Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»


# Отчет
# Домашняя работа № 1
# По курсу «Проектирование интеллектуальных систем»

Вариант 9

**ИСПОЛНИТЕЛЬ:**
Попов Илья Андреевич
Группа ИУ5-23М

_____

"__"_____2022 г.

**ПРЕПОДАВАТЕЛЬ:**
Канев А.И.

_____

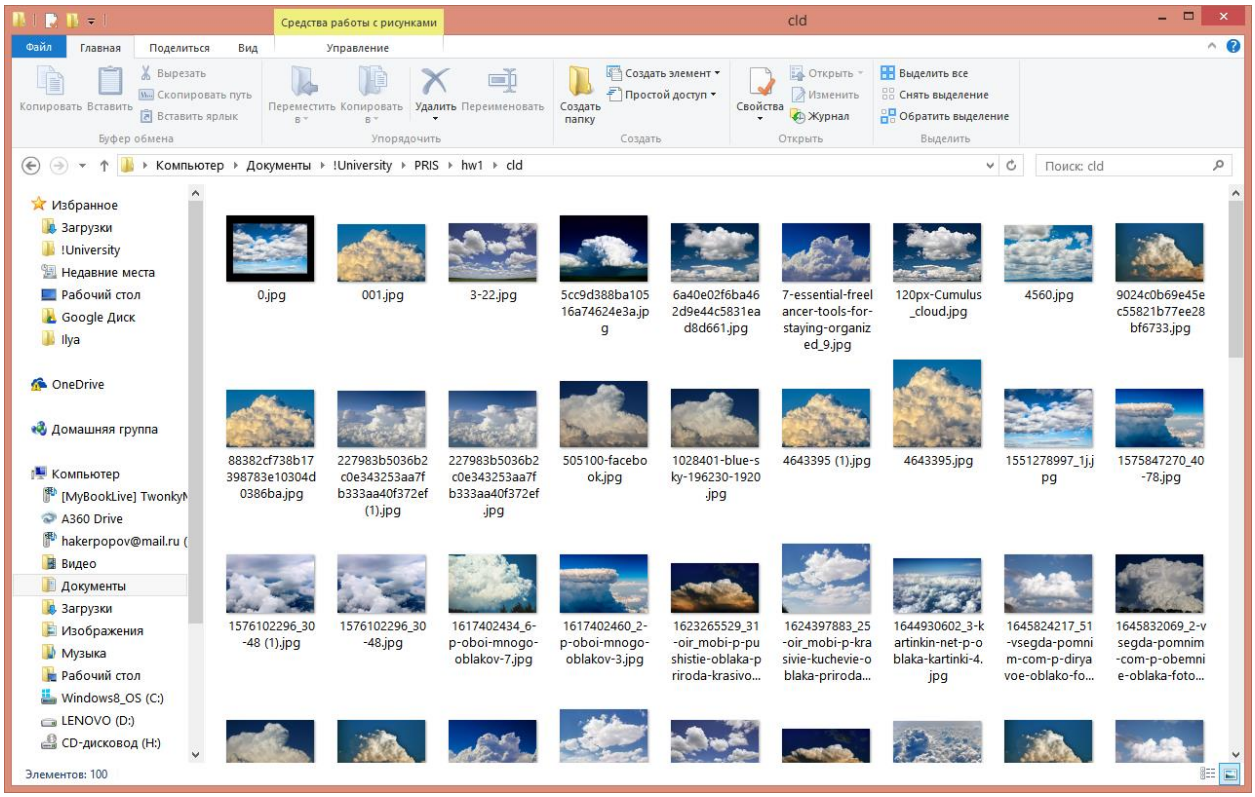
"__"_____2022 г.

Москва 2022

## Задание

Необходимо создать и разметить собственный набор данных, состоящий из изображений. Набор содержит не менее 3 классов и не менее 100 экземпляров каждый. Изображения можно скачать из интернета или объединить несколько существующих датасетов. Создать web-приложение для классификации изображений полученного набора данных. Использовать аугментацию данных, регуляризацию, перенос обучения.

## Выполнение




Загруженные фото (облака)

## Обучение модели

```
[ ]  heigth_width = 32

     CLASSES = ['bottle', 'cloud', 'keyboard'] # Здесь требуется указать ваши кла

     images = []
     images_t = []
     classes = []
     classes_t = []

     for CLASS in range(0, len(CLASSES)):
       path_class_1 = "/content/%s/*.*"%CLASSES[CLASS]
       i=0
       for photo in glob(path_class_1):
           i+=1
           img = Image.open(photo).convert('RGB')
           img = img.resize((heigth_width, heigth_width), Image.ANTIALIAS)
           if i > int(len(os.listdir("/content/%s/"%CLASSES[CLASS]))*0.8):
               images_t.append(np.asarray(img))
               classes_t.append(np.asarray(CLASS))
           else:
               images.append(np.asarray(img))
               classes.append(np.asarray(CLASS))


     train_X = np.array(images)
     train_y = np.array(classes)

     test_X = np.array(images_t)
     test_y = np.array(classes_t)
```

Чтение тестовой и тренировочной выборок

```
[ ]  class Normalize(nn.Module):
         def __init__(self, mean, std):
             super(Normalize, self).__init__()
             self.mean = torch.tensor(mean)
             self.std = torch.tensor(std)

         def forward(self, input):
             x = input / 255.0
             x = x - self.mean
             x = x / self.std
             return torch.flatten(x, start_dim=1) # nhwc -> nm
             #return x.permute(0, 3, 1, 2) # nhwc -> nm

     class Cifar100_MLP(nn.Module):
         def __init__(self, hidden_size=32, classes=100):
             super(Cifar100_MLP, self).__init__()
             # https://blog.jovian.ai/image-classification-of-cifar100-dataset-using-pytorch-8b7145242df1
             self.norm = Normalize([0.5074,0.4867,0.4411],[0.2011,0.1987,0.2025])
             self.seq = nn.Sequential(
                 #self.norm,
                 nn.Linear(32*32*3, HIDDEN_SIZE),
                 #nn.ReLU(),
                 #nn.Linear(HIDDEN_SIZE*2, HIDDEN_SIZE),
                 nn.ReLU(),
                 nn.Linear(hidden_size, classes),
                 #nn.Conv2d(3, HIDDEN_SIZE, 3, stride=4),
                 #nn.ReLU(),
                 #nn.Dropout2d(p=0.2),
                 # второй способ уменьшения размерности картинки - через слой пуллинг
                 #nn.Conv2d(HIDDEN_SIZE, HIDDEN_SIZE*2, 3, stride=1, padding=1),
                 #nn.ReLU(),
                 #nn.AvgPool2d(4),#nn.MaxPool2d(4),
                 #nn.Dropout2d(p=0.4),
                 #nn.Flatten(),
                 #nn.Linear(HIDDEN_SIZE*8, classes),
             )

         def forward(self, input):
             x = self.norm(input)
```

Создание модели (полносвязная сеть с одним скрытым слоем)

```python
EPOCHS = 250
steps_per_epoch = len(dataloader['train'])
steps_per_epoch_val = len(dataloader['test'])
for epoch in range(EPOCHS):  # проход по набору данных несколько раз
    running_loss = 0.0
    model.train()
    for i, batch in enumerate(dataloader['train'], 0):
        # получение одного минибатча; batch это двуэлементный список из [inputs, labels]
        inputs, labels = batch

        # очищение прошлых градиентов с прошлой итерации
        optimizer.zero_grad()

        # прямой + обратный проходы + оптимизация
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        #loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()

        # для подсчёта статистик
        running_loss += loss.item()
    print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / steps_per_epoch:.3f}')
    running_loss = 0.0
    model.eval()
    with torch.no_grad(): # отключение автоматического дифференцирования
        for i, data in enumerate(dataloader['test'], 0):
            inputs, labels = data

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.item()
    print(f'[{epoch + 1}, {i + 1:5d}] val loss: {running_loss / steps_per_epoch_val:.3f}')
print('Обучение закончено')
```

```
[1,    11] loss: 0.576
[1,     3] val loss: 0.520
[2,    11] loss: 0.360
[2,     3] val loss: 0.589
```

Обучение модели

```
[ ]     with torch.no_grad(): # отключение автоматического дифференцирования
            for i, data in enumerate(dataloader[part], 0):
                inputs, labels = data

                outputs = model(inputs).detach().numpy()
                y_pred.append(outputs)
                y_true.append(labels.numpy())
            y_true = np.concatenate(y_true)
            y_pred = np.concatenate(y_pred)
            print(part)
            print(classification_report(y_true.argmax(axis=-1), y_pred.argmax(axis=-1),
                                        digits=4, target_names=list(map(str, CLASSES))))
            print('-'*50)
```

```
train
              precision    recall  f1-score   support

      bottle     0.9895    0.9792    0.9843        96
       cloud     1.0000    1.0000    1.0000        80
    keyboard     0.9880    0.9940    0.9910       166

    accuracy                         0.9912       342
   macro avg     0.9925    0.9910    0.9918       342
weighted avg     0.9912    0.9912    0.9912       342

--------------------------------------------------
test
              precision    recall  f1-score   support

      bottle     0.9091    0.8333    0.8696        24
       cloud     0.9474    0.9000    0.9231        20
    keyboard     0.9333    1.0000    0.9655        42

    accuracy                         0.9302        86
   macro avg     0.9299    0.9111    0.9194        86
weighted avg     0.9298    0.9302    0.9289        86
```

Результат обучения

## Создание web-приложения для классификации изображений полученного набора данных

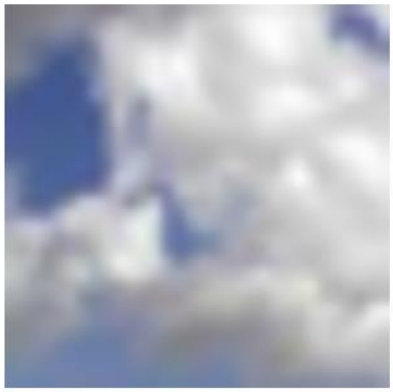
```python
from django.shortcuts import render
from django.core.files.storage import FileSystemStorage
import onnxruntime
import numpy as np
from PIL import Image

imageClassList = {'0': 'Бутылка', '1': 'Облако', '2': 'Клавиатура'}  #Сюда указать классы

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 = predictImageData(modelName, '.'+filePathName)
    context = {'scorePrediction': scorePrediction, 'img': filePathName}
    return render(request, 'scorepage.html', context)

def predictImageData(modelName, filePath):
    img = Image.open(filePath).convert("RGB")
    img = np.asarray(img.resize((32, 32), Image.ANTIALIAS))
    sess = onnxruntime.InferenceSession(r'C:\Users\Ilya\Documents\1University\PRIS\hw1\media\models\cifar100_CNN_MOBILENET2_1.onnx') #
    outputOFModel = np.argmax(sess.run(None, {'input': np.asarray([img]).astype(np.float32)}))
    score = imageClassList[str(outputOFModel)]
    return score
```

Загрузка и распознавание фотографий

Загрузить изображение:
Выберите файл Файл не выбран

Submit

**На картинке Клавиатура**



Загрузить изображение:
Выберите файл Файл не выбран

Submit

**На картинке Облако**

Загрузить изображение:
Выберите файл Файл не выбран

Submit

**На картинке Бутылка**

Результат работы приложения