**Классификация изображений**

**Теоретическая справка**

Для обучения нейронных сетей требуются большие объемы данных. Для задачи классификации используются стандартные наборы изображений, которые поставляются вместе с Keras :

* MNIST – 70'000 рукописных цифр; из них 60'000 входят в обучающую выборку, а 10'000 – в тестовую; размер каждого образа – 28\*28 пикселей; рисунки выполнены в оттенках серого цвета;
* EMNIST-letters – 145'600 рукописных букв английского алфавита; из них 124'800 входят в обучающую выборку, а 20'800 – в тестовую; размер каждого образа – 28\*28 пикселей; рисунки выполнены в оттенках серого цвета;
* CIFAR-10 – 60'000 цветных рисунков десяти классов: самолеты, легковые автомобили, птицы, кошки, олени, собаки, лягушки, лошади, корабли, грузовики; размер каждого образа – 32\*32 пикселей. В обучающую выборку входят 50'000 рисунков, а 10'000 – в тестовую.

Эти наборы данных загружаются в NumPy массив.

**Набор данных - CIFAR10**

С помощью следующего программного кода можно осуществить загрузку и подготовку данных CIFAR10 для дальнейшей обработки с помощью нейронных сетей:

*import numpy as np*

*import matplotlib.pyplot as plt*

*import matplotlib.font\_manager as font\_manager*

*from keras.datasets import cifar10*

*import os*

*#*

*def load\_cifar10(pathToData): # pathToData – путь к папке с данными*

*def unpickle(file):*

*import pickle*

*with open(file, 'rb') as fo:*

*dict\_file = pickle.load(fo, encoding = 'bytes')*

*return dict\_file*

*fileNm = os.path.join(pathToData, 'data\_batch\_')*

*# Загрузка обучающих данных*

*im\_size = 32 \* 32 \* 3 # 3072*

*X\_train = np.zeros((50000, im\_size), dtype = 'uint8')*

*y\_train = np.zeros(50000, dtype = 'uint8')*

*m = -1*

*for i in range(5):*

*dict\_i = unpickle(fileNm + str(i + 1))*

*X = dict\_i[b'data']*

*Y = dict\_i[b'labels']*

*Y = np.array(Y)*

*for k in range(10000):*

*m += 1*

*X\_train[m, :] = X[k, :]*

*y\_train[m] = Y[k]*

*X\_train = X\_train.reshape(50000, 3, 32, 32).transpose(0,2,3,1)*

*# Загрузка тестовых данных*

*len\_test = 10000*

*fileNm = os.path.join(pathToData, 'test\_batch')*

*dict\_i = unpickle(fileNm)*

*X\_test = dict\_i[b'data']*

*y\_test = dict\_i[b'labels']*

*X\_test = X\_test.reshape(len\_test, 3, 32, 32).transpose(0,2,3,1) # .astype('uint8')*

*y\_test = np.array(y\_test)*

*# Выводим 50 случайных изображений обучающего набора*

*names = []*

*names.append('Самолет')*

*names.append('Авто')*

*names.append('Птица')*

*names.append('Кошка')*

*names.append('Олень')*

*names.append('Собака')*

*names.append('Лягушка')*

*names.append('Лошадь')*

*names.append('Судно')*

*names.append('Грузовик')*

*fig, axs = plt.subplots(5, 10, figsize = (7, 4))*

*title\_font = {'fontname':'Arial', 'size':'9', 'color':'black'}*

*for j in range(5):*

*for k in range(10):*

*i = np.random.choice(range(50000))*

*ax1 = axs[j][k]*

*ax1.set\_axis\_off()*

*ax1.imshow(X\_train[i])*

*ax1.set\_title(names[y\_train[i]], \*\*title\_font)*

*plt.subplots\_adjust(hspace = 0.5, wspace = 0.0)*

*plt.show()*

*pathToData = os.path.join(os.getcwd(), 'cifar10')*

*load\_cifar10(pathToData)*

Результат работы программы

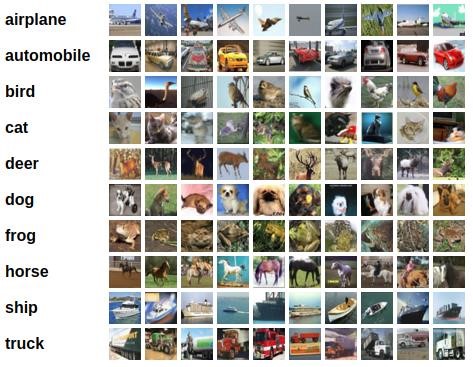
*from \_\_future\_\_ import print\_function import keras from keras.datasets import cifar10 from keras.preprocessing.image import ImageDataGenerator from keras.models import Sequential from keras.layers import Dense, Dropout, Activation, Flatten from keras.layers import Conv2D, MaxPooling2D import os batch\_size = 32 num\_classes = 10 epochs = 100 num\_predictions = 20 save\_dir = os.path.join(os.getcwd(), 'saved\_models') model\_name = 'keras\_cifar10\_trained\_model.h5'*

*# Разделяем данные на обучающий и тестовый наборы:*

*(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data() print('x\_train shape:', x\_train.shape) print(x\_train.shape[0], 'train samples') print(x\_test.shape[0], 'test samples')*

*# Преобразование векторов классов в двоичные матрицы y\_train = keras.utils.to\_categorical(y\_train, num\_classes) y\_test = keras.utils.to\_categorical(y\_test, num\_classes)*

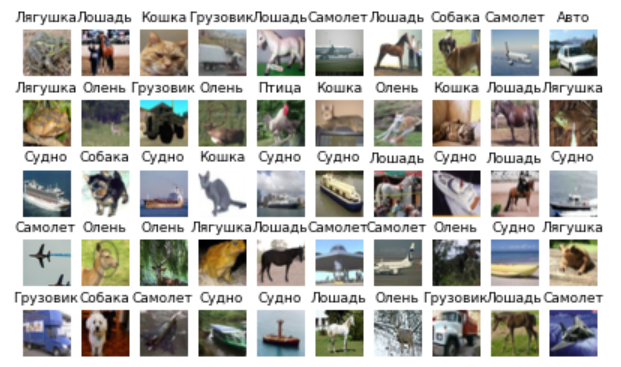
Изображения имеют размер 32 × 32. На рисунке приведены несколько примеров.



Сверточная нейронная сеть будет состоять из сверточных слоев, и слоев **MaxPooling**. Мы также включим **Dropout** слой для избежания переобучения. На выходе сети мы добавим полносвязный слой (**Dense**), за которым следует слой **softmax**. Здесь приведен программный код создания структуры модели. В приведенном выше коде мы используем 6 сверточных слоев и 1 полносвязный слой. Сначала в модель добавляем сверточные слои с 32 фильтрами с размером окна 3 × 3. Далее мы добавляем сверточныйслой с 64 фильтрами. За каждым слоем добавлен слой максимального пуллинга с размером окна 2 × 2. Также добавлены слои **Dropout** с коэффициентами 0,25 и 0.5 для того чтобы не произошло переобучение сети. В заключительных строках мы добавляем плотный слой **Dense**, который выполняет классификацию среди 10 классов с использованием функции активации **softmax.**

*model = Sequential() model.add(Conv2D(32,(3,3),padding='same', input\_shape=x\_train.shape[1:])) model.add(Activation('relu')) model.add(Conv2D(32, (3, 3))) model.add(Activation('relu'))*

*model.add(MaxPooling2D(pool\_size=(2, 2))) model.add(Dropout(0.25)) model.add(Conv2D(64, (3, 3), padding='same')) model.add(Activation('relu')) model.add(Conv2D(64, (3, 3))) model.add(Activation('relu')) model.add(MaxPooling2D(pool\_size=(2, 2))) model.add(Dropout(0.25)) model.add(Flatten()) model.add(Dense(512)) model.add(Activation('relu')) model.add(Dropout(0.5)) model.add(Dense(num\_classes)) model.add(Activation('softmax')) model.summary()*

******

**Обучение сети**

Поскольку это проблема классификации по 10 классам, мы будем использовать категорическую потерю энтропии и использовать оптимизатор RMSProp для обучения сети.

# initiate RMSprop optimizer opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop model.compile(loss='categorical\_crossentropy',optimizer=opt,metrics=['accuracy']) x\_train = x\_train.astype('float32') x\_test = x\_test.astype('float32') x\_train /= 255 x\_test /= 255

//Запустим его на количество эпох epochs.

model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=epochs, validation\_data=(x\_test, y\_test), shuffle=True)

# Сохраняем модель и веса

if not os.path.isdir(save\_dir):

os.makedirs(save\_dir) model\_path = os.path.join(save\_dir, model\_name) model.save(model\_path) print('Saved trained model at %s ' % model\_path)

**Распознавание рукописных цифр с использованием сверточных нейронных сетей**

MNIST - это набор данных, разработанный Янном ЛеКуном, Коринной Кортес и Кристофер Бургес для оценки моделей машинного обучения по проблеме классификации рукописных цифр. Набор данных был построен из ряда отсканированных наборов документов, доступных в Национальном институте стандартов и технологий (NIST). Изображения цифр были взяты из множества отсканированных документов, нормированных по размеру и по центру. Это делает его отличным набором данных для оценки моделей, позволяя разработчику сосредоточиться на механизме обучения с очень небольшой очисткой данных или необходимой подготовкой.

Каждое изображение представляет собой квадрат размером 28 на 28 пикселей (всего 784 пикселя). В этом наборе 60 000 изображений используются для обучения модели, и для ее тестирования используется отдельный набор из 10 000 изображений. Это задача распознавания 10 цифр (от 0 до 9) или классификация на 10 классов.

Библиотека глубокого обучения Keras предоставляет удобный метод **mnist.load\_data()** для загрузки набора данных MNIST. Набор данных загружается автоматически при первом вызове этой функции и сохраняется в вашем домашнем каталоге в ~ / .keras /datasets /mnist.pkl.gz в виде файла 15 МБ.

Это очень удобно для разработки и тестирования моделей глубокого обучения.

Чтобы продемонстрировать, насколько легко загружать набор данных MNIST, мы сначала напишем небольшой скрипт для загрузки и визуализации первых четырех изображений в наборе учебных материалов.

*from keras.datasets import mnist import matplotlib.pyplot as plt*

*# load (downloaded if needed) the MNIST dataset*

*(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()*

*# plot 4 images as gray scale plt.subplot(221) plt.imshow(X\_train[0], cmap=plt.get\_cmap('gray')) plt.subplot(222) plt.imshow(X\_train[1], cmap=plt.get\_cmap('gray')) plt.subplot(223) plt.imshow(X\_train[2], cmap=plt.get\_cmap('gray')) plt.subplot(224) plt.imshow(X\_train[3], cmap=plt.get\_cmap('gray'))*

*# show the plot plt.show()*

Запустив приведенный выше пример, вы должны увидеть изображение ниже.



**Базовая модель с многослойным перцептроном**

Используем очень простую модель нейронной сети с одним скрытым слоем. Начнем с импорта классов и функций, которые нам понадобятся.

*import numpy from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout*

*from keras.utils import np\_utils*

Учебный набор данных структурирован как трехмерный массив. Чтобы подготовить данные, сперва мы представим изображения в виде одномерных массивов (так как считаем каждый пиксель отдельным входным признаком). В этом случае изображения размером 28 × 28 будут преобразованы в массивы, содержащие 784 элементов.

Мы можем сделать это преобразование, используя функцию **reshape()** библиотеки **NumPy**. Для уменьшения потребления оперативной памяти преобразуем точность значений пикселей в 32. *(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data() num\_pixels = X\_train.shape[1] \* X\_train.shape[2]*

*X\_train = X\_train.reshape(X\_train.shape[0], num\_pixels).astype('float32') X\_test = X\_test.reshape(X\_test.shape[0], num\_pixels).astype('float32')*

Значения пикселей заданы в оттенках серого со значениями от 0 до 255. Для эффективного обучения нейронных сетей практически всегда рекомендуется выполнять некоторое масштабирование входных значений. Мы можем нормализовать значения пикселей в диапазоне 0 и 1, разделив каждое значение на максимальные значения 255.

*X\_train = X\_train / 255*

*X\_test = X\_test / 255*

Выходная переменная представляет собой целое число от 0 до 9, т.к. это задача классификации с несколькими классами. Хорошей практикой является использование кодирования значений класса преобразованием вектора целых чисел класса в двоичную матрицу.

Мы можем легко сделать это, используя встроенную вспомогательную функцию **np\_utils.to\_categorical ()** в Keras. *y\_train = np\_utils.to\_categorical(y\_train) y\_test = np\_utils.to\_categorical(y\_test)*

*num\_classes = y\_test.shape[1]*

Теперь создадим нашу простую модель однослойной нейронной сети и определим ее в функции. *def baseline\_model():*

*# create model model = Sequential() model.add(Dense(num\_pixels,input\_dim=num\_pixels, kernel\_initializer='normal', activation='relu')) model.add(Dense(num\_classes,kernel\_initializer='normal', activation='softmax')) model.compile(loss='categorical\_crossentropy',optimizer='adam', metrics=['accuracy'])*

*return model*

Модель представляет собой простую нейронную сеть с одним скрытым слоем с таким же количеством нейронов, что и количество входов (784). В скрытом слое используем полулинейную функцию активации **relu**.

На выходном слое используется функция активации **softmax** для преобразования выходов в вероятностные значения и позволяет выбрать один класс из 10 в качестве выходного значения модели. Теперь нам осталось только определить функцию потерь, алгоритм оптимизации и метрики, которые мы будет собирать. В задачах с вероятностной классификацией, в качестве функции потерь лучше всего использовать не квадратичную ошибку, а перекрестную энтропию. Потери будут меньше для вероятностных задач (например, с логистической/**softmax** функцией для выходного слоя), в основном из-за того, что данная функция предназначена для максимизации уверенности модели в правильном определении класса, и ее не заботит распределение вероятностей попадания образца в другие классы. Используемый алгоритм оптимизации будет напоминать какую-то форму алгоритма градиентного спуска, отличие будет лишь в том, как выбирается скорость обучения. В нашем случае мы будем использовать оптимизатор Адама, который обычно показывает хорошую производительность. Так как наши классы сбалансированы (количество рукописных цифр, принадлежащих каждому классу, одинаково), подходящей метрикой будет точность (**accuracy**) — доля входных данных, отнесенных к правильному классу.

Теперь мы можем обучить и оценить качество обученности модели. *model = baseline\_model() model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=10, batch\_size=200, verbose=2) scores = model.evaluate(X\_test, y\_test, verbose=0)*

*print("Baseline Error: %.2f%%" % (100-scores[1]\*100))*

Модель подходит 10 эпох обучения, при каждом обновлении весов используется 200 изображений. Тестовые данные которые используются в качестве набора данных валидации, позволяют вам видеть качество распознавания модели по мере ее обучения. Значение **verbose** =2 используется для уменьшения вывода на одну строку для каждой учебной эпохи. Наконец, тестовый набор данных используется для оценки модели и печатается ошибка классификации.

**Простая сверточная нейронная сеть для MNIST**

Создадим простую CNN для MNIST, которая продемонстрирует, как использовать все аспекты современной реализации CNN.

Первый шаг - импортировать необходимые классы и функции.

*import numpy from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.layers import Flatten from keras.layers.convolutional import Conv2D from keras.layers.convolutional import MaxPooling2D from keras.utils import np\_utils from keras import backend as K*

*K.set\_image\_dim\_ordering('th')*

Далее инициализируем генератор случайных чисел на постоянное начальное значение для воспроизводимости результатов.

*seed = 7*

*numpy.random.seed(seed)*

Затем нам нужно загрузить набор данных MNIST и изменить его, чтобы он был подходящим для обучения CNN.

*(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()*

*# reshape to be [samples][pixels][width][height]*

*X\_train = X\_train.reshape(X\_train.shape[0], 1, 28, 28).astype('float32')*

*X\_test = X\_test.reshape(X\_test.shape[0], 1, 28, 28).astype('float32')*

*Как и прежде нормализуем значения пикселей в диапазоне 0 и 1.*

*X\_train = X\_train / 255 X\_test = X\_test / 255 y\_train = np\_utils.to\_categorical(y\_train) y\_test = np\_utils.to\_categorical(y\_test)*

*num\_classes = y\_test.shape[1]*

Затем мы определяем модель нейронной сети. Сверточные нейронные сети более сложны, чем стандартные многослойные персептроны, поэтому мы начнем с использования простой структуры Ниже представлена архитектура сети.

1. Первый скрытый слой - это сверточный слой, Convolution2D. Этот слой имеет 32 карты функций, размер которых равен 5 × 5 и функции активации relu.

2. Затем мы определяем слой пулинга maxPooling2D с размером пула 2 × 2, который дает максимальные значения.

3. Следующий уровень - это уровень регуляризации Dropout. Он настроен на случайное исключение 20% нейронов в слое, чтобы уменьшить переобучение.

4. Далее - слой, который преобразует данные двумерной матрицы в вектор, называемый Flatten. Он позволяет обрабатывать выходные данные стандартными полносвязными слоями.

5. Затем полносвязный слой с 128 нейронами и функцией активации relu.

6. Наконец, выходной слой имеет 10 нейронов для 10 классов и функцию активации softmax для вывода вероятностных результатов распознавания для каждого класса.

*def baseline\_model():*

*# create model model = Sequential() model.add(Conv2D(32, (5, 5), input\_shape=(1, 28, 28), activation='relu')) model.add(MaxPooling2D(pool\_size=(2, 2))) model.add(Dropout(0.2)) model.add(Flatten()) model.add(Dense(128, activation='relu')) model.add(Dense(num\_classes, activation='softmax'))*

*# Compile model*

*model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['ac curacy'])*

*return model*

Как и в примере с моногослойным персептроном эта модель 10 эпох обучения, при каждом обновлении весов используется 200 изображений.

*model = baseline\_model() model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=10, batch\_size=200, verbose=2) scores = model.evaluate(X\_test, y\_test, verbose=0)*

*print("CNN Error: %.2f%%" % (100-scores[1]\*100))*

Точность классификации модели печатается в каждую эпоху обучения и в конце отпечатается ошибка ошибки классификации.