****

**Department of Data science and Statistics**

**COURSE: MDA- NEURAL NETWORK AND DEEP LEARNING**

**Submitted by,**

**SHIGHA VELLOTH**

**2282475**

**Submitted to,**

**Dr. DALWIN VINOTH KUMAR**

**Department of Data science and Statistics**

**Date of Submission**

**04/12/2023**

**Handwritten Digit Prediction**

**INTRODUCTION:**

In order to forecast the content of handwritten photos, we start working on a Handwritten Image Recognition System using machine learning. Training a convolutional neural network on a dataset, like MNIST, forms the basis of the system, and it is followed by a smooth user interface integration. Users can communicate with the machine learning model through three frameworks: Flask, Streamlit, and Gradio.

Gradio, renowned for its simplicity, provides a rapid deployment solution by enabling simple interface with pre-trained models. The creation of web-based interfaces is made easier with Streamlit, a framework for interactive data applications. A lightweight web application framework called Flask offers versatility in creating user interfaces with built-in machine learning features. Our goal with these frameworks is to build a user-friendly, responsive site where people can upload handwritten photos, get predictions in real time, and explore the nexus between AI and human interaction.

HANDWRITTEN IMAGES

The technique of giving machines the capacity to recognise handwritten numbers by humans is known as handwritten digit recognition. Because handwritten numbers are imperfect, differ from person to person, and come in a wide variety of flavours, it is a difficult work for the machine.

machine learning or deep learning model, as well as any function. It takes only a few lines of code to create a gradio user interface. The installation instructions and basic gradio examples can be found at the previously stated link. Gradio may be used to create a basic user interface using any function in Python. A deep

A grid of letters with numbers

Description automatically generated with medium confidence

Requirements

To operate this incredible project, you must have a basic understanding of Python programming, the Tkinter toolkit for GUI creation, and deep learning with the Keras library.

MNIST's dataset

MNIST is the most widely used dataset for machine learning and deep learning enthusiasts among the thousands of datasets available on the market. The MNIST dataset contains more than 10,000 images for testing and more than 60,000 training images of handwritten digits from zero to nine. Thus, the MNIST dataset contains ten distinct classes. The handwritten digit images are displayed as a 28 by 28 matrix, with a grayscale pixel value in each cell.

**Gradio**

**A logo with a square and a square in the middle

Description automatically generated with medium confidence**

Gradio is a Python package with an intuitive interface that makes working with machine learning models and deploying them easier. It offers a high-level abstraction for building interactive user interfaces (UIs) for your models, making deployment simple even for those without a deep understanding of web development. Gradio is flexible for many model types because it supports a wide range of machine learning frameworks.

Here's a quick rundown of Gradio's main attributes:

**Simpleness of Use:**

Gradio is a graphical user interface-based machine learning model deployment tool that requires very little coding. You may give your model an interactive user interface (UI) with a few lines of code.

**Support for Multiple Frameworks**: Radio is compatible with a number of machine learning frameworks, such as PyTorch, TensorFlow, Scikit-Learn, and others. This makes it possible for you to easily deploy models created using various libraries.

**Extensive Variety of Input and Output Elements:** The range of input and output components offered by Gradio facilitates user interface design. While output components can show text, graphics, or even interactive graphs, input components can also include text boxes, sliders, and images.

**Real-time Model Updates:** Radio’s ability to dynamically alter the user interface (UI) in real-time when users interact with the model is one of its features. Users may instantly see how changes in input impact the model's predictions thanks to this capability.

**Secure Deployment:** You can use HTTPS encryption to deploy your machine learning model using the secure deployment options that Radio offers. This guarantees secure connection between the deployed model and the user's browser.

**Combining Pre-trained Models:** Pre-trained model deployment is made easier using Gradio. Gradio makes it simple to provide an interface for users to interact with a pre-trained machine learning model without needing to comprehend the underlying model architecture.

**Custom Style**:Gradio has stylistic choices that enable you to tailor the UI's appearance. The colours, layout, and other elements can be changed to fit the style of your application.

**Basic example for deploy model:**

**import gradio as gr**

**import tensorflow as tf**

**# Load your pre-trained model**

**model = tf.keras.models.load\_model('your\_model.h5')**

**# Create a Gradio interface**

**iface = gr.Interface(fn=model.predict, inputs="image", outputs="label")**

**# Launch the interface**

**iface.launch()**

**Example: Image Recognition Interface of App built by Gradio.**

A screenshot of a computer

Description automatically generated

**Streamlit**

**A screenshot of a computer

Description automatically generated**

An open-source, user-friendly Python framework called Streamlit was created to make data science and machine learning interactive web applications more quickly and easily. Streamlit is a well-known tool that makes it easy for developers to convert data scripts into shareable, working web applications with just a few lines of code. With a high-level API that makes it easy to integrate widgets, charts, and other components quickly, it excels at rapid prototyping. Real-time updating, which enables applications to react dynamically to user activities, is one of its main advantages. Streamlit is adaptable for a variety of data-driven applications since it easily interfaces with well-known data science libraries like Pandas and Matplotlib.

**Key Concepts**:

**Routing:** To specify the URL patterns that cause particular functions in your Python code to run, Flask employs route decorators. For instance, you can specify a path to manage forecasts or generate an online website.

**from flask import Flask, request**

**app = Flask(\_\_name\_\_)**

**@app.route('/')**

**def home():**

**return 'Hello, World!'**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(debug=True)**

**Request Handling**: You can handle GET and POST HTTP requests with Flask. Either the request body or its parameters contain data that can be extracted. Sending data to your machine learning model requires this.

**@app.route('/predict', methods=['POST'])**

**def predict():**

**data = request.get\_json() # Extract JSON data from the request**

**# Make predictions using the model**

**return jsonify({'prediction': 'result'})**

**Rendering HTML Templates:** Flask has the ability to render HTML templates. Templates can be used to make dynamic webpages with forms for user interaction or results displays.

**from flask import render\_template**

**@app.route('/result')**

**def result():**

**prediction = 'some result'**

**return render\_template('result.html', prediction=prediction)**

**Static Files:** JavaScript files, stylesheets, and pictures are examples of static files that Flask can serve. When adding resources to your web application, this is helpful.

**from flask import send\_from\_directory**

**@app.route('/static/<path:filename>')**

**def serve\_static(filename):**

**return send\_from\_directory('static', filename)**

**Model Integration:** By loading your machine learning model and utilising it to generate predictions in the request-handling routines, you may include your model into a Flask application.

**import tensorflow as tf**

**model = tf.keras.models.load\_model('your\_model.h5')**

**@app.route('/predict', methods=['POST'])**

**def predict():**

**data = request.get\_json()**

**# Make predictions using the loaded model**

**result = model.predict(data['input'])**

**return jsonify({'prediction': result.tolist()})**

**Deployment Example:**

* **Set up Flask:**

Install Flask using pip

* **Make a Flask application:**

Make your Flask application defined in a file called app.py.

* **Load Model:**

In the Flask app, load your machine learning model.

* **Define the routes:**

Make routes to manage various components of your programme, such managing predictions or page rendering.

* **Launch the App:**

Open your terminal and type python app.py to launch the Flask development server. Use your browser to navigate to http://127.0.0.1:5000/ to interact with your application.

**Flask**

Flask is a popular Python online application framework that is lightweight, incredibly versatile, and easy to use. Flask functions as a microframework, offering the fundamental elements required for web development without enforcing a burdensome structure, enabling developers to make well-informed decisions regarding the libraries and tools they incorporate. Flask contains a straightforward routing system that makes it possible to define routes for responding to different types of HTTP requests. Its integrated template engine, Jinja2, also makes it easier to render information The Important attributes and traits of Flask are as follows:

**Microframework:** Flask is frequently called a "microframework" because to its user-friendly, straightforward design. It gives developers the flexibility to select various libraries and tools while still offering the essential features required for web development.

**Routing:** The routing mechanism used by Flask is simple and easy to understand. By defining routes to distinct areas of their application, developers can indicate how the program should react to different types of HTTP requests.

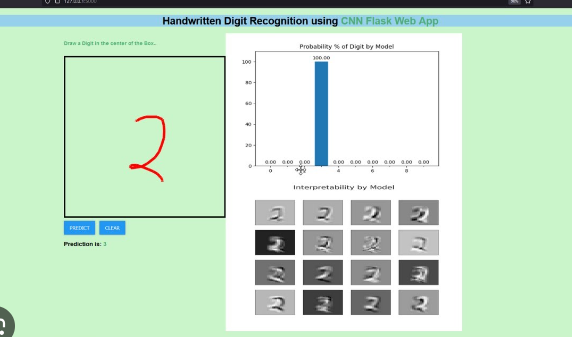
**Template Engine**: Flask comes with Jinja2, an integrated template engine that makes it easier to render dynamic information in HTML. This makes it possible to create dynamic webpages using backend data.

**Werkzeug Integration**: The Werkzeug WSGI serves as the foundation for Flask.extensibility of the framework, which allows for a variety of modifications to improve functions like database connectivity and authentication, is what makes it so strong.

**Community and Documentation**: There are a ton of resources and third-party extensions available for Flask due to its huge and active community. For developers, the official documentation is an invaluable resource because it is thorough.

**Scalable and Lightweight**: Flask is an excellent choice for small to medium-sized applications because to its lightweight design. its flexibility and simplicity allow it to be easily scaled to meet the demands of individual projects.

**Example: Handwritten Recognition App**



**Model Building**

Developed Convolutional Neural Network (CNN) model, for exploring deployment options through Gradio, Streamlit, and Flask to provide interactive and user-friendly interfaces for utilizing the model.

****

**A screenshot of a computer

Description automatically generated**

Using the MNIST dataset, I trained a Convolutional Neural Network (CNN) model, and saved the model as **handwritten\_digit\_mode.h5** , which I'm currently deploying using Gradio, Streamlit, and Flask to build user-friendly interfaces where users can interact with the predictions of the model.

**Link:** https://colab.research.google.com/drive/1lqSdi05rmfLrjbpceFBOWTOVf24O6RRz?usp=sharing

**User Interfaces using**

**1.Gradio**

**2.Sreamlit**

**3.Flask**

1. **GRADIO**

**CODE:**

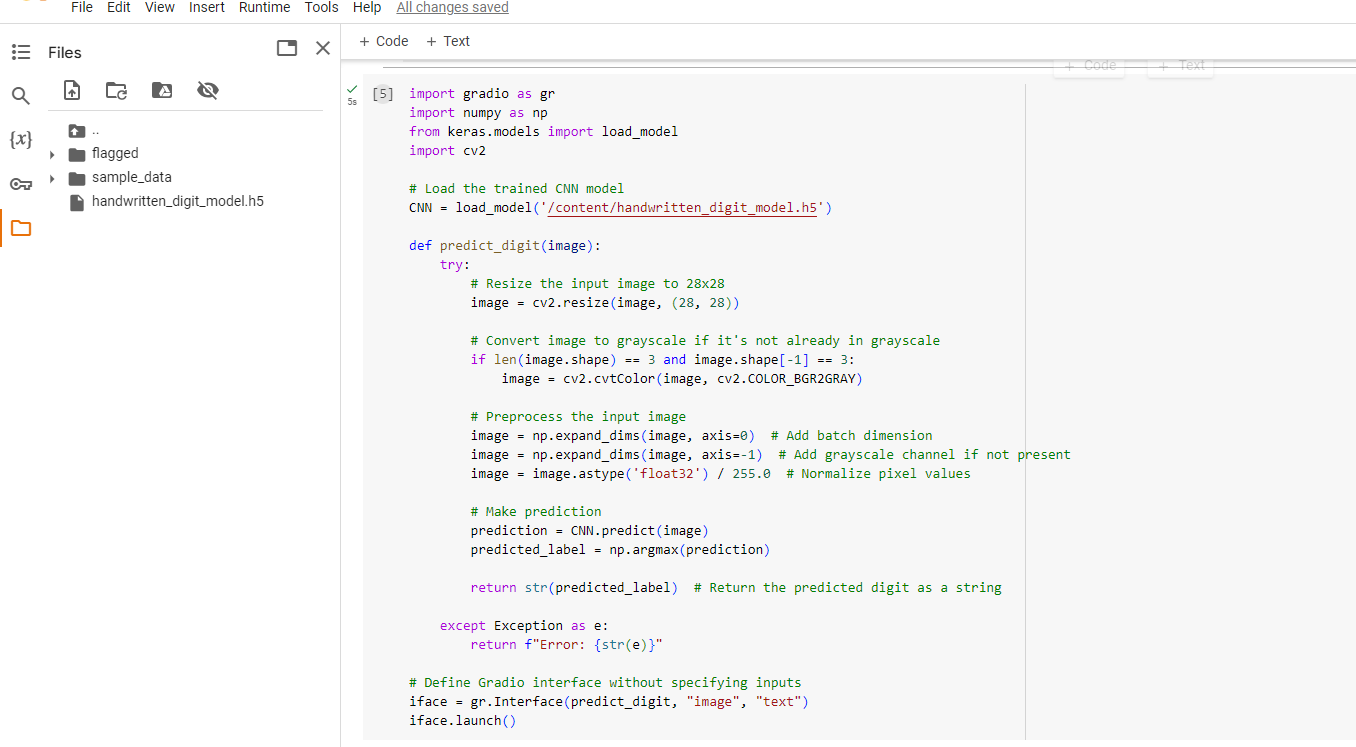
**Installing gradio into Google Colab using pip**

A screenshot of a computer program

Description automatically generated

Loading our model handwriiten\_digit\_model.h5 inti new colab Notebook.

Launching Gradio interface for predicting Handwritten digits



**Input image:**



A black heart with a white background

Description automatically generated with medium confidence

**Output:**

**We got the predicted digit as 2**

1. **Streamlit**

**CODE:**

import streamlit as st

import numpy as np

from keras.models import load\_model

import cv2

# Load the trained CNN model

CNN = load\_model("C:/project/handwritten\_digit\_model.h5")

st.title('HandWritten Prediction App')

# Upload image

uploaded\_file = st.file\_uploader("Choose an image...", type=["jpg", "png"])

if uploaded\_file is not None:

    # Read the uploaded image as OpenCV image

    file\_bytes = np.asarray(bytearray(uploaded\_file.read()), dtype=np.uint8)

    opencv\_image = cv2.imdecode(file\_bytes, 1)

    # Preprocess the image

    resized\_image = cv2.resize(opencv\_image, (28, 28))  # Resize to 28x28 pixels

    gray\_image = cv2.cvtColor(resized\_image, cv2.COLOR\_BGR2GRAY)  # Convert to grayscale

    normalized\_image = gray\_image.astype('float32') / 255.0  # Normalize pixel values

    # Make prediction

    prediction = CNN.predict(np.expand\_dims(normalized\_image, axis=(0, -1)))

    predicted\_label = np.argmax(prediction)

    st.image(resized\_image, caption='Uploaded Image', use\_column\_width=True)

    st.write(f"Predicted Digit: {predicted\_label}")

First of all we saved the Streamlit code in Visual Studio as Streamlit.py in a new folder called project in C directory. Our model also copied into the same folder.

**Using command Prompt, we got our Handwritten prediction App URL.**

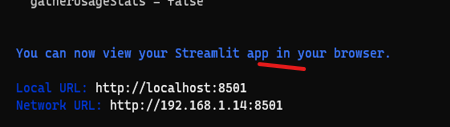
**Code:**

First, we installed necessary packages and set the command into our project folder in C directory

then using the below prompt

**-Sreamlit run Streamlit.py**

we got URL for our HANDWRITTEN PREDICTION APP

****

A computer screen with white text and green text

Description automatically generated

A screenshot of a computer

Description automatically generated

**While clicking the URL, we directed into our Handwritten Prediction APP**

**Interface:**

A screenshot of a computer

Description automatically generated

We give a Handwritten digit as input



Streamlit App Predicted it as 2.

1. **FLASK:**

This is done in visual studio code.

Created code for both app.py and the HTML templates (index.html and result.html). then Create a folder named templates in the same directory as your app.py, and place the HTML files inside it.

A screenshot of a computer

Description automatically generated

**Code:**

**app.py**

A screenshot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

**Index.html**

A screen shot of a computer program

Description automatically generated

**result.html**

A screen shot of a computer

Description automatically generated

Saved these files in the respective locations and run Flask application using the command

**python app.py** in the terminal of the visual studio code. Opened the URL http://127.0.0.1:5000/

to interact application.

A screenshot of a computer screen

Description automatically generated

**Interface of the Flas App:**

A screenshot of a computer

Description automatically generated

**Input:**



**Output:**

A screenshot of a computer

Description automatically generated

**Conclusion:**

Using Gradio, Streamlit, and Flask to develop a handwritten digit prediction application offered a variety of deployment choices, each with its own advantages.

**Gradio:** Gradio's outstanding simplicity and usability allowed for the quick creation of an interactive user interface (UI) for the digit prediction model. Quick experimentation and smooth integration with the machine learning model were made possible by its high-level abstraction. Gradio was a good option for designing an interface that was both aesthetically pleasing and intuitive because of its varied input/output components and dynamic updating.

**Streamlit:** Streamlit's ease of use and emphasis on data apps made it possible to quickly and easily create an interactive online application. Building a user-friendly interface was made simple by its reactive programming paradigm, and the handwritten digit prediction algorithm's smooth interaction with the Streamlit workflow was a nice bonus. A seamless user experience was facilitated by Streamlit's capacity to manage user interactions and present forecasts in an understandable and straightforward manner.

**Flask:** Flask allowed for more customisation and offered a finer level of control over the structure of the application. Compared to Gradio and Streamlit, Flask required more manual setup, but it provided the flexibility required for larger-scale projects or applications with particular needs. The routing, request handling, and integration features of Flask made it possible to customise the deployment procedure.

Each framework provided a different method for using the prediction model for handwritten numbers. Flask offered the flexibility for more extensive customisation, Gradio excelled in simplicity and speedy deployment, and Streamlit offered an effective technique to develop interactive data apps.

**Reference:**

* [**https://analyze314.blogspot.com/2023/11/navigating-through-tech-landscape-step.html?m=1**](https://analyze314.blogspot.com/2023/11/navigating-through-tech-landscape-step.html?m=1)
* [**https://stackshare.io/gradio**](https://stackshare.io/gradio)
* [**https://rcarrata.com/ai/gradio-k8s/**](https://rcarrata.com/ai/gradio-k8s/)
* [**https://stackoverflow.com/questions/77155566/integrating-gradio-streamlit-with-flask-for-multi-model-hosting-and-route-custom**](https://stackoverflow.com/questions/77155566/integrating-gradio-streamlit-with-flask-for-multi-model-hosting-and-route-custom)
* [**https://www.linkedin.com/pulse/small-comparison-web-ui-tools-machine-learning-balayogi-g**](https://www.linkedin.com/pulse/small-comparison-web-ui-tools-machine-learning-balayogi-g)
* [**https://www.tutorialspoint.com/flask/index.htm**](https://www.tutorialspoint.com/flask/index.htm)
* [**https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/**](https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/)

**THANK YOU!**