УДК 519.68:007.5

Лабораторная работа 2. Сверточные сети и автоэнкодеры

Цели работы: работа с предобученными сверточными сетями. Адаптация готовых сетей под новые задачи распознания изображений.

Ключевые слова: сверточные сети, распознание изображений, автоэнкодеры.

§ 1. Общая схема для адаптации предобученных сверточных сетей

У готовой сверточной сети мы заменяем последние слои, которые отвечали за итоговую классификацию объектов. При этом все внутренние слои мы сохраняем без изменений, используя для них старые веса. Это позволяет избежать длительного обучения сверточных слоёв, которые уже были обучены на выделение примитивных признаков.

Начнем работу с подключения одной из предобученных сверточных нейронных сетей (список см. https://keras.io/applications/)

```
import keras
from keras.applications import MobileNet
from keras.applications.MobileNet import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
```

Для каждой из этих сетей предусмотрена своя функция предобработки изображений (preprocess_input). Эту функцию мы возьмём за основу для нашего генератора обучающих и тестовых примеров.

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)

train_generator=train_datagen.flow_from_directory('./Train/',
      target_size=(224,224),# размер итоговых изображений
      color_mode='rgb',# (альтернатива greyscale)
      batch_size=32, # генерируется за один проход
      class_mode='categorical',# режим множественной классификации
      shuffle=True)
```

Для корректной работы данного генератора потребуется создать директорию Train и разместить все изображения в её поддиректориях с названиями, соответсвующими названиям классов объектов. Аналогично должен быть создан и генератор для тестовых примеров (назовём его valid_generator), и отдельная директория для хранения тестовых изображений ('./Valid').

Возмьмем за основу модель MobileNet с весами для классификации изображений из набора Imagenet, отключив у неё выходные слои. На замену им добавим свои слои, подключив в качестве последнего — персептронный слой с числом нейронов равным числу итоговых классов изображений.

```
from keras.layers import Dense,GlobalAveragePooling2D
from keras.layers.core import Activation

base_model=MobileNet(weights='imagenet',include_top=False)
output_model=base_model.output
output_model=GlobalAveragePooling2D()(output_model)
output_model=Dense(1024,activation='relu')(output_model)
output_model=Dense(512,activation='relu')(output_model)
output_model=Dense(numClasses,activation='softmax')(output_model)
# последний слой с числом нейронов = числу классов
model=Model(inputs=base_model.input,outputs=output_model)
```

Для ускорения процесса обучения заморозим изменение весовых коэффициентов для всех внутренних слоев нашей сети.

```
for layer in base_model.layers:
    layer.trainable=False
```

Подготовим модель для обучения, указав в качестве метода градиентного спуска "Adam а в качестве оценки ошибки категоральную перекрестную энтропию (categorical crossentropy).

$$H(y, y^e) = -\sum_j \sum_i y_{i\,j} \cdot log\left(y^e_{i\,j}\right)$$

```
model.compile(optimizer='Adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Запустим процесс обучения с помощью функции fit_generator, указав ей источники для обучающих и тестовых данных, а также число эпох обучения.

```
step_size_train=train_generator.n//train_generator.batch_size
step_size_valid=valid_generator.n//valid_generator.batch_size
model.fit_generator(generator=train_generator,
                    steps_per_epoch=step_size_train,
                    validation_data=valid_generator,
                    validation_steps =step_size_valid,
                    epochs=10)
```

Итоговую модель мы можем экспортировать в файл с помощью функции model.save('our_first_convmodel.h5'). Проверить её работу далее можно с помощью функций predict и evaluate, предварительно прочитав из файла функцией model.load_model и скомпилировав model.compile со старыми настройками (loss='categorical_crossentropy', metrics=['accuracy']).

§ 2. Построение автоэнкодера на основе сверточных сетей

Для нашего примера возьмём базу изображений MNIST (рукописные цифры в монохромном формате). Искомый автоэнкодер должен будет проводить фильтрацию изображений и удалять из них шумы. Строить итоговую модель автоэнкодера мы будем на основе двух базовых моделей: энкодера и декодера.

```
from __future__ import absolute_import
from __future__ import division
import keras
from keras.datasets import mnist
import numpy as np
# Считываем данные из MNIST
(x_train, _), (x_test, _) = mnist.load_data()
# Нормируем данные и приводим массивы к виду (N,size1,size1,1)
image_size = x_train.shape[1]
x_train = np.reshape(x_train, [-1, image_size, image_size, 1])
x_test = np.reshape(x_test, [-1, image_size, image_size, 1])
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

Добавим в наши обучающие примеры случайный шум с нормальным законом распределения и $m = \sigma = 0.5$.

```
noise = np.random.normal(loc=0.5, scale=0.5, size=x_train.shape)
x_train_noisy = x_train + noise
noise = np.random.normal(loc=0.5, scale=0.5, size=x_test.shape)
x_test_noisy = x_test + noise
# Ограничим итоговые значения интервалом [0,1]
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

Зададим общие параметры для наших нейронных сетей.

```
input_shape = (image_size, image_size, 1)
batch_size = 128
kernel_size = 3
latent_dim = 16
# Зададим два сверточных слоя и число нейронов в каждом слое:
layer_filters = [32, 64]
```

Построим модель энкодера на основе стека сверточных слоев.

```
from keras.layers import Activation, Dense, Input
from keras.layers import Conv2D, Flatten
from keras.layers import Reshape, Conv2DTranspose
from keras.models import Model
from keras import backend as K

encoder_inputs = Input(shape=input_shape, name='encoder_input')
x = encoder_inputs
# Стек из сверточных слоев (strides - дискрет сдвига окна свертки
# в пикселях, padding - без заполнения нулями):
for filters in layer_filters:
    x = Conv2D(filters=filters,
               kernel_size=kernel_size,
               strides=2,
               activation='relu',
               padding='same')(x)
```

```
# Запоминаем размерность выхода для построения модели декодера
shape = K.int_shape(x)

# Преобразуем многомерный массив в вектор
x = Flatten()(x)
latent = Dense(latent_dim, name='latent_vector')(x)
# Итоговый код - одномерный вектор меньшей размерности (latent_dim)

# Итоговая модель энкодера:
encoder = Model(encoder_inputs , latent, name='encoder')
```

Распечатать краткую сводку по нашей модели можно командой encoder.summary(). Если всё было сделано без ошибок, то должно отобразиться:

```
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 64)          18496
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
latent_vector (Dense)        (None, 16)                50192
=================================================================
Total params: 69,008
Trainable params: 69,008
Non-trainable params: 0
_________________________________________________________________
```

Аналогичным образом построим модель декодера, задействовав информацию о размере (shape) последнего слоя свертки.

```
latent_inputs = Input(shape=(latent_dim,), name='decoder_input')
# Обратное преобразование к размеру "shape":
x = Dense(shape[1] * shape[2] * shape[3])(latent_inputs)
# Выход должен быть трехмерным массивом:
x = Reshape((shape[1], shape[2], shape[3]))(x)

# Вместо сверточных слоев "разверточные", цикл в обратном порядке:
for filters in layer_filters[::-1]:
    x = Conv2DTranspose(filters=filters,
                        kernel_size=kernel_size,
                        strides=2,
                        activation='relu',
                        padding='same')(x)
```

```
x = Conv2DTranspose(filters=1,
                    kernel_size=kernel_size,
                    padding='same')(x)

outputs = Activation('sigmoid', name='decoder_output')(x)

# Итоговая модель декодера:
decoder = Model(latent_inputs, outputs, name='decoder')
decoder.summary()
```

Объединив две модели в одну мы получим наш автоэнкодер:

```
autoencoder = Model(encoder_inputs,
                    decoder(encoder(encoder_inputs)),
                    name='autoencoder')
autoencoder.compile(loss='mse', optimizer='adam')

# Запускаем модель на обучение, используя незашумленные данные как эталонные з
autoencoder.fit(x_train_noisy,
                x_train,
                validation_data=(x_test_noisy, x_test),
                epochs=30,
                batch_size=batch_size)
```

Полученную модель также можно сохранить с помощью model.save.

§ 3. Практические задания

Задание № 1: Классификация изображений.

1. Выбрать изображения как минимум с 10 классами объектов. Датасеты должны отличаться от стандартных, а также от датасетов других студентов более чем на 60
2. Разбить изображения на обучающую (./Train/Class1 ./Train/Class2) и тестовую (./Valid/Class1 ./Valid/Class2) выборки и распределив их по соответсвующим директориям.
3. Импортировать сверточную сеть и адаптировать для обработки и распознания этих классов.
4. Сравнить качество и скорость обучения при использовании заморозки коэффициентов предобученных слове и без заморозки.
5. Обучить сеть с заморозкой предобученных слоев, а затем разморозить их и повторить обучение с меньшим числом эпох.

Задание № 2: Автоэнкодеры.

1. Доработать модель из разобранного примера, педусмотрев визуализацию итогового результата.
2. Вывести в виде изображений зашумленные и исправленные данные.
3. Реализовать альтернативную модель автоэнкодера с тремя слоями свертки/развертки. Сравнить качество работы с моделью из примера.