import files
```

```
1 files.download("cifar10_model.json")
```

Downloading "cifar10_model.json":

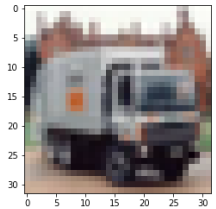
```
1 files.download("cifar10_model.h5")
```

Downloading "cifar10_model.h5":

Применяем сеть для распознавания объектов на изображениях

**Просмотр изображения из набора данных для тестирования**

```
1 from keras.preprocessing.image import array_to_img
2 index=11
3 img_pil = array_to_img(X_test[index])
4 plt.imshow(img_pil)
5 plt.show()
```




**Преобразование тестового изображения**

```
1 x = X_test[index]
2 x = np.expand_dims(x, axis=0)
```

**Запуск распознавания**

# Cifar-10 - CNN

```
1 prediction = model.predict(x)
```

**Печатаем результаты распознавания**

```
1 print(prediction)
```

```
[[4.63482551e-13 2.32940183e-05 3.39877317e-16 1.35745390e-14
  6.24044764e-20 1.12244315e-17 2.83191649e-16 3.51047263e-18
  2.38571191e-10 9.99976754e-01]]
```

**Преобразуем результаты из формата one hot encoding**

```
1 prediction = np.argmax(prediction)
2 print(classes[prediction])
```

```
грузовик
```

**Печатаем правильный ответ**

```
1 print(classes[y_test[index][0]])
```

```
грузовик
```

## Распознаем дополнительное изображение

```
1 from google.colab import files
2
3 files.upload()
```

```
Выбрать файлы  Файл не выбран    Upload widget is only available when the cell has been
executed in the current browser session. Please rerun this cell to enable.
Saving x_58a97df8.jpg to x_58a97df8.jpg
{'x_58a97df8.jpg': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x00H\x00H\x0
```

Проверяем загрузку файлов

```
1 !ls
```

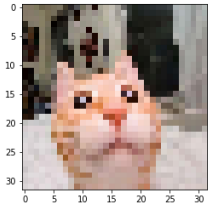
```
cifar10_model.h5  cifar10_model.json  sample_data  x_58a97df8.jpg
```

**Смотрим загруженную картинку**

```
1 img_path = 'x_58a97df8.jpg'
2 img = image.load_img(img_path, target_size=(32, 32))
```

# Cifar-10 - CNN

```
plt.imshow(img)
plt.show()
```



**Преобразуем картинку в массив для распознавания**

```
x = image.img_to_array(img)
x /= 255
x = np.expand_dims(x, axis=0)
```

**Запускаем распознавание**

```
prediction = model.predict(x)
prediction = np.argmax(prediction)
print(classes[prediction])
```

```
кот
```