Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»(ВлГУ)

Кафедра информационных систем и программной инженерии

Лабораторная работа №5

по дисциплине: "Введение в искусственный интеллект"

Тема работы: Нейросетевое распознавание печатных символов.

Выполнил:

ст. гр. ПРИ-120

Богдан С. С.

Принял:

зав. каф. ИСПИ

Озерова М. И.

Владимир, 2024

*Цель работы***:**

Построение нейронных сетей на Python. Исследование возможностей распознавания печатных символов с помощью нейронных сетей.

*Практическое задание 1 (Вариант 5)***:**

1. Описание задания:
   1. Подготовить графические файлы эталонных образов для символов, заданных преподавателем.
   2. В среде MATLAB создать и обучить НС, предназначенную для распознавания печатных символов.
   3. Исследовать зависимость качества работы НС от:
      1. cтепени искажения символов (параметр p);
      2. числа нейронов в скрытом слое.
2. Задание выполнено на Python. Файл среды (рис. 1.2.1). Исходный код доступен по ссылке <https://github.com/vlsu-labs/iiai-labs>.

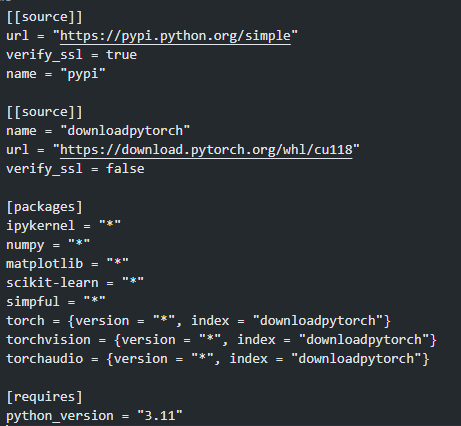


Рис. 1.2.1 Файл виртуального окружения.

1. Создание данных для обучения (рис. 1.3.1–1.3.2). И данные для тестирования.

Изображение выглядит как желтый, Шрифт, дизайн, типография

Автоматически созданное описание

Рис. 1.3.1 Изображения 16х16.

Изображение выглядит как дизайн, желтый

Автоматически созданное описание

Рис. 1.3.2 Изображения 64х64.

1. Подключим все необходимые компоненты (рис. 1.4.1). Объявим необходимые константы и классы (рис. 1.4.2–1.4.3).

import torch

from PIL import Image

import torch.nn as nn

import torch.optim as optim

import torchvision.transforms as transforms

from torch.utils.data import DataLoader

from torchvision.datasets import ImageFolder

from pathlib import Path

from typing import List, Dict

from os import listdir

from os.path import isfile, join

import re

Рис. 1.4.1 Подключенные пакеты.

IMAGE\_SIZE = 64

USED\_DIR\_SIZE = 64

LEAR\_DATA\_PATH = f'C:\my\study\IIAI\LB5\dataset\_root\{USED\_DIR\_SIZE}'

TEST\_DATA\_PATH = f'C:\my\study\IIAI\LB5\input\{USED\_DIR\_SIZE}'

MODEL\_INSTANCE\_PATH = f'model\_state\_{USED\_DIR\_SIZE}\_to\_{IMAGE\_SIZE}.pth'

CLASSES = ['А', 'Б', 'В', 'Г', 'Д', 'Е', 'Ё', 'Ж', 'З', 'И', 'Й', 'К', 'Л', 'М', 'Н', 'О', 'П', 'Р', 'С', 'Т', 'У', 'Ф', 'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ъ', 'Ы', 'Ь', 'Э', 'Ю', 'Я']

SIMPLE\_CLASSES = ['А', 'Б', 'В', 'Г', 'Д', 'Е', 'Ж', 'З', 'И', 'К', 'Л', 'М', 'Н', 'О', 'П', 'Р', 'С', 'Т', 'У', 'Ф', 'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ъ', 'Ы', 'Ь', 'Э', 'Ю', 'Я']

LETTERS = SIMPLE\_CLASSES if IMAGE\_SIZE == 16 else CLASSES

Рис. 1.4.2 Константы.

class ImageToModelFormatConverter:

def \_\_init\_\_(self, size):

self.transform = transforms.Compose([

transforms.Grayscale(),

transforms.Resize((size, size)),

transforms.ToTensor(),

])

def process\_single\_image(self, path: str):

image = Image.open(path)

return self.transform(image).unsqueeze(0) # Add batch dimension

def process\_folder(self, folder\_path: str):

return ImageFolder(root=folder\_path, transform=self.transform)

class FromMultipleTestDataSet:

def \_\_init\_\_(self, size):

self.converter = ImageToModelFormatConverter(size)

self.pattern = pattern = re.compile(r'(.+)\.(jpg|png)$', re.IGNORECASE)

def form\_dataset(self, root\_path: str, noise\_level: float = 0) -> Dict['str', Dict['str', List]]:

multiple\_data = {}

dirs = [f for f in listdir(root\_path) if not isfile(join(root\_path, f))]

for dir in dirs:

data = {}

path = join(root\_path, dir)

for file in listdir(path):

if (match := self.pattern.match(file)):

name = match.group(1)

if name not in data:

data[name] = []

data[name] = self.converter.process\_single\_image(join(path, file))

if noise\_level != 0:

noise = torch.randn\_like(data[name]) \* noise\_level

noisy\_image\_tensor = data[name] + noise

noisy\_image\_tensor = torch.clamp(noisy\_image\_tensor, 0, 1)

data[name] = noisy\_image\_tensor

multiple\_data[dir] = data

return multiple\_data

Рис. 1.4.3 Используемые классы.

1. Объявим модель и обучим её (рис. 1.5.1). Состояние модели сохраним в отдельных файлах (рис. 1.5.2).

class LetterRecognitionModel(nn.Module):

def \_\_init\_\_(self):

super(LetterRecognitionModel, self).\_\_init\_\_()

var = int(IMAGE\_SIZE / 4)

self.conv1 = nn.Conv2d(1, 16, kernel\_size=3, padding=1)

self.relu = nn.ReLU()

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.conv2 = nn.Conv2d(16, 32, kernel\_size=3, padding=1)

self.fc1 = nn.Linear(32 \* var \* var, 256)

self.fc2 = nn.Linear(256, len(LETTERS))

def forward(self, x):

var = int(IMAGE\_SIZE / 4)

x = self.pool(self.relu(self.conv1(x)))

x = self.pool(self.relu(self.conv2(x)))

x = x.view(-1, 32 \* var \* var)

x = self.relu(self.fc1(x))

x = self.fc2(x)

return x

converter = ImageToModelFormatConverter(IMAGE\_SIZE)

dataloader = DataLoader(converter.process\_folder(LEAR\_DATA\_PATH), batch\_size=IMAGE\_SIZE, shuffle=True)

model = LetterRecognitionModel()

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

iterations = 500

for iter in range(iterations):

for images, labels in dataloader:

optimizer.zero\_grad()

outputs = model(images)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

print(f'Iteration {iter+1}/{iterations}, Loss: {loss.item()}')

torch.save(model.state\_dict(), MODEL\_INSTANCE\_PATH)

Рис. 1.5.1 Обучение модели.

1. Теперь получим предсказания от модели с наложением некоторого шума поверх изображением (рис. 1.6.1) и проведём некоторый анализ (рис. 1.6.2–1.6.10).

from matplotlib.pyplot import subplots\_adjust

import matplotlib.pyplot as plt

if 'model' not in locals():

model = LetterRecognitionModel()

model.load\_state\_dict(torch.load(MODEL\_INSTANCE\_PATH))

model.eval()

tests\_loader = FromMultipleTestDataSet(IMAGE\_SIZE)

used\_data\_set = 'hard' if USED\_DIR\_SIZE != 16 else 'complex'

noises = np.arange(0, 0.7+0.1, 0.1)

prediction\_data = {}

for level in noises:

count = 0

match = 0

for letter, data in tests\_loader.form\_dataset(TEST\_DATA\_PATH, level)[used\_data\_set].items():

for test in data:

with torch.no\_grad():

output = model(test)

count += 1

predicted\_class = torch.argmax(output).item()

predicted\_class = torch.argmax(output).item()

predicted\_letter = LETTERS[predicted\_class]

match += 1 if predicted\_letter == letter else 0

print(f"Noise: {level} Actual: {letter} Predicted: {predicted\_letter} Check: {letter==predicted\_letter}")

prediction\_data[f'{level}'] = (match, count)

fig = plt.figure(num=1, facecolor='w', edgecolor='k')

fig.set\_figwidth(10)

fig.set\_figheight(5)

subplots\_adjust(hspace=0.3, wspace=0.3)

ax = fig.add\_subplot(1, 1, 1)

ax.plot(

prediction\_data.keys(),

list(map(lambda x: float(x[0])/x[1] ,prediction\_data.values())),

color='red',

linestyle='solid',

marker='o'

)

ax.set\_title(

f"Model accuracuty for '{used\_data\_set}' on image: {IMAGE\_SIZE}x{IMAGE\_SIZE}" \

if IMAGE\_SIZE == USED\_DIR\_SIZE else \

f"Model accuracuty for '{used\_data\_set}' on image: {IMAGE\_SIZE}x{IMAGE\_SIZE} (From {USED\_DIR\_SIZE}x{USED\_DIR\_SIZE})"

)

ax.set\_xlabel('noise level')

ax.set\_ylabel('model accuracy')

ax.grid()

i = 1

fig = plt.figure(num=2, facecolor='w', edgecolor='k')

fig.set\_figwidth(10)

fig.set\_figheight(20)

subplots\_adjust(hspace=-0.8, wspace=0.4)

for level in prediction\_data:

result = prediction\_data[level]

labels = 'Match', 'Not match'

sizes = [float(result[0])/result[1], float(result[1] - result[0])/result[1]]

colors = ['green', 'red']

explode = (0, 0)

ax = fig.add\_subplot(2, 4, i)

i += 1

ax.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)

ax.set\_title(f'Noise = {level}')

plt.show()

Рис. 1.6.1 Блок получения предсказания и анализа.

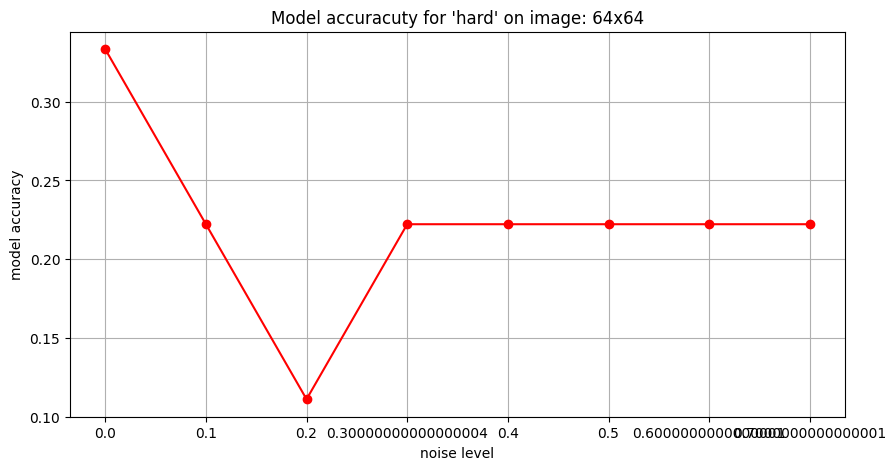


Рис. 1.6.2 Статистика предсказания модели для датасета 1 без зажатия.

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рис. 1.6.3 Статистика предсказания модели для датасета 2 без зажатия.

Изображение выглядит как текст, линия, График, число

Автоматически созданное описание

Рис. 1.6.4 Статистика предсказания модели для датасета 3 без зажатия.

Изображение выглядит как текст, График, линия, диаграмма

Автоматически созданное описание

Рис. 1.6.5 Статистика предсказания модели для датасета 3 со сжатием до 32.

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рис. 1.6.6 Статистика предсказания модели для датасета 2 со сжатием до 32.

Изображение выглядит как текст, линия, График, снимок экрана

Автоматически созданное описание

Рис. 1.6.7 Статистика предсказания модели для датасета 1 со сжатием до 32.

Изображение выглядит как линия, диаграмма, График, текст

Автоматически созданное описание

Рис. 1.6.8 Статистика предсказания модели для датасета 1 без увеличения.

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рис. 1.6.9 Статистика предсказания модели для датасета 1 с увеличением до 32.

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рис. 1.6.10 Статистика предсказания модели для датасета 1 с увеличением до 64.

*Вывод***:**

Изучены способы построения нейронных сетей на Python, и проведено исследование возможностей распознавания печатных символов с помощью нейронных сетей.