

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский
университет имени академика С.П. Королева (Самарский университет)»

Институт _____ информатики, математики и электроники

Факультет _____ информатики

Кафедра _____ программных систем

ОТЧЕТ

_____ к лабораторной работе по дисциплине
_____ «Нейронные сети глубокого обучения»

Студент _____ Е. Г. Плешаков

Студент _____ О.В. Ширяева

Преподаватель _____ А. Н. Жданова

Самара 2021

СОДЕРЖАНИЕ

1	Постановка задачи.....	3
2	Описание архитектуры сети, метода обучения.....	4
3	Вычислительные эксперименты	8
4	Результат работы программы на своих данных	8
5	Вывод.....	9
	Приложение А Код программы	10

Постановка задачи

Цель лабораторной работы: реализовать нейронную сеть для классификации изображений.

Классификация (рубрикация) изображений представляет собой отнесение изображения к одной из нескольких категорий (рубрик) на основании его содержания. В одной из своих многочисленных постановок задача распознавания образов трактуется как отнесение предъявленного объекта по его формальному описанию к одному из заданных классов. Важно, что классы (категории, рубрики) в задаче классификации являются пересекающимися подмножествами, это значит, что текст может быть отнесен к нескольким категориям одновременно.

- имеется множество категорий (классов, рубрик): $C = \{c_1, \dots, c_{|C|}\}$;
- имеется множество документов: $D = \{d_1, \dots, d_{|D|}\}$;
- существует неизвестная целевая функция: $\Phi : C \times D \rightarrow \{0,1\}$;
- имеется некоторая начальная коллекция размеченных изображений $R \subset C \times D$, для которых известны значения Φ . Её делят на «обучающую» и «тестируемую» части. Первая используется для обучения классификатора, вторая – для проверки качества его работы.
- необходимо создать классификатор Φ' , максимально близкий к Φ , который бы выдавал степень подобия $\Phi' : C \times D \rightarrow \{0,1\}$.

1 Описание архитектуры сети, метода обучения

Для решения поставленной задачи была построена свёрточная (convolutional) нейронная сеть, которая была обучена на нескольких тысячах изображений кошек и собак, чтобы затем предсказывать, кто изображен на картинке – кошка или собака.

Данные для датасета взяты с Kaggle: <https://www.kaggle.com/tongpython/cat-and-dog>. При работе над кодом сети использован источник <https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8>

Для создания CNN (свёрточной нейронной сети) была использована библиотека глубокого обучения Keras с бэкендом на TensorFlow на языке Python. Процесс построения CNN включает четыре основных этапа:

Шаг 1: Свертка (Convolution).

Шаг 2: Объединение (Pooling).

Шаг 3: Сглаживание (Flattening).

Шаг 4: Полное соединение (Full connection)

В качестве модели сети была выбрана последовательная (sequential) модель из `keras.models`. Существует два основных способа инициализации нейронной сети: последовательностью слоев или графом.

На первом шаге мы использовали двумерную свертку (`Conv2D` из `keras.layers`), поскольку здесь мы работаем с изображениями, которые представляют собой в основном двухмерные массивы. Трехмерные свёрточные слои используются, например, при работе с видео, где третьим измерением будет время.

Для второго шага – операции объединения – использована `MaxPooling2D` из `keras.layers`. С помощью `MaxPooling` мы выбираем пиксель максимального значения из соответствующей интересующей области.

Для третьего шага – сглаживания – используем `Flatten` из `keras.layers`. Сглаживание – это процесс преобразования всех результирующих двумерных массивов в один длинный непрерывный линейный вектор.

Для выполнения полного подключения нейронной сети на 4 шаге используем Dense из keras.layers.

Ниже приведён исходный код создания CNN.

```
from keras.layers import Conv2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import MaxPooling2D
from keras.models import Sequential
from tensorflow import keras

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1. / 255)

batch_size = 32

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                target_size=(64, 64),
                                                batch_size=batch_size,
                                                class_mode='binary')
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size=(63, 64),
                                            batch_size=batch_size,
                                            class_mode='binary')

import os

cp_callback = keras.callbacks.ModelCheckpoint(filepath=os.getcwd(),
                                              save_weights_only=True,
                                              verbose=1)

model.fit(training_set,
          steps_per_epoch=8000 // batch_size,
          epochs=25,
          validation_data=test_set,
          validation_steps=2000 // batch_size,
          callbacks=[cp_callback])

model.save("keras_cnn")
print("Model saved")
```

Разберем приведенный выше код. Мы взяли объект, который уже имеет представление о том, какой будет наша нейронная сеть (последовательная),

затем мы добавили свёрточный слой с помощью функции «Conv2D». Функция Conv2D принимает 4 параметра, первый – это количество фильтров, т.е. 32 здесь, второй аргумент – это форма, которую будет иметь каждый фильтр, то есть здесь 3x3, третий – это форма ввода и тип изображения (RGB или черно-белое) каждого изображения. Таким образом, входное изображение, которое будет принимать наша CNN, имеет разрешение 64x64, а код «3» означает RGB. Четвертый параметр – это функция активации, которую мы хотим использовать, здесь «relu» означает функцию выпрямителя (rectified linear unit).

Теперь нужно выполнить операцию объединения на результирующих картах характеристик, которые мы получаем после выполнения операции свёртки на изображении. Основная цель операции объединения – максимально уменьшить размер изображений, т.е. уменьшить общее количество узлов для следующих слоев.

Мы начинаем с того, что берём наш объект-классификатор и добавляем слой объединения. Взяв матрицу 2x2, с минимальной потерей пикселей мы получим точную область, в которой находится объект. Так, мы уменьшили сложность модели, не снизив ее производительности.

Далее преобразуем все объединённые изображения в непрерывный вектор с помощью сглаживания. По сути, мы берём двумерный массив, то есть объединённые пиксели изображения, и преобразуем их в одномерный одномерный вектор.

Теперь нужно создать полностью связанный слой, и к этому слою мы собираемся подключить набор узлов, которые мы получили после этапа выравнивания. Эти узлы будут действовать как входной слой для этих полностью связанных слоев. Поскольку этот слой будет находиться между входным и выходным слоями, мы можем назвать его скрытым слоем.

Dense – это функция для добавления полностью связанного слоя, «единицы» – это то место, где мы определяем количество узлов, которые должны присутствовать в этом скрытом слое, значение этих единиц всегда

будет между количеством входных и выходных узлов, но оптимальное количество узлов может быть достигнуто только экспериментальным путем. Чаще всего используют степень двойки и активационную функцию relu.

В конце инициализировать выходной слой, который должен содержать только один узел, поскольку это двоичная классификация. Этот единственный узел даст бинарный вывод: либо Cat, либо Dog. Для активации последнего слоя используем функцию сигмоида. Для компиляции выбираем алгоритм стохастического градиентного спуска («adam»), функцию потерь («binary_crossentropy»), метрику эффективности («accuracy»).

Для подготовки тренировочного датасета используем библиотеку keras.preprocessing, предлагающую различные типы операций, такие как переворачивание, вращение, размытие и т. д. Изображения в датасете структурированы по папкам и помечены как кошки или собаки. Параметром «steps_per_epoch» задаётся количество обучающих изображений в каждой эпохе. Эпоха – это один шаг в обучении нейронной сети; в нашем случае тренировочный процесс состоит из 25 эпох.

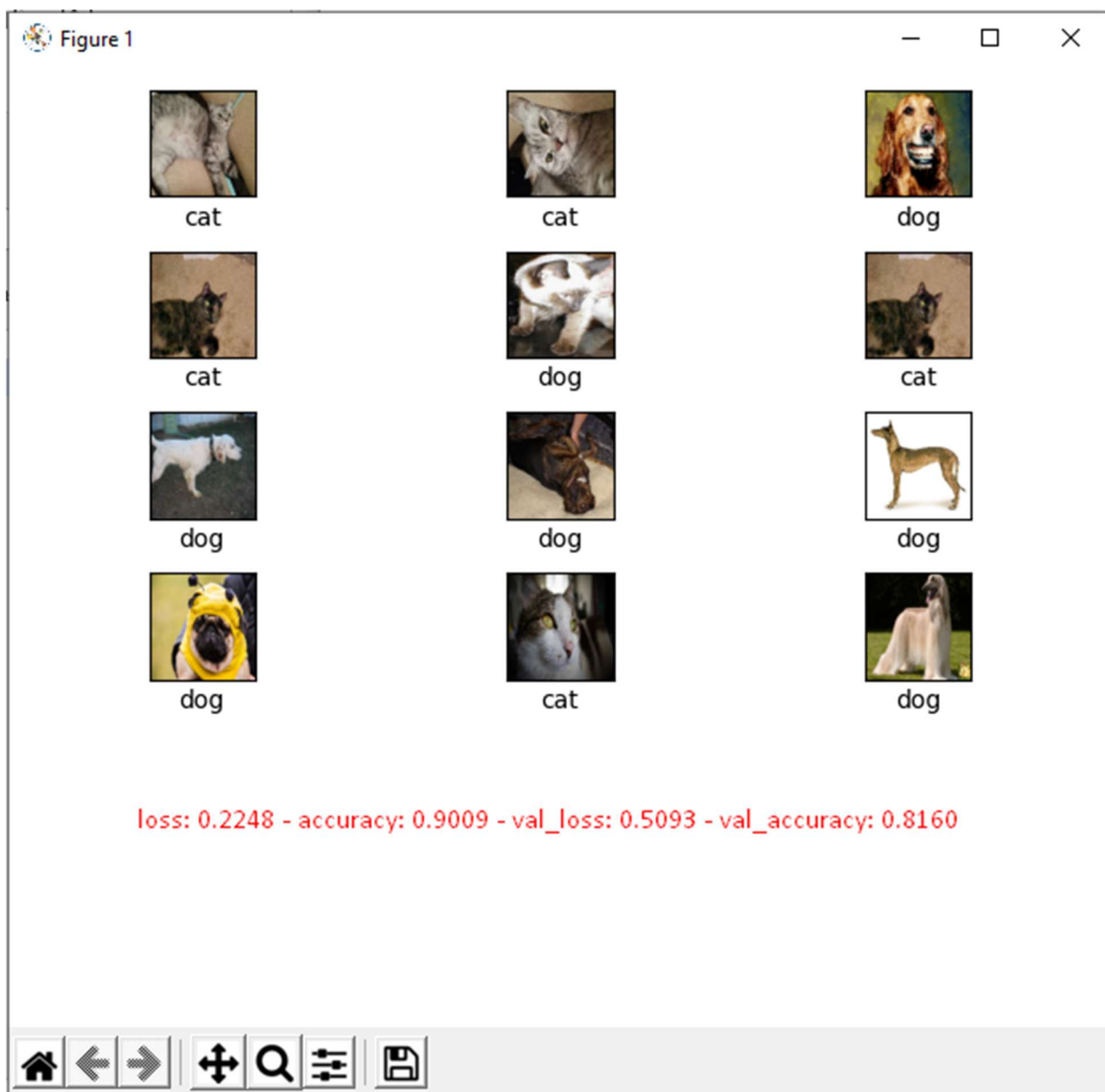
Для прогнозирования результатов обученной сетью используется метод «predict()», в который передаётся своё изображение. Поскольку прогноз будет в двоичной форме, мы получим либо 1, либо 0, что будет представлять собаку или кошку соответственно.

```
plt.figure(figsize=(6, 6))
for index, row in sample_test.iterrows():
    filename = row['filename']
    img = load_img("dataset/control_set/" + filename, target_size=(200, 200))
    test_image = image.load_img("dataset/control_set/" + filename,
                                target_size=(200, 200))
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = model.predict(test_image)
    if result[0][0] == 1:
        prediction = 'dog'
    else:
        prediction = 'cat'
    category = prediction
    axes = plt.subplot(6, 3, index + 1)
    plt.imshow(img)
    plt.xlabel(format(category))
    axes.set_yticks([])
    axes.set_xticks([])
plt.tight_layout()
plt.show()
```

2 Вычислительные эксперименты

1-й свёрточный слой	2-й свёрточный слой	1-й основной слой	2-й основной слой	3-й основной слой	время обучения , мин	точность обучения	точность тестиров ания
да	да	1	0	0	58	0.8348	0.8201
да	да	64	1	0	62	0.9281	0.8034
да	да	256	32	1	67	0.9324	0.7949
да	нет	1	0	0	39	0.7716	0.7576
да	нет	64	1	0	45	0.8758	0.7520
да	нет	256	32	1	60	0.5081	0.4995

3 Результат работы программы на своих данных



4 Вывод

В результате вычислительных экспериментов было обнаружено что оптимальной структурой для сети, классифицирующей изображения размером 128x128 точек, будет 4 слоя, включая входной и выходной. При удалении одного слоя резко уменьшается точность распознавания. При добавлении дополнительного слоя точность незначительно увеличивается, но намного больше вырастает время обучения.

Приложение А

Код программы

Файл `create_cnn.py`

```
# Importing the Keras libraries and packages
from keras.layers import Conv2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import MaxPooling2D
from keras.models import Sequential
from tensorflow import keras

# Initialising the CNN
model = Sequential()

# Step 1 - Convolution
model.add(Conv2D(32, (3, 3), input_shape=(128, 128, 3),
activation='relu'))

# Step 2 - Pooling
model.add(MaxPooling2D(pool_size=(2, 2)))

# Adding a second convolutional layer
# model.add(Conv2D(32, (3, 3), activation='relu'))
# model.add(MaxPooling2D(pool_size=(2, 2)))

# Step 3 - Flattening
model.add(Flatten())

# Step 4 - Full connection
# model.add(Dense(units=256, activation='relu'))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

# Compiling the CNN
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Part 2 - Fitting the CNN to the images
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1. / 255)

batch_size = 32
```

```

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                target_size=(128, 128),
                                                batch_size=batch_size,
                                                class_mode='binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size=(128, 128),
                                            batch_size=batch_size,
                                            class_mode='binary')

import os

cp_callback = keras.callbacks.ModelCheckpoint(filepath=os.curdir,
                                              save_weights_only=True,
                                              verbose=1)

model.fit(training_set,
          steps_per_epoch=8000 // batch_size,
          epochs=25,
          validation_data=test_set,
          validation_steps=2000 // batch_size,
          callbacks=[cp_callback])

model.save("keras_cnn")
print("Model saved")

```

Файл create_cnn.py

```

import os
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator, load_img

model = tf.keras.models.load_model("keras_cnn")
print("Model loaded")

train_datagen = ImageDataGenerator(rescale=1. / 255,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    horizontal_flip=True)

```

```

test_datagen = ImageDataGenerator(rescale=1. / 255)
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                target_size=(128, 128),
                                                batch_size=32,
                                                class_mode='binary')
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size=(128, 128),
                                            batch_size=32,
                                            class_mode='binary')

test_filenames = os.listdir("dataset/control_set")
test_df = pd.DataFrame({
    'filename': test_filenames
})
nb_samples = test_df.shape[0]

sample_test = test_df.head(18)
sample_test.head()

plt.figure(figsize=(6, 6))
for index, row in sample_test.iterrows():
    filename = row['filename']
    img = load_img("dataset/control_set/" + filename, target_size=(128,
128))

    test_image = image.load_img("dataset/control_set/" + filename,
target_size=(128, 128))
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = model.predict(test_image)
    if result[0][0] == 1:
        prediction = 'dog'
    else:
        prediction = 'cat'
    category = prediction
    axes = plt.subplot(6, 4, index + 1)
    plt.imshow(img)
    plt.xlabel(format(category))
    axes.set_yticks([])
    axes.set_xticks([])
plt.tight_layout()
plt.show()

```