

ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное учреждение

высшего профессионального образования

Московский технический университет связи и информатики

Кафедра «Сетевых информационных технологий и сервисов»

**Отчет по лабораторным работам по дисциплине**

**«ПОСИИ»**

**Выполнил:**

Студент 2-го курса

группы М091901(76)

Денисов Василий

Москва, 2020

Задание:

Лабораторная работа №1

Необходимо взять датасет, относящийся к одной из категорий:

1. Датасет, используемый в магистерской диссертации
2. Датасет с работы
3. Датасет с площадок для соревнований по машинному обучению (Например kaggle)
4. Сгенерированный датасет (Например, спарсенные картинки и т.д.)

Далее на имеющемся датасете обучается несколько моделей и замеряются метрики качества.

В ходе защиты необходимо:

1. Уметь в общих чертах пояснять работу используемых моделей
2. Уметь интерпретировать метрики
3. Уметь объяснить почему на имеющемся датасете решается конкретная задача (Например, классификация) задача машинного обучения
4. Понимать каким еще функционалом обладают используемые пакеты/бибилиотеки

Лабораторная работа №2

Необходимо разработать нечто напоминающее микросервисное приложение, которое собирается использованием docker-контейнеров.

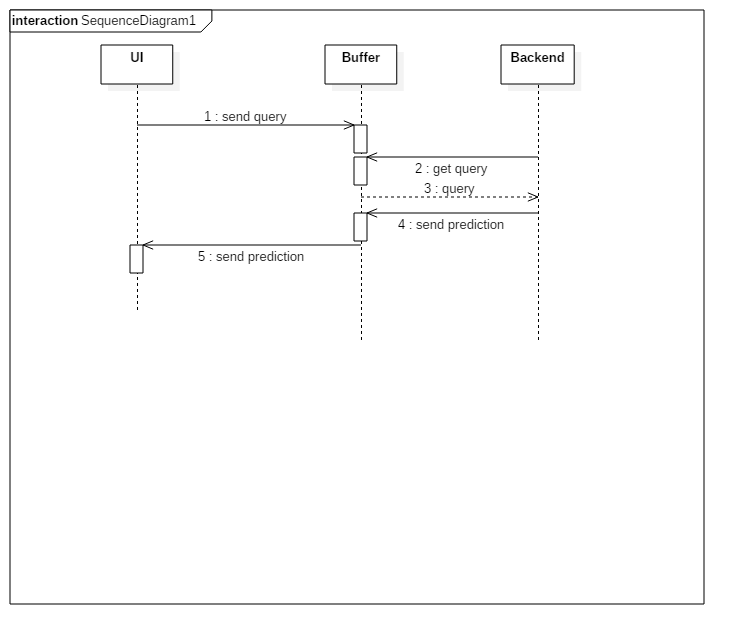


Рис.1 – концептуальная схема

На рисунке 1 представлена концептуальная схема работы приложения. Оно состоит из нескольких компонент:

1. UI – некоторый способ взаимодействия пользователя с системой. Это может быть как полноценный GUI, так и просто поле вводы наподобие командной строки
2. Buffer – хранилище данных. В качестве него может выступать любая реляционная БД (MySQL, PostgreSQL и т.д.), Key-Value хранилище (Redis), объектное хранилище (S3) и т.д. Можно так же использовать очереди сообщений.
3. Backend – программа, которая использует модель из ЛР1 для предсказания ответа на основе данных из UI

Каждый компонент представляет собой отдельный docker-образ. Сборка и запуск всех контейнеров может быть осуществлена как по отдельности, так и с использованием различных инструментов (Например, docker-compose)

В ходе защиты необходимо:

1. Продемонстрировать, что все запускается и на вводимые данные получен ответ
2. Понимать принцип работы docker
3. Понимать принцип работы используемых хранилищ данных
4. Иметь представление как работают используемые пакеты/библиотеки

Выполнение:

Распознавание объектов с помощью Keras c датасэтом Cifar-10

CIFAR-10 - это открытый набор данных для обучения с учителем. Он содержит 60 тыс. изображений объектов 10 классов, для каждого изображения указан правильный класс.

Для распознавания объектов из набора данных CIFAR-10 используется сверточная нейронная сеть. Сеть включает два каскада из слоев свертки и подвыборки (всего шесть слоев). Затем следует классификатор из двух полносвязных слоев.

Для предотвращения переобучения сверточной сети используется техника Dropout - отключение нейронов в процессе обучения с заданной вероятностью при каждом предъявлении объекта.

На вход подаются изображения размером 3Х32Х32 (rgb).

Загружается набор данных Cifar10 и распределяется по двум наборам.

Данные интенсивности изображений нормализуются от 0 до 1. Также метки классов преобразуются в категории.

Используемая модель: Sequential()

Техника для предотвращения переобучения: Dropout

Сеть является глубокой.

Сеть компилируется в методе model.compile()

Обучение в методе model.fit ()

Код:

import numpy  
from keras.datasets import cifar10  
from keras.models import Sequential  
from keras.layers import Dense, Flatten, Activation  
from keras.layers import Dropout  
from keras.layers.convolutional import Conv2D, MaxPooling2D  
from keras.utils import np\_utils  
from keras.optimizers import SGD  
  
# Задаем seed для повторяемости результатов  
numpy.random.seed(42)  
  
# Загружаем данные  
(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()  
# Размер мини-выборки  
batch\_size = 32  
# 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  
  
# Нормализуем данные  
X\_train = X\_train.astype('float32')  
X\_test = X\_test.astype('float32')  
X\_train /= 255  
X\_test /= 255  
  
# Преобразуем метки в категории  
Y\_train = np\_utils.to\_categorical(y\_train, nb\_classes)  
Y\_test = np\_utils.to\_categorical(y\_test, nb\_classes)  
  
# 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  
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'))  
  
# Задаем параметры оптимизации  
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)  
model.compile(loss='categorical\_crossentropy',  
 optimizer=sgd,  
 metrics=['accuracy'])  
# Обучаем модель  
model.fit(X\_train, Y\_train,  
 batch\_size=batch\_size,  
 epochs=nb\_epoch,  
 validation\_split=0.1,  
 shuffle=True,  
 verbose=2)  
  
# 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))

Docker:

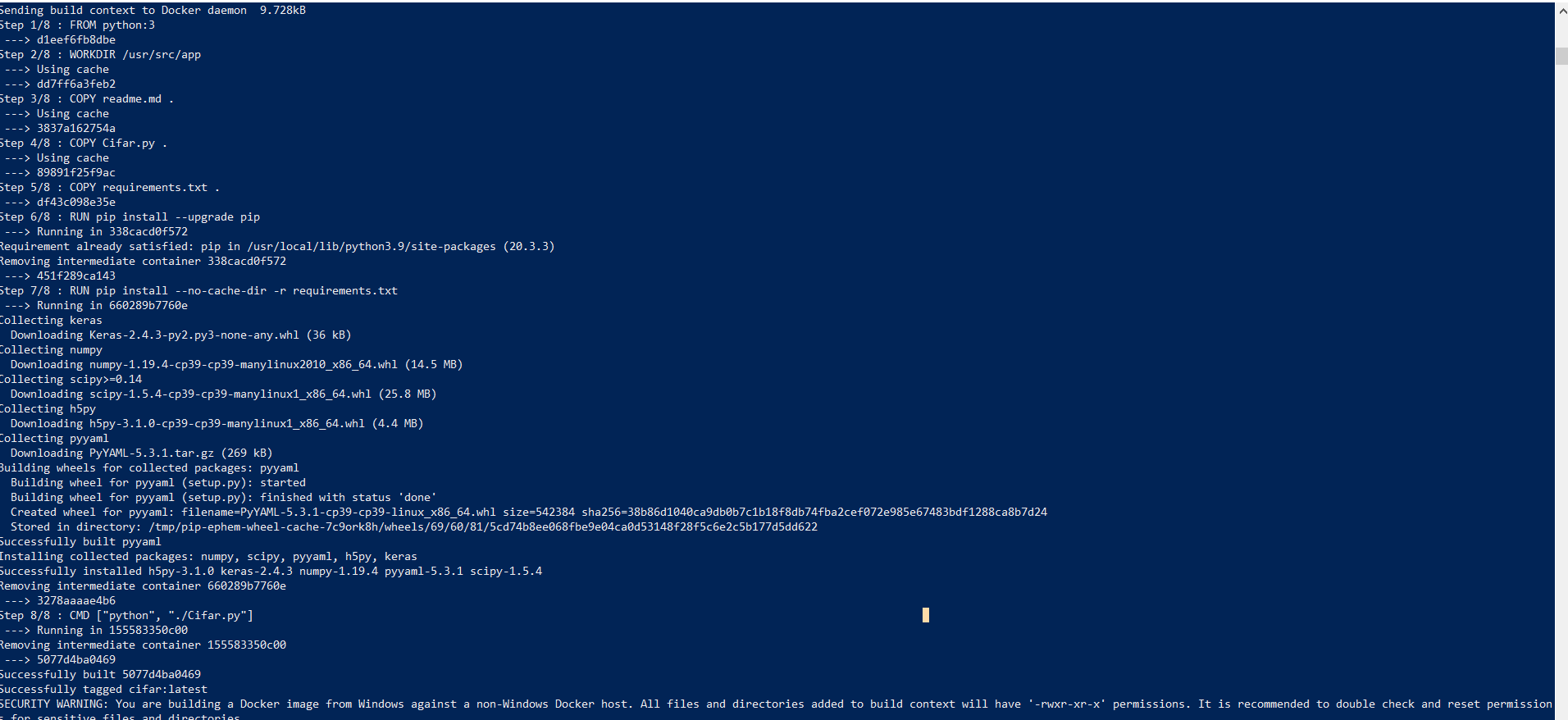
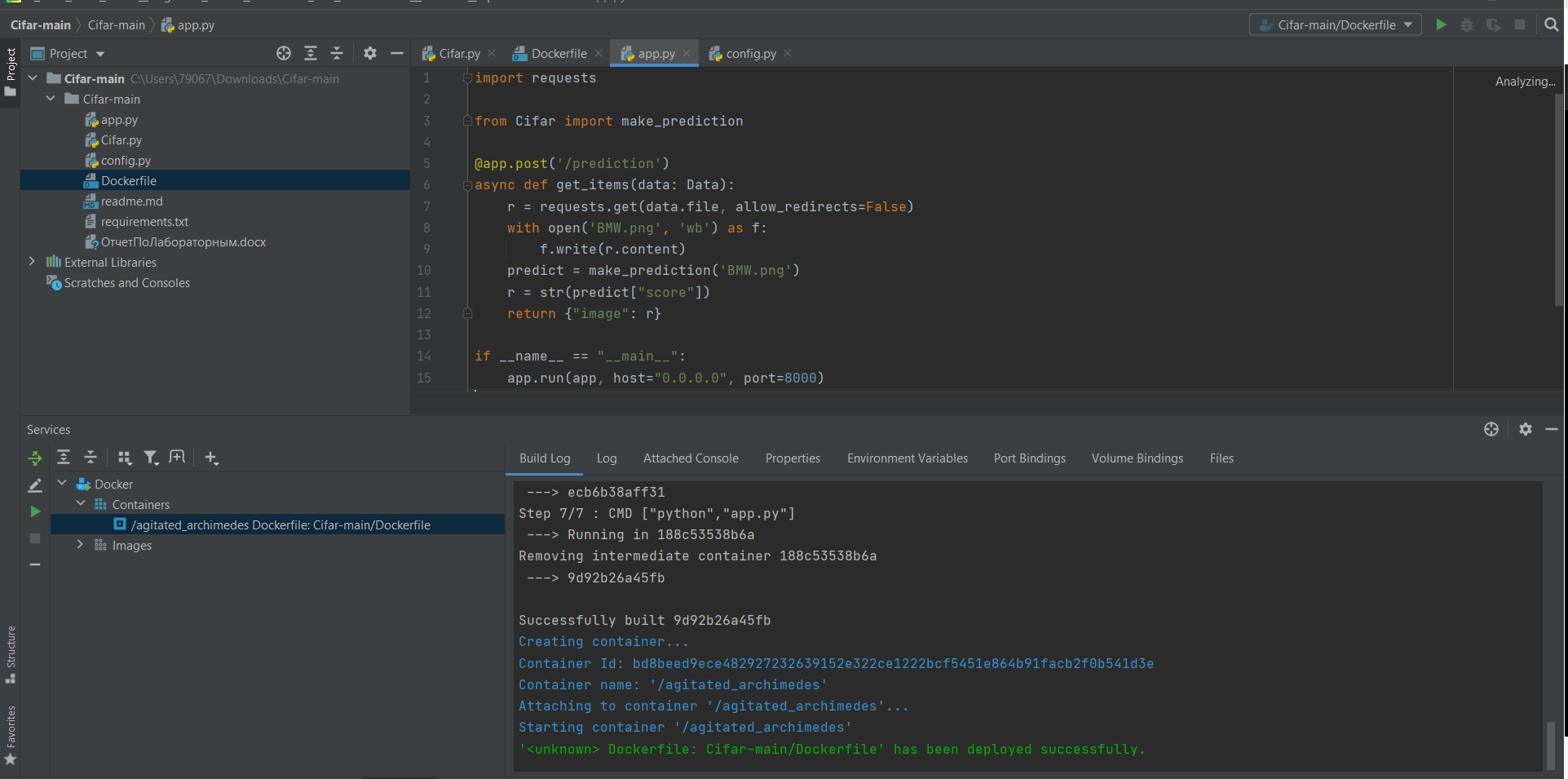


Рисунок 1 — загрузка образа



Docker