SAVITRIBAI PHULE PUNE UNIVERSITY

Zeal College Of Engineering And Research,
Pune2022-2023

DEPARTMENT OF AI&DS ENGINEERING

A PROJECT REPORT ON

Handwritten Digit Recognition Using Deep Learning

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE
FOR PROJECT BASED LEARNING (PBL)

BACHELOR OF ENGINEERING

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SUBMITTED BY

Shivam Awasare	S190532008
Shivam Bhansali	S190532010
Sudhanshu Dixit	S190532020
Shivani Gadhave	S190532021

CERTIFICATE

This is to certify that the project report entitled

Expenses Management System

Is Satisfactorily Completed By

Shivam Awasare	S1411004
Shivam Bhansali	S1411006
Sudhanshu Dixit	S1411015
Shivani Gadhave	S1411016

are bonafide students of this institute and the work has been carried out by them. It is approved for the partial fulfillment of the partial requirement of Savitribai Phule Pune University, for the award of the degree of Bachelor of Engineering (Artificial Intelligence and Data Science).

Prof. Shilpa Gaikwad
Guide

Prof. Dikshendra Sarpate
HOD

Dr. Ajit M. Kate
Principal

Date: 23/05/2023

Place: Pune

Acknowledgment

It gives us great pleasure in presenting the preliminary project report on "**Handwritten Digit Recognition Using Deep Learning**". I would like to take this opportunity to thank my internal guide **Prof. Shilpa Gaikwad** Department of Artificial Intelligence and Data Science, Zeal College of Engineering and Research. for his unconditional guidance. I am really grateful to him for his kind support. Their valuable suggestions were very helpful an erudite teacher, a magnificent person, and a strict disciplinarian, we consider our- selves fortunate to have worked under her supervision. We are thankful to Honorable Principal, Zeal College of Engineering and Research **Dr. Ajit M. Kate** for the support.

We are highly grateful to **Prof. Dikshendra Sarpate** Head of Department, Artificial Intelligence and Data Science, Zeal College of Engineering and Research for indispensable support, suggestions. We admit thanks to project coordinator, Department of Artificial Intelligence and Data Science, Zeal College of Engineering Research for giving us such an opportunity to carry on such mind stimulating and innovative Project.

Shivam Awasare	S1411004
Shivam Bhansali	S1411006
Sudhanshu Dixit	S1411015
Shivani Gadhave	S1411016

(SE Artificial Intelligence and Data Science)

INDEX

1 Introduction 5

1.1 Abstract 5

1.2 Problem Definition 5

1.3 Objective 5

2 Project Scope 6

3 System Design 6

3.1 Architecture 6

3.2 Database Design 6

3.3 User Interface Design 7

4 Implementation 7

5 User Interface 7

6 Result 7

7 Testing and Quality Assurance 8

8 Conclusion 8

9 References 9

10 Code 10

11 Output 14

1. Introduction

1.1 Abstract

This project develops a web application for image classification using Flask and PyTorch. The application allows users to upload images, which are processed and classified using a pre-trained model. Visualizations of the prediction results, including a probability bar chart and an interpretability image, are provided. The project showcases the integration of deep learning models into web applications and offers an intuitive user interface for interacting with image classification algorithms.

1.2 Problem Definition:

User-friendly web application that performs image classification using a pre-trained model. Existing solutions often lack intuitive interfaces and visualizations to help users understand the model's predictions. There is a demand for a web application that allows users to upload images, accurately classify them, and provide visual explanations of the prediction results. The challenge is to design and implement a web application using Flask and PyTorch that effectively handles image preprocessing, model inference, and visualization generation to deliver an interactive and informative user experience for image classification tasks.

1.3 Objective

a. Develop a user-friendly web application: The primary objective is to create a web application that is intuitive and easy to use, allowing users to upload images and obtain classification results effortlessly. The interface should provide clear instructions and feedback to guide users through the process.

b. Perform image classification using a pre-trained model: The application should utilize a pre-trained model, specifically designed for image classification tasks, to accurately classify uploaded images. The model should be capable of handling various image formats and sizes.

c. Implement image preprocessing: To ensure compatibility with the pre-trained model, the application should incorporate image preprocessing techniques such as converting images to grayscale, resizing them to a specific size (e.g., 28x28 pixels), and normalizing pixel values.

d. Generate visualizations of prediction results: In addition to the classification output, the application should generate visualizations to aid users in understanding the model's predictions. This may include generating

a probability bar chart that shows the predicted probabilities for each class and an interpretability image that highlights the model's decision-making process.

e. Provide interpretability and insights: The visualizations generated by the application should provide interpretability and insights into the model's classification process. The interpretability image, showcasing the output of intermediate convolutional layers, should allow users to understand how the model analyzes and processes the uploaded image.

f. Ensure scalability and performance: The application should be designed to handle multiple user requests simultaneously and exhibit reasonable response times. Efficient memory management and optimization techniques should be employed to ensure smooth and responsive user experience.

g. Enable extensibility and adaptability: The application should be designed in a modular and flexible manner, allowing for easy integration of different pre-trained models and datasets. It should support future enhancements and accommodate various image classification tasks beyond the MNIST dataset.

2. Project Scope

The expense tracker application will be developed as a web-based system accessible through popular web browsers. The system will be designed to cater to individual users as well as small to medium-sized organizations. It will focus on providing an easy-to-use interface, robust expense tracking capabilities, and insightful reports for effective financial management.

3. System Design

3.1 Architecture

The system architecture follows a three-tier model, consisting of a presentation layer, application layer, and database layer. The presentation layer handles user interactions, the application layer processes business logic, and the database layer stores and retrieves data.

3.2 Database Design

The database design includes tables for user information, expense records, categories, budgets, and other relevant entities. Proper relationships and normalization techniques are applied to ensure data integrity and efficiency.

3.3 User Interface Design

The user interface design focuses on simplicity and usability. It employs responsive design principles to ensure a consistent experience across different devices and screen sizes.

4. Implementation

- Data Preprocessing: When an image is uploaded, it is preprocessed to ensure compatibility with the model. The image is converted to grayscale, resized to 28x28 pixels, and converted to a NumPy array.
- Model Loading: The pre-trained model is loaded into memory. It is a custom implementation of the MNISTModel class, designed for image classification with 10 classes (digits 0-9).
- Model Inference: The preprocessed image is passed through the model to obtain the predicted class probabilities. The softmax function is applied to obtain a probability distribution over the classes.
- Visualization: Two types of visualizations are generated:

a) Probability Bar Chart: A bar chart showing the predicted probabilities for each class. It is generated using Matplotlib and saved as a PNG image.

b) Interpretability Image: An image grid showing the output of intermediate convolutional layers during the forward pass. This visualization provides insight into the model's decision-making process.

- Response Generation: The predicted digit, the encoded probability bar chart, and the encoded interpretability image are packaged into a JSON response and sent back to the user.

5. User Interface

The web application provides a simple and intuitive user interface. It consists of an upload button for selecting an image file and a submit button for processing the image.
Once the image is processed, the predicted digit, probability bar chart, and interpretability image are displayed on the webpage.

6. Results

The developed web application provides accurate image classification results using the pre-trained MNIST model. The probability bar chart allows users to visualize the model's confidence in its predictions for each class. The interpretability image provides a glimpse into the internal representations learned by the model, helping users understand how the model arrives at its predictions.

7. Testing and Quality Assurance

Comprehensive testing strategies are employed throughout the development process to ensurethe application's functionality, reliability, and security. The testing phases include unit testing,integration testing, system testing, and user acceptance testing. Code reviews and bug trackingsystems are utilized to maintain code quality and resolve issues promptly.

8. Conclusion

- This project successfully implements a web application for image classification using Flask and PyTorch. The application leverages a pre-trained model to classify images uploaded by users and provides visualizations of the prediction results.
- The web application can be extended to support other image classification tasks and different pre-trained models by modifying the model loading and inference steps.
- Overall, this project demonstrates the integration of deep learning models into web applications, enabling users to interact with and benefit from the capabilities of image classification algorithms.

9. References

1. Flask Documentation. (2021). Retrieved from: https://flask.palletsprojects.com/

2. PyTorch Documentation. (2021). Retrieved from: https://pytorch.org/docs/stable/index.html

3. Base64 Documentation. (2021). Retrieved from: https://docs.python.org/3/library/base64.html

4. Matplotlib Documentation. (2021). Retrieved from: https://matplotlib.org/stable/contents.htm

5. PIL (Python Imaging Library) Documentation. (2021). Retrieved from: https://pillow.readthedocs.io/en/stable/

6. Chollet, F. (2017). Deep learning with Python. Manning Publications.

7. MNIST handwritten digit database. (n.d.). Retrieved from http://yann.lecun.com/exdb/mnist/

8. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

CODE:

```
import torch
import base64
import config
import matplotlib
import numpy as np
from PIL import Image
from io import BytesIO
from train import MnistModel
import matplotlib.pyplot as plt
from flask import Flask, request, render_template, jsonify
matplotlib.use('Agg')


MODEL = None
DEVICE = torch.device('cuda:0' if torch.cuda.is_available()
else 'cpu')

app = Flask(__name__)


class SaveOutput:
    def __init__(self):
        self.outputs = []

    def __call__(self, module, module_in, module_out):
        self.outputs.append(module_out)

    def clear(self):
        self.outputs = []


def register_hook():
    save_output = SaveOutput()
    hook_handles = []

    for layer in MODEL.modules():
        if isinstance(layer, torch.nn.modules.conv.Conv2d):
            handle = layer.register_forward_hook(save_output)
```

```
            hook_handles.append(handle)
    return save_output


def module_output_to_numpy(tensor):
    return tensor.detach().to('cpu').numpy()


def autolabel(rects, ax):
    """Attach a text label above each bar in *rects*,
displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{0:.2f}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2,
height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


def prob_img(probs):
    fig, ax = plt.subplots()
    rects = ax.bar(range(len(probs)), probs)
    ax.set_xticks(range(len(probs)), (0, 1, 2, 3, 4, 5, 6, 7,
8, 9))
    ax.set_ylim(0, 110)
    ax.set_title('Probability % of Digit by Model')
    autolabel(rects, ax)
    probimg = BytesIO()
    fig.savefig(probimg, format='png')
    probencoded =
base64.b64encode(probimg.getvalue()).decode('utf-8')
    return probencoded


def interpretability_img(save_output):
    images = module_output_to_numpy(save_output.outputs[0])
    with plt.style.context("seaborn-white"):
        fig, _ = plt.subplots(figsize=(20, 20))
```

```
        plt.suptitle("Interpretability by Model", fontsize=50)
        for idx in range(16):
            plt.subplot(4, 4, idx+1)
            plt.imshow(images[0, idx])
        plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
    interpretimg = BytesIO()
    fig.savefig(interpretimg, format='png')
    interpretencoded = base64.b64encode(
        interpretimg.getvalue()).decode('utf-8')
    return interpretencoded


def mnist_prediction(img):
    save_output = register_hook()
    img = img.to(DEVICE, dtype=torch.float)
    outputs = MODEL(x=img)

    probs = torch.exp(outputs.data)[0] * 100
    probencoded = prob_img(probs)
    interpretencoded = interpretability_img(save_output)

    _, output = torch.max(outputs.data, 1)
    pred = module_output_to_numpy(output)
    return pred[0], probencoded, interpretencoded


@app.route("/process", methods=["GET", "POST"])
def process():
    data_url = str(request.get_data())
    offset = data_url.index(',')+1
    img_bytes = base64.b64decode(data_url[offset:])
    img = Image.open(BytesIO(img_bytes))
    img = img.convert('L')
    img = img.resize((28, 28))
    # img.save(r'templates\image.png')
    img = np.array(img)
    img = img.reshape((1, 28, 28))
    img = torch.tensor(img, dtype=torch.float).unsqueeze(0)
```

```
    data, probencoded, interpretencoded =
mnist_prediction(img)

    response = {
        'data': str(data),
        'probencoded': str(probencoded),
        'interpretencoded': str(interpretencoded),
    }
    return jsonify(response)


@app.route("/", methods=["GET", "POST"])
def start():
    return render_template("default.html")


if __name__ == "__main__":
    MODEL = MnistModel(classes=10)
    MODEL.load_state_dict(torch.load(
        'checkpoint/mnist.pt', map_location=DEVICE))
    MODEL.to(DEVICE)
    MODEL.eval()
    app.run(host=config.HOST, port=config.PORT,
debug=config.DEBUG_MODE)
```

OUTPUTS:

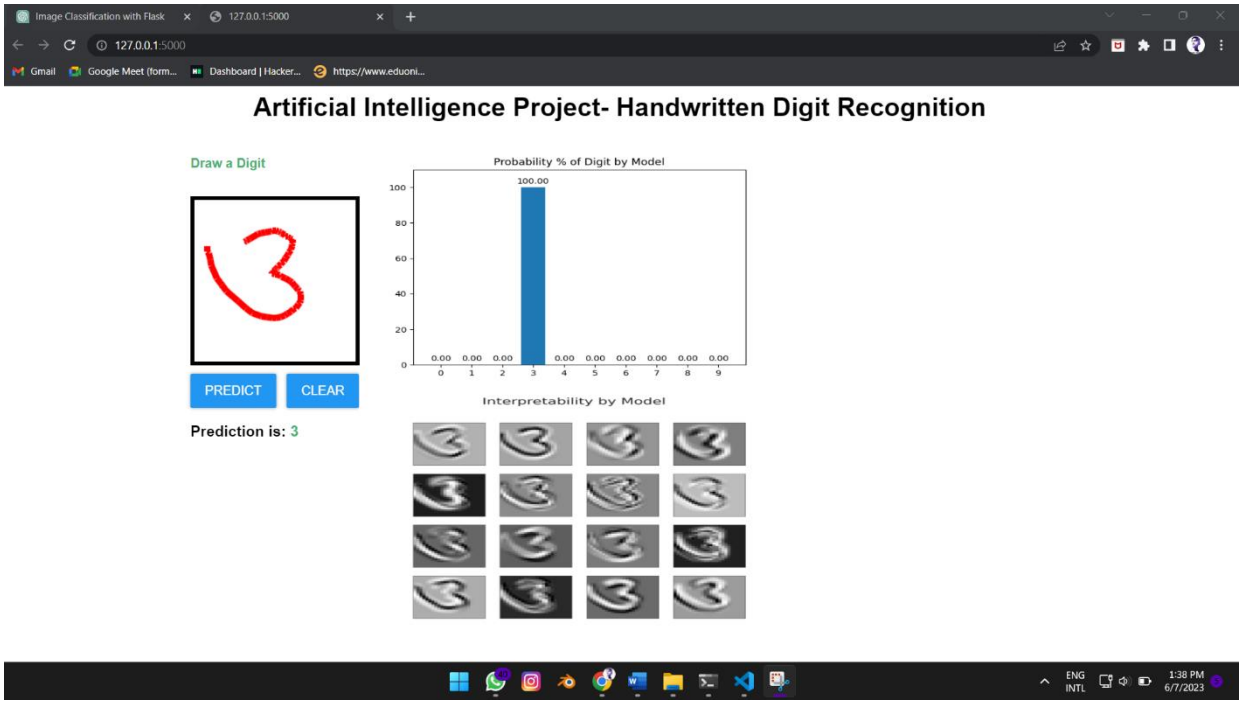
Image Classification with Flask
127.0.0.1:5000
Gmail
Google Meet (form...
Dashboard | Hacker...
https://www.eduoni...
Artificial Intelligence Project- Handwritten Digit Recognition
Draw a Digit
PREDICT
CLEAR
Prediction is: 3
Probability % of Digit by Model
100.00
Interpretability by Model
ENG
INTL
1:38 PM
6/7/2023

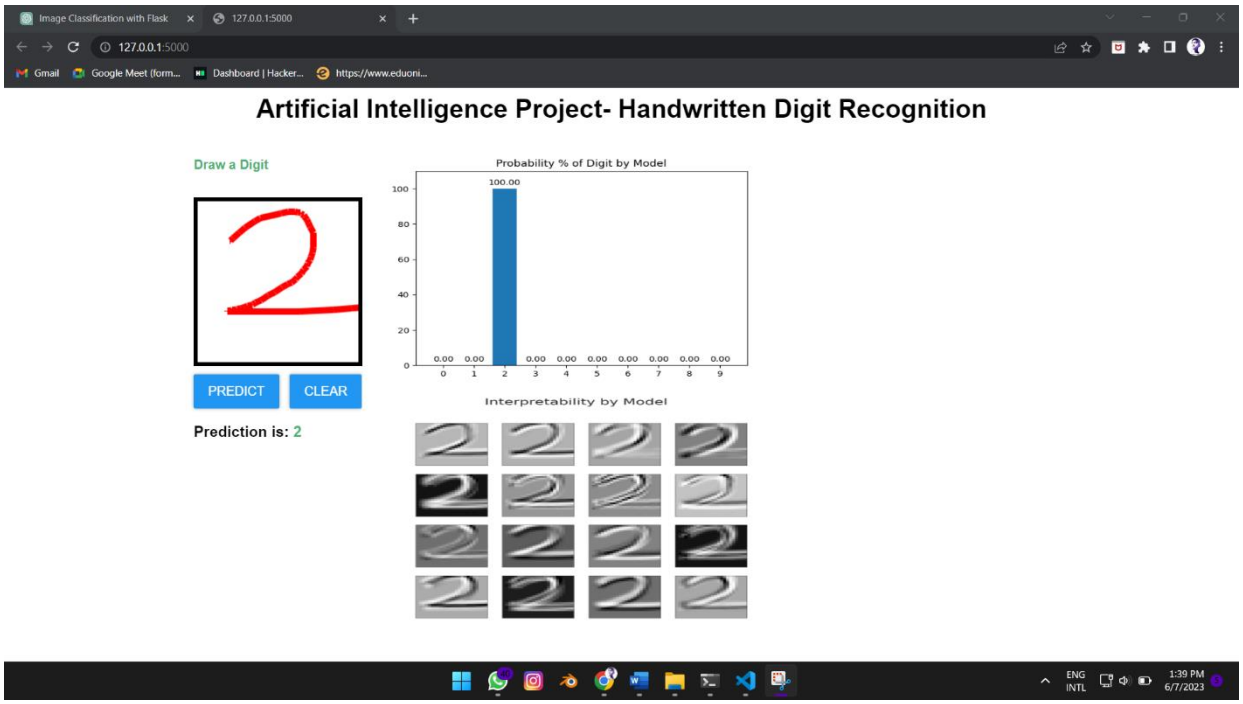
Image Classification with Flask
127.0.0.1:5000
Gmail
Google Meet (form...
Dashboard | Hacker...
https://www.eduoni...
Artificial Intelligence Project- Handwritten Digit Recognition
Draw a Digit
PREDICT
CLEAR
Prediction is: 2
Probability % of Digit by Model
100.00
Interpretability by Model
ENG
INTL
1:39 PM
6/7/2023