| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |
| --- | --- |

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**К ДОМАШНЕМУ ЗАДАНИЮ № 3**

**По дисциплине «Методы поддержки принятия решений»**

**МЕТОДЫ ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ НА ОСНОВЕ МЕТОДОВ ВЫЧИСЛИТЕЛЬНОГО ИНТЕЛЛЕКТА**

Студент ИУ5-71Б  Ю.Н.Рыжкова

(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель  А.Н. Нардид

(Подпись, дата) (И.О.Фамилия)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва – 2023

**Задание**

Необходимо создать web-приложение для классификации изображений с использованием предобученной модели на основе датасета cifar100. При выполнении задания Вы должны использовать 3-и класса из CIFAR100.

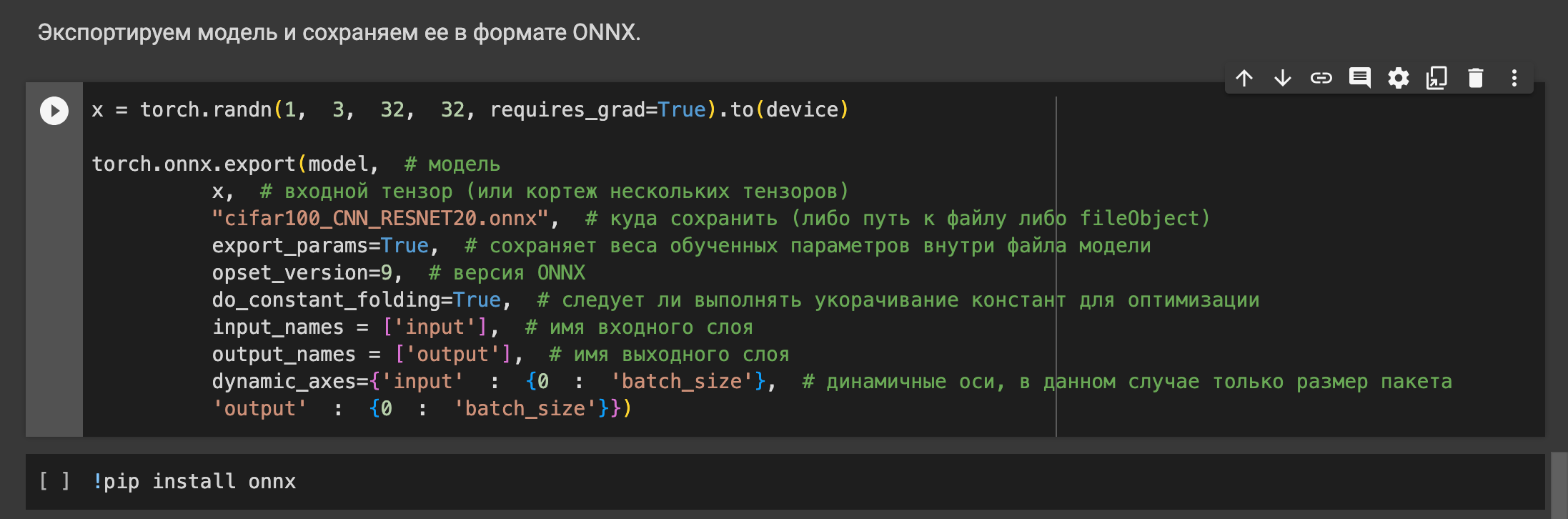
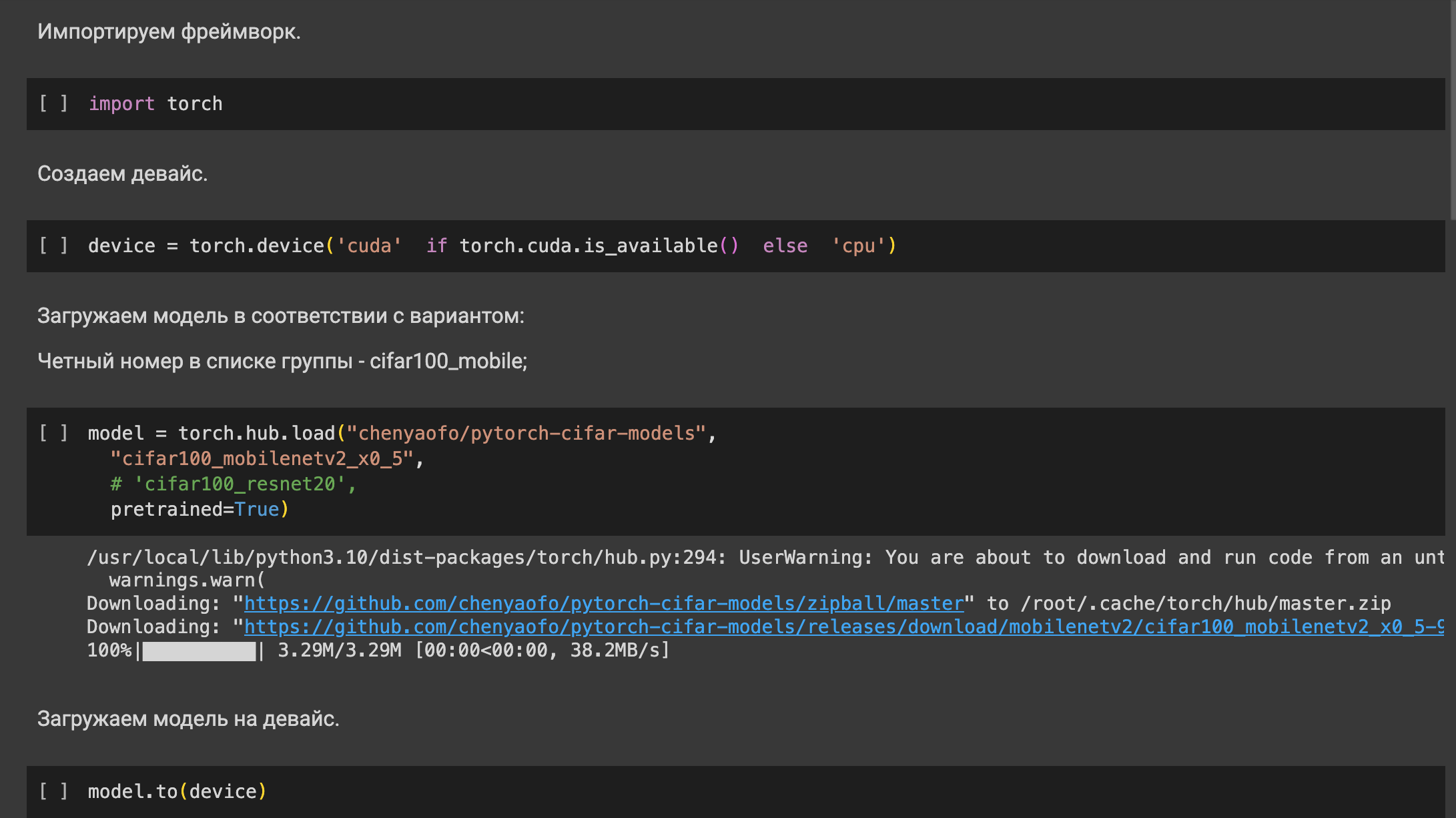
**Исходные данные:**

1. Класс 15: **верблюд**;
2. Класс 45: **лобстер**;
3. Класс 75: c**кунс**;
4. Модель: **cifar100\_resnet.**

**Выполнение работы**

**Часть 1. Сохранение модели в Google Colab**

Была загружена модель cifar100\_resnet и экспортирована в формат onnx.



**Часть 2. Web-приложение классификации изображений**

Был создан проект Django в IDE Pycharm. В корне проекта была создана папка “media” для последующего сохранения изображений и файлов формата ONNX. Внутри папки “media” для этого были созданы папки "images" и "models".

Изменения файла settings.py (добавлен путь в папке media):

import os

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))), 'media')

Файл views.py:

from django.shortcuts import render

from django.core.files.storage import FileSystemStorage

import onnxruntime

import numpy as np

from PIL import Image

from io import BytesIO

import base64

from torchvision import transforms

*# imageClassList = {0: 'apple', 1: 'aquarium\_fish', 2: 'baby', 3: 'bear', 4: 'beaver', 5: 'bed', 6: 'bee', 7: 'beetle',*

*# 8: 'bicycle', 9: 'bottle', 10: 'bowl', 11: 'boy', 12: 'bridge', 13: 'bus', 14: 'butterfly',*

*# 15: 'camel', 16: 'can', 17: 'castle', 18: 'caterpillar', 19: 'cattle', 20: 'chair', 21: 'chimpanzee',*

*# 22: 'clock', 23: 'cloud', 24: 'cockroach', 25: 'couch', 26: 'cra', 27: 'crocodile', 28: 'cup',*

*# 29: 'dinosaur', 30: 'dolphin', 31: 'elephant', 32: 'flatfish', 33: 'forest', 34: 'fox', 35: 'girl',*

*# 36: 'hamster', 37: 'house', 38: 'kangaroo', 39: 'keyboard', 40: 'lamp', 41: 'lawn\_mower',*

*# 42: 'leopard', 43: 'lion', 44: 'lizard', 45: 'lobster', 46: 'man', 47: 'maple\_tree', 48: 'motorcycle',*

*# 49: 'mountain', 50: 'mouse', 51: 'mushroom', 52: 'oak\_tree', 53: 'orange', 54: 'orchid', 55: 'otter',*

*# 56: 'palm\_tree', 57: 'pear', 58: 'pickup\_truck', 59: 'pine\_tree', 60: 'plain', 61: 'plate',*

*# 62: 'poppy', 63: 'porcupine', 64: 'possum', 65: 'rabbit', 66: 'raccoon', 67: 'ray', 68: 'road',*

*# 69: 'rocket', 70: 'rose', 71: 'sea', 72: 'seal', 73: 'shark', 74: 'shrew', 75: 'skunk',*

*# 76: 'skyscraper', 77: 'snail', 78: 'snake', 79: 'spider', 80: 'squirrel', 81: 'streetcar',*

*# 82: 'sunflower', 83: 'sweet\_pepper', 84: 'table', 85: 'tank', 86: 'telephone', 87: 'television',*

*# 88: 'tiger', 89: 'tractor', 90: 'train', 91: 'trout', 92: 'tulip', 93: 'turtle', 94: 'wardrobe',*

*# 95: 'whale', 96: 'willow\_tree', 97: 'wolf', 98: 'woman', 99: 'worm'} # Сюда указать классы*

imageClassList = {15: 'camel', 45: 'lobster', 75: 'skunk'}

def scoreImagePage(*request*):

return render(*request*, 'scorepage.html')

def predictImage(*request*):

fileObj = *request*.FILES['filePath']

fs = FileSystemStorage()

filePathName = fs.save('images/' + fileObj.name, fileObj)

filePathName = fs.url(filePathName)

modelName = *request*.POST.get('modelName')

scorePrediction, img\_uri = predictImageData(modelName, '.' + filePathName)

context = {'scorePrediction': scorePrediction, 'filePathName': filePathName, 'img\_uri': img\_uri}

return render(*request*, 'scorepage.html', context)

def predictImageData(*modelName*, *filePath*):

img = Image.open(*filePath*).convert("RGB")

resized\_img = img.resize((32, 32), Image.ANTIALIAS)

img\_uri = to\_data\_uri(resized\_img)

input\_image = Image.open(*filePath*)

preprocess = transforms.Compose([

transforms.Resize(32),

transforms.CenterCrop(32),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),

])

input\_tensor = preprocess(input\_image)

input\_batch = input\_tensor.unsqueeze(0)

sess = onnxruntime.InferenceSession(

r'/Users/artisia/PycharmProjects/MPPR\_DZ3/media/models/cifar100\_CNN\_RESNET20.onnx') *# <-Здесь требуется указать свой путь к модели*

outputOFModel = np.argmax(sess.run(None, {'input': to\_numpy(input\_batch)}))

score = imageClassList[outputOFModel]

return score, img\_uri

def to\_numpy(*tensor*):

return *tensor*.detach().cpu().numpy() if *tensor*.requires\_grad else *tensor*.cpu().numpy()

def to\_image(*numpy\_img*):

img = Image.fromarray(*numpy\_img*, 'RG')

return img

def to\_data\_uri(*pil\_img*):

data = BytesIO()

*pil\_img*.save(data, "JPEG") *# pick your format*

data64 = base64.b64encode(data.getvalue())

return u'data:img/jpeg;base64,' + data64.decode('utf-8')

Файл urls.py:

from django.contrib import admin

from django.urls import path

from django.conf.urls.static import static

from django.conf import settings

from . import views

urlpatterns = [

path('admin/', admin.site.urls),

path('', views.scoreImagePage, name='scoreImagePage'),

path('predictImage', views.predictImage, name='predictImage'),

]

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Файл scorepage.html:

<!DOCTYPE *html*>

<html *lang*="ru">

<head>

<meta *charset*="windows-1251">

<title>DZ1</title>

<style>

*/\* Add some padding on document's body to prevent the content*

*to go underneath the header and footer \*/* body{

*padding-top*: 60px;

*padding-bottom*: 40px;

}

.fixed-header, .fixed-footer{

*width*: 100%;

*position*: fixed;

*background*: #333;

*padding*: 10px 0;

*color*: #fff;

}

.fixed-header{

*top*: 0;

}

.fixed-footer{

*bottom*: 0;

}

.container{

*width*: 80%;

*margin*: 0 auto; */\* Center the DIV horizontally \*/*

}

nav a{

*color*: #fff;

*text-decoration*: none;

*padding*: 7px 25px;

*display*: inline-block;

}

</style>

</head>

<body>

<div *class*="fixed-header">

<div *class*="container">

</div> </div> <div *class*="container">

<form *action*="predictImage" *method*="post" *enctype*="multipart/form-data">

{% csrf\_token %}

<div *class*="col-md-4 col-sm-4">

<label *for*="FilePath">Select:</label>

</div> <input *name*="filePath" *type*="file"><br><br>

<input *type*="submit" *value*="Submit" >

</form> </div> <div> <br> {% if *scorePrediction* %}

<h3>The classification is : {{*scorePrediction*}}</h3>

{% endif %}

</div>

<div> {% if *scorePrediction* %}

<img *src*="{{ *img\_uri* }}">

{% endif %}

</div>

<div *class*="fixed-footer">

<div *class*="container"></div>

</div></body>

</html>

**Результаты работы приложения**

