

Методические указания по выполнению лабораторной работы №3 «Распознавание изображений»

В данной лабораторной работе мы будем заниматься обучением сверточных нейронных сетей и распознаванием изображений с их помощью. Проведем сравнение сверточной нейронной сети и полносвязной нейронной сети для распознавания набора MNIST. Обучим сверточную нейронную сеть для распознавания изображений в наборе CIFAR-10. Научимся вычислять классические показатели качества классификации: матрицу ошибок (confusion matrix), accuracy, precision, recall, F-score.

Начало работы, импорт библиотек и модулей

Для работы потребуется создать папку на своем Google-диске, поместить в нее новый блокнот, а также скопировать собственное изображение рукописной цифры, созданное в ЛР №1. Далее необходимо предоставить блокноту доступ к диску и установить путь к созданной папке:

```
import os
os.chdir('/content/drive/MyDrive/Colab Notebooks/IS_LR3')
```

Для ускорения обучения рекомендуется подключиться к аппаратному ускорителю GPU (Среда выполнения > Сменить среду выполнения > Аппаратный ускоритель > GPU).

Программный код для импорта необходимых библиотек может выглядеть следующим образом:

```
# импорт модулей
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

Загрузка набора данных

Загрузка набора данных MNIST и разбиение его на обучающие и тестовые данные происходит в точности, как в лабораторной работе №1:

```
# загрузка датасета
from keras.datasets import mnist

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

```

# создание своего разбиения датасета
from sklearn.model_selection import train_test_split

# объединяем в один набор
X = np.concatenate((X_train, X_test))
y = np.concatenate((y_train, y_test))

# разбиваем по вариантам
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 10000,
                                                    train_size = 60000,
                                                    random_state = 123)

```

```

# вывод размерностей
print('Shape of X train:', X_train.shape)
print('Shape of y train:', y_train.shape)

print('Shape of X test:', X_test.shape)
print('Shape of y test:', y_test.shape)

```

Предобработка данных

Как известно, в исходном наборе данных MNIST изображения представляются в виде двумерных массивов размером 28 на 28 с целыми значениями интенсивности каждого пикселя от 0 до 255. Хотя сверточные сети подходят для обработки данных в виде двумерных массивов, тем не менее, данные все равно требуют изменения размерности. Дело в том, что сверточный слой в Keras принимает на вход не просто двумерный массив с размерностью (высота, ширина), а трехмерный массив с размерностью (высота, ширина, количество каналов). Для цветных изображений количество каналов равно 3 – по количеству цветов в модели RGB. Изображения в MNIST монохромные: количество каналов равно 1. Поэтому для перевода изображений в корректную размерность воспользуемся функцией `numpy.expand_dims` с параметром `axis = -1`. Эта функция добавляет размерность переданного ей массива данных.

Выходные значения (метки цифр) стандартно переведем в one-hot encoding.

```

# Зададим параметры данных и модели
num_classes = 10
input_shape = (28, 28, 1)

# Приведение входных данных к диапазону [0, 1]
X_train = X_train / 255
X_test = X_test / 255

# Расширяем размерность входных данных, чтобы каждое изображение имело
# размерность (высота, ширина, количество каналов)

```

```

X_train = np.expand_dims(X_train, -1)
X_test = np.expand_dims(X_test, -1)
print('Shape of transformed X train:', X_train.shape)
print('Shape of transformed X test:', X_test.shape)

# переведем метки в one-hot
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
print('Shape of transformed y train:', y_train.shape)
print('Shape of transformed y test:', y_test.shape)

```

Реализация сверточной нейронной сети и оценка качества классификации

Сверточная нейронная сеть состоит из слоев свертки и подвыборки (пуллинга). Данные слои реализуются в Keras классами Conv2D и MaxPooling2D. Основными параметрами слоя Conv2D являются количество фильтров, размер фильтра, функция активации и др. Для первого слоя необходимо также указать входную размерность данных. Основным параметром слоя MaxPooling2D является размер окна пуллинга.

После нескольких слоев свертки и пуллинга в сверточной сети обычно добавляют один или несколько полносвязных слоев (Dense) для классификации, подобно многослойному персептрон. Между слоями, обрабатывающими многомерные массивы, и полносвязными слоями необходимо добавить слой Flatten. Данный слой «вытянет» многомерный массив на выходе слоя пуллинга в одномерный вектор, пригодный для подачи на полносвязный слой.

Также в глубоких сетях с целью регуляризации, предотвращения перевозбуждения связей между нейронами, используют слой Dropout. Основным параметром данного слоя является вероятность, с которой выходы случайных нейронов предыдущего слоя будут обнулены, и обучение через которые проходить не будет.

Подробно параметры слоев можно посмотреть в документации [Keras: Conv2D MaxPooling2D Flatten Dropout Dense](#).

Программный код для создания сверточной нейронной сети и вывода информации о ее архитектуре может выглядеть следующим образом (**можно изменять по своему усмотрению**):

```

# создаем модель
model = Sequential()
model.add(layers.Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=input_shape))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, kernel_size=(3, 3), activation="relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.5))
model.add(layers.Flatten())
model.add(layers.Dense(num_classes, activation="softmax"))

model.summary()

```

После того, как мы собрали модель, остается ее скомпилировать и обучить.

```
# компилируем и обучаем модель
batch_size = 512
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

После того, как модель обучена, оценка качества ее работы на тестовых данных может быть произведена с помощью функции `evaluate`. Функция вернет список, состоящий из значения функции ошибки и метрики качества на тестовых данных.

```
# Оценка качества работы модели на тестовых данных
scores = model.evaluate(X_test, y_test)
print('Loss on test data:', scores[0])
print('Accuracy on test data:', scores[1])
```

Применение обученной модели

Для вывода результата обработки нейронной сетью одного изображения воспользуемся функцией `predict`. Для определения результата распознавания необходимо воспользоваться функцией `argmax`, то есть определить на каком месте стоит нейрон, выдающий наибольшее значение.

```
# вывод тестового изображения и результата распознавания
n = 123
result = model.predict(X_test[n:n+1])
print('NN output:', result)

plt.imshow(X_test[n].reshape(28,28), cmap=plt.get_cmap('gray'))
plt.show()
print('Real mark: ', np.argmax(y_test[n]))
print('NN answer: ', np.argmax(result))
```

Обратите внимание, что при вызове функции `predict` на вход сети должен подаваться массив, имеющий четыре размерности (количество примеров, высота, ширина, количество каналов), поэтому следует воспользоваться срезом (slice) `X_test[n:n+1]`, а не просто `X_test[n]`, который бы был трехмерным массивом.

Вычисление показателей качества классификации

Для вычисления матрицы ошибок классификации, а также основанных на ней показателей качества классификации `accuracy`, `precision`, `recall`, `F-score` требуются два

массива: массив истинных и массив предсказанных (назначенных моделью) меток тестовых объектов.

Для вычисления указанных характеристик можно воспользоваться функциями из пакета sklearn: `classification_report` и `confusion_matrix`. Массивы меток передаются им как входные параметры. Подробнее о показателях качества можно посмотреть в документации [Sklearn: Confusion matrix Classification report Accuracy Precision, Recall, F-score, Support](#)

```
# истинные метки классов
true_labels = np.argmax(y_test, axis=1)
# предсказанные метки классов
predicted_labels = np.argmax(model.predict(X_test), axis=1)

# отчет о качестве классификации
print(classification_report(true_labels, predicted_labels))
# вычисление матрицы ошибок
conf_matrix = confusion_matrix(true_labels, predicted_labels)
# отрисовка матрицы ошибок в виде "тепловой карты"
display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
display.plot()
plt.show()
```

Распознавание собственного изображения

Для распознавания собственного изображения, так же как в лабораторной работе №1, воспользуемся библиотекой PIL (Python Image Library), загрузим изображение как массив, отобразим его, предобработаем, и подадим на вход обученной модели для распознавания:

```
# загрузка собственного изображения
from PIL import Image
file_data = Image.open('test.png')
file_data = file_data.convert('L') # перевод в градации серого
test_img = np.array(file_data)

# вывод собственного изображения
plt.imshow(test_img, cmap=plt.get_cmap('gray'))
plt.show()

# предобработка
test_img = test_img / 255
test_img = np.reshape(test_img, (1,28,28,1))

# распознавание
result = model.predict(test_img)
print('I think it\'s ', np.argmax(result))
```

Работа с набором CIFAR-10

Набор данных CIFAR-10 представляет собой 60 000 цветных изображений, размеченных на 10 классов: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль, грузовик.

Загрузка набора данных происходит с помощью функции `cifar10` в Keras:

```
# загрузка датасета
from keras.datasets import cifar10

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

Для вывода изображений с подписями классов можно воспользоваться следующим:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train[i])
    plt.xlabel(class_names[y_train[i][0]])
plt.show()
```

Программный код для создания сверточной нейронной сети и вывода информации о ее архитектуре может выглядеть следующим образом:

```
# создаем модель
model = Sequential()
model.add(layers.Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=input_shape))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, kernel_size=(3, 3), activation="relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(128, kernel_size=(3, 3), activation="relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation="softmax"))

model.summary()
```

Для отображения названий классов, а не числовых меток на отчете о качестве классификации и на матрице ошибок необходимо передать список `class_names` в соответствующие параметры функций:

```
# отчет о качестве классификации
print(classification_report(true_labels, predicted_labels, target_names=class_names))
# вычисление матрицы ошибок
conf_matrix = confusion_matrix(true_labels, predicted_labels)
# отрисовка матрицы ошибок в виде "тепловой карты"
display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=class_names)
display.plot()
plt.show()
```