Министерство науки и высшего образования Российской Федерации
государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

Проект «Классификация рукописных образов»

Руководитель СКБ Е.П. Жарикова

Подпись/дата

Ответственный исполнитель Н.В. Атюков

Подпись/дата

2021

Карточка проекта

Название	Проект «Классификация рукописных образов»
Тип проекта	Инициативный (инициативный, по заказу, в рамках конкурса, учебная работа, другое)
Исполнители	Н.В. Атюков ответственный исполнитель
Срок реализации	06.2021 Месяц, год

Использованные программные средства

Наименование	**Версия**
Pytorch	**1.9.0**
Windows	**10**

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

З А Д А Н И Е
на разработку

Выдано студентам:
Атюкову Никите Вячеславовичу
Название проекта:
Классификация рукописных образов
Назначение: Программное обеспечение предназначено для классификации рукописных образов.
Применение: Программное обеспечение может быть применено для распознания образов в задачах дополненной реальности.

План работ:

Этап	Дата начала	Дата окончания
Формирование требований к программному обеспечению	01.06.2021	03.06.2021
Разработка структуры программного обеспечения	03.06.2021	07.06.2021
Разработка и утверждение технического задания	01.06.2021	03.06.2021
Программная реализация	07.06.2021	19.06.2021
Тестирование и отладка системы	19.06.2021	23.06.2021
Подготовка документации	23.06.2021	25.06.2021
Опытная эксплуатация	25.06.2021	27.06.2021

Руководитель СКБ Е.П. Жарикова

Подпись/дата

Министерство науки и высшего образования Российской Федерации
государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

Руководство администратора
Проект «Классификация рукописных образов»

Руководитель СКБ Е.П. Жарикова

Подпись/дата

Ответственный исполнитель Н.В. Атюков

Подпись/дата

2021

Содержание

1 Общие положения 6
1.1 Наименование программы 6
1.2 Наименование документов, на основании которых ведется проектирование системы 6
1.3 Перечень организаций, участвующих в разработке системы 6
1.4 Сведения об использованных при проектировании нормативнотехнических документах 7
2 Назначение и принцип действия 8
2.1 Назначение изделия 8
Программное обеспечение для распознавания рукописных образов 8
2.2 Области использования изделия 8
2.3 Принцип действия 8
3. Описание программного обеспечения 8
3.1 Описание работы программного обеспечения 8
3.2 Входные данные 10
3.3 Выходные данные 11
4. Текст программы 12

1 Общие положения

Настоящий документ представляет собой руководство администратора программного обеспечения «Классификация рукописных образов» далее ПО. Руководство определяет порядок установки, настройки и администрирования ПО. Перед установкой и эксплуатацией системы рекомендуется внимательно ознакомиться с настоящим руководством. Документ подготовлен в соответствии с РД 50-34.698-90 - в части структуры и содержания документов, и в соответствии с ГОСТ 34.201-89 - в части наименования и обозначения документов.

1.1 Наименование программы

Наименование программного продукта: программное обеспечение для классификации рукописных образов.

1.2 Наименование документов, на основании которых ведется проектирование системы

Создание ПО осуществляется на основании требований и положений следующих документов: задание.

1.3 Перечень организаций, участвующих в разработке системы

Разработчики: студент Комсомольского-на-Амуре государственного технического университета Н.В. Атюков.

Заказчик: Комсомольский-на-Амуре государственный университет, студенческое конструкторское бюро «Интеллектуальные технологии».

1.4 Сведения об использованных при проектировании нормативно-технических документах

При проектировании использованы следующие нормативно-технические документы:

- ГОСТ 19.101-77 – виды программ и программных документов.

- ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом.

- ГОСТ 19.201-78 Техническое задание, требования к содержанию и оформлению;

- ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению.

- ГОСТ 2.004-88. Единая система конструкторской документации. Общие требования к выполнению конструкторских технологических документов на печатающих и графических устройствах вывода ЭВМ.

2 Назначение и принцип действия

2.1 Назначение изделия

Программное обеспечение для распознавания рукописных образов

2.2 Области использования изделия

Программное обеспечение может быть применено для распознания образов в задачах дополненной реальности.

2.3 Принцип действия

Программное обеспечение классифицирует рукописные образы

3. Описание программного обеспечения

3.1 Описание работы программного обеспечения

Данные поступают с камеры. Изображения подаются в виде контуров объектов. Контуры рисуются на листе бумаги (Рисунок 1).

```
from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))
  display(Image(filename))
except Exception as err:
  print(str(err))
```

```
Saved to photo.jpg
```

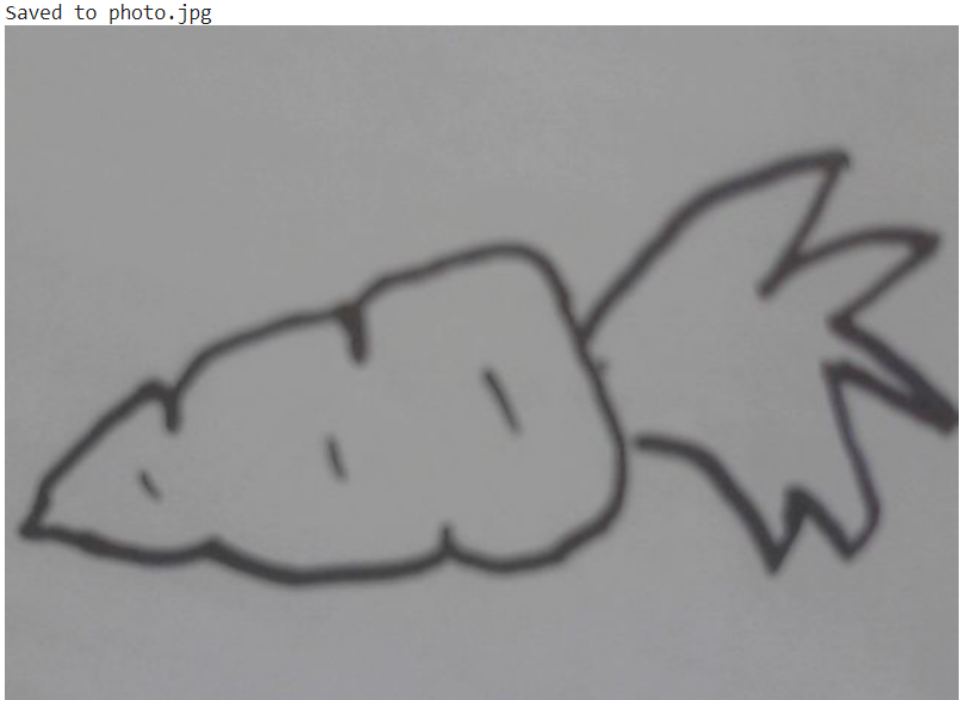

Рисунок 1 – Работа с камерой

Изображение переводится в бинарный вид, изменяется до размеров 224x224x1, преобразовывается в тензорный вид, подается на вход нейронной сети. Результаты работы представлены рисунках 2 и 3.

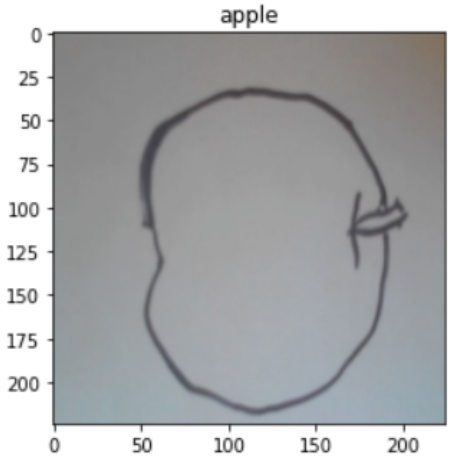

Рисунок 2– Классификация изображения

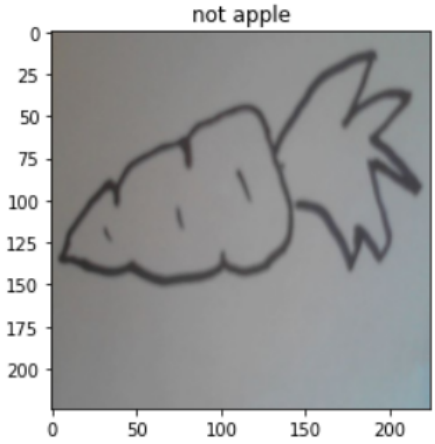

Рисунок 3 – Результат работы программы

Схема работы программного обеспечения представлена на рисунке 4.

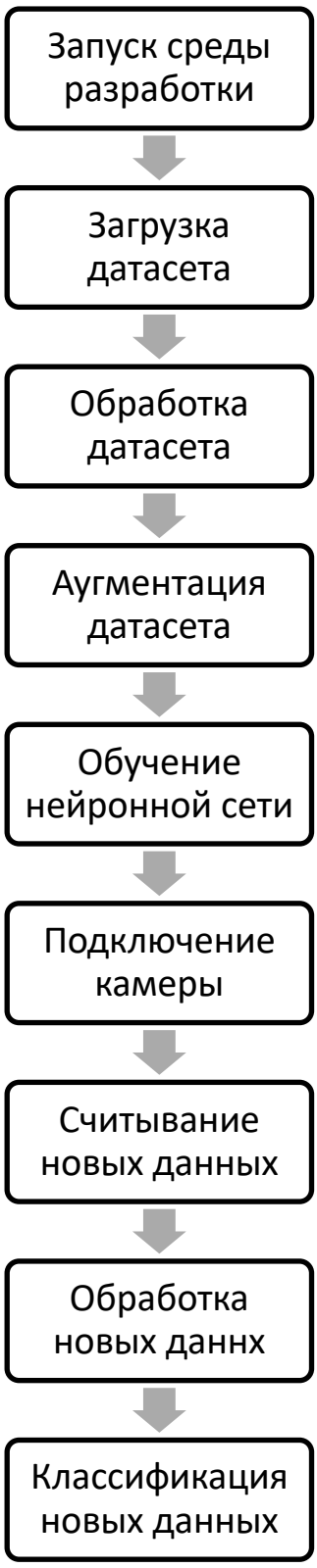

Рисунок 4 – Схема работы программы

3.2 Входные данные

Поток с камеры

3.3 Выходные данные

Выходными данными программного обеспечения является класс поданного изображения.

4. Текст программы

Реализация представлена в листинге 1.

Листинг 1 – «Программа запуска системы»

```
import random
import numpy as np
random.seed(0)
np.random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.backends.cudnn.deterministic = True
from google.colab import files
uploaded = files.upload()
import zipfile
!unzip Нейро.zip -d Нейро
import torchvision
from torchvision import transforms
train_transforms_1 = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
train_transforms_2 = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=1.0),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

```
train_transforms_3 = transforms.Compose([
    transforms.Resize((224, 224)),
    torchvision.transforms.RandomVerticalFlip(p=1.0),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
train_transforms_4 = transforms.Compose([
    transforms.Resize((224, 224)),
    torchvision.transforms.RandomRotation(degrees=90),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
train_transforms_5 = transforms.Compose([
    transforms.Resize((224, 224)),
    torchvision.transforms.RandomRotation(degrees=180),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
train_transforms_6 = transforms.Compose([
    transforms.Resize((224, 224)),
    torchvision.transforms.RandomRotation(degrees=270),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
val_transforms_1 = transforms.Compose([
    transforms.Resize((224, 224)),
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
val_transforms_2 = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=1.0),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
val_transforms_3 = transforms.Compose([
    transforms.Resize((224, 224)),
    torchvision.transforms.RandomVerticalFlip(p=1.0),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
val_transforms_4 = transforms.Compose([
    transforms.Resize((224, 224)),
    torchvision.transforms.RandomRotation(degrees=90),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
val_transforms_5 = transforms.Compose([
    transforms.Resize((224, 224)),
    torchvision.transforms.RandomRotation(degrees=180),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
```

```
                    std=[0.229, 0.224, 0.225])
])
val_transforms_6 = transforms.Compose([
    transforms.Resize((224, 224)),
    torchvision.transforms.RandomRotation(degrees=270),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                    std=[0.229, 0.224, 0.225])
])
train_dir = '/content/Нейро/Нейро/train'
val_dir = '/content/Нейро/Нейро/val'
train_dataset_1 = torchvision.datasets.ImageFolder(train_dir, train_transforms_1)
train_dataset_2 = torchvision.datasets.ImageFolder(train_dir, train_transforms_2)
train_dataset_3 = torchvision.datasets.ImageFolder(train_dir, train_transforms_3)
train_dataset_4 = torchvision.datasets.ImageFolder(train_dir, train_transforms_4)
train_dataset_5 = torchvision.datasets.ImageFolder(train_dir, train_transforms_5)
train_dataset_6 = torchvision.datasets.ImageFolder(train_dir, train_transforms_6)
val_dataset_1 = torchvision.datasets.ImageFolder(val_dir, val_transforms_1)
val_dataset_2 = torchvision.datasets.ImageFolder(val_dir, val_transforms_2)
val_dataset_3 = torchvision.datasets.ImageFolder(val_dir, val_transforms_3)
val_dataset_4 = torchvision.datasets.ImageFolder(val_dir, val_transforms_4)
val_dataset_5 = torchvision.datasets.ImageFolder(val_dir, val_transforms_5)
val_dataset_6 = torchvision.datasets.ImageFolder(val_dir, val_transforms_6)
train_dataset = torch.utils.data.ConcatDataset([train_dataset_1, train_da-
taset_2, train_dataset_3, train_dataset_4, train_dataset_5, train_dataset_6])
val_dataset = torch.utils.data.ConcatDataset([val_dataset_1, val_dataset_2, val_da-
taset_3, val_dataset_4, val_dataset_5, val_dataset_6])
len(train_dataset), len(val_dataset)
batch_size = 8
train_dataloader = torch.utils.data.DataLoader(
```

```
    train_dataset, batch_size=batch_size, shuffle=True, num_workers=batch_size)
val_dataloader = torch.utils.data.DataLoader(
    val_dataset, batch_size=batch_size, shuffle=False, num_workers=batch_size)
len(train_dataloader), len(val_dataloader)
from torchvision import models
def train_model(model, loss, optimizer, scheduler, num_epochs):
    for epoch in range(num_epochs):
        print('Epoch {}/{}:'.format(epoch+1, num_epochs), flush=True)
        for phase in ['train', 'val']:
            if phase == 'train':
                dataloader = train_dataloader
                scheduler.step()
                model.train()
            else:
                dataloader = val_dataloader
                model.eval()
            running_loss = 0.
            running_acc = 0.
            for inputs, labels in tqdm(dataloader):
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()
                with torch.set_grad_enabled(phase == 'train'):
                    preds = model(inputs)
                    loss_value = loss(preds, labels)
                    preds_class = preds.argmax(dim=1)
                    if phase == 'train':
                        loss_value.backward()
                        optimizer.step()
                running_loss += loss_value.item()
```

```
            running_acc += (preds_class == labels.data).float().mean()
        epoch_loss = running_loss / len(dataloader)
        epoch_acc = running_acc / len(dataloader)
        print('{} Loss: {:.3f} Acc: {:.3f}'.for-
mat(phase, epoch_loss, epoch_acc), flush=True)
  return model
model = models.resnet152(pretrained=True, progress=True)
for param in model.parameters():
  param.requires_grad = False
model.fc = torch.nn.Linear(model.fc.in_features, 2)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1.0e-3)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=4, gamma=0.1)
import shutil
from tqdm import tqdm
train_model(model, loss, optimizer, scheduler, num_epochs=10);
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
def take_photo(filename='photo.jpg', quality=0.8):
 js = Javascript('''
  async function takePhoto(quality) {
    const div = document.createElement('div');
    const capture = document.createElement('button');
    capture.textContent = 'Capture';
    div.appendChild(capture);
    const video = document.createElement('video');
    video.style.display = 'block';
```

```
      const stream = await navigator.mediaDevices.getUserMedia({video: true});
      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();
      google.colab.output.setIframeHeight(document.documentElement.scroll-
Height, true);
      await new Promise((resolve) => capture.onclick = resolve);
      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename
from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))
  display(Image(filename))
except Exception as err:
  print(str(err))
```

```
import random
from PIL import Image, ImageDraw
image = Image.open("/content/photo.jpg")
draw = ImageDraw.Draw(image)
width = image.size[0]
height = image.size[1]
pix = image.load()
for i in range(width):
  for j in range(height):
    a = pix[i, j][0]
    b = pix[i, j][1]
    c = pix[i, j][2]
    S = a + b + c
    if (S > (((255 - 50) // 2) * 3)):
      a, b, c = 255, 255, 255
    else:
      a, b, c = 0, 0, 0
    draw.point((i, j), (a, b, c))
image.save("new_photo.jpg", "JPEG")
del draw
!mv photo.jpg /content/Нейро/Нейро/test/image
!mv new_photo.jpg /content/Нейро/Нейро/test/image
class ImageFolderWithPaths(torchvision.datasets.ImageFolder):
    def __getitem__(self, index):
        original_tuple = super(ImageFolderWithPaths, self).__getitem__(index)
        path = self.imgs[index][0]
        tuple_with_path = (original_tuple + (path,))
        return tuple_with_path
test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
test_dataset = ImageFolderWithPaths('/content/Нейро/Нейро/test', test_trans-
forms)
test_dataloader = torch.utils.data.DataLoader(
    test_dataset, batch_size=1, shuffle=False, num_workers=0)
model.eval()
test_predictions = []
test_img_paths = []
for inputs, labels, paths in tqdm(test_dataloader):
    inputs = inputs.to(device)
    labels = labels.to(device)
    with torch.set_grad_enabled(False):
        preds = model(inputs)
    test_predictions.append(
        torch.nn.functional.softmax(preds, dim=1)[:,0].data.cpu().numpy())
    test_img_paths.extend(paths)
import matplotlib.pyplot as plt
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])
def show_input(input_tensor, title=''):
    image = input_tensor.cpu().permute(1, 2, 0).numpy()
    image = std * image + mean
    plt.imshow(image.clip(0, 1))
    plt.title(title)
    plt.show()
    plt.pause(0.001)
for img, pred in zip(inputs, test_predictions):
```

```
    show_input(img, title=pred)
for img, pred in zip(inputs, test_predictions):
    if pred>0.5:
      answer='apple'
    else:
      answer='not apple'
    show_input(img, title=answer)
```