Software Application for Artificial Intelligence Powered Seed Identification Proposal

By

Matthew Morrow

Florida Institute of Technology

CIS 5898 – Projects in Computer Information Systems

for

Dr. Bernard Parenteau

December 1, 2024

Introduction

The Seed Identifier project, named SeedID, is an advanced AI-driven application designed to facilitate the precise identification of various seed types through image recognition technology. The application serves to solve the problem of diet and agriculture seed identification in the professional fields of agriculture, conservation, and research. A process that would typically require highly specialized experts or countless hours of bookwork to ID. SeedID makes this process easy by including a feature for users to upload their seed images, delivering reliable identification labels accompanied by weighted confidence scores.

By the end of the project, the goal is to implement online hosting for both data and processing. Training and datasets will be hosted on Microsoft Azure, where machine learning models will be trained on extensive seed image datasets to optimize accuracy and performance. Additionally, strategies for image preprocessing and computational optimization will be employed to enhance the system's overall robustness and responsiveness.

The project addresses the much needed crossroads between technology and the complex world of organic biology. Offering rapid, accurate seed identification- a cornerstone for effective crop management, biodiversity monitoring, and quality assurance within seed distribution systems. By developing a scalable, AI-powered solution, SeedID offers the potential to streamline workflows, reduce human error, and enhance decision-making across industries that depend on accurate seed classification.

Algorithms/Project Solution

Leveraging Python and the PyTorch framework, the following will be the structured approach based on the SDLC. The listed functions are built into the framework and sourced from the information found in the PyTorch official documentation. They, in addition to supplemental custom helper functions, will offer ease of use for preparing and running SeedID.

1. **Image Preprocessing**. This includes the handling of image data and formatting the image to be a in a palatable setting for the recognition engine to process.

    Functions:

    load_image(path: string): Load image from a file path.

    resize_image(image, sizing: tuple): Resize the images to fit the required input from the model.

    normalize_image(image): adjust the pixel values.

    Data Input Values:

    Image files – .jpg and/or .png.

Tensoring:

After resizing, the image needs to be converted to a tensor format. PyTorch provides utilities like transforms.ToTensor() that convert an image to a tensor and automatically scales pixel values from [0, 255] to [0, 1] by dividing by 255.

2. **Model Definition Module**. Defining the architecture of your image recognition neural network.

   Functions:

   define_model(): Sets up the neural network layers using. This will be used for pooling, convolutional, and fully connecting layers.

   initialize_weights(model): Applies weight initialization to the model layers for consistent training.

   normalize_image(image): adjust the pixel values.

   Data Input Values:

   Architecture params - number of layers and activation.

   Output:

   PyTorch model object.

3. **Training**. Setting up the training for the model.

   Functions:

   train_model(model, dataloader, criterion, optimizer): Trains the model over a specified number of epochs, adjusting weights based on loss.

   adjust_learning_rate(optimizer, epoch): Dynamically adjusts the learning rate during training for better convergence.

   Data Input Values:

   Training the model to receive labels

   Output:

   Having a trained model to use in the core implementation.

4. **Prediction and Inteference**. The application will have a method of displaying prediction confidence. This should be displayed in addition to the user selected image. Providing the user with the information needed to determine if the accuracy of the application is sufficient.

Functions:

predict_image(model, image): Accepts an input image and returns the predicted class label or probability distribution

visualize_predictions(image, predictions): Displays the image alongside its predicted labels and confidences.

Data Input Values:

Provided user images.

Output:

Prediction labels and confidence scores.

5. **Model Saving and Loading**. After a model is trained there needs to be functionality to save and load it into the application for running the comparisons. This will require the following two functions:

   Functions:

   save_model(model, path: str): Saves the trained model's state dictionary to a file.

   load_model(path: str): Loads the previously saved model for inference or further training.

   Data Input Values:

   Model object and the file path.

   Output:

   The model file and the loaded/trained model.

6. **Graphical User Interface (GUI)**. An interface is required for the user to be able to interact with the application. Such actions include training the model, uploading images from local directories to for identification, and reviewing prediction scores and results.

   Functions:

   upload_image(): Allows the user to upload an image for recognition.

   display_results(predictions): Displays the output to the user.

   Data Input Values:

   User input images sourced from local repositories.

   Output:

   Prediction results and confidence scores.

7. **<u>Error Handling and Logging.</u>** When errors or user actions result in a log worthy event the messages should be stored in a local file for triage and in some instances displayed to the user via a prompt on the UI.

    Functions:

    log_event(event: str): Logs key events such as model training start/stop, errors, or other actions.

    handle_errors(): Catches and handles any errors that occur during runtime, preventing the system from crashing.

    Data Input Values:

    Events and user actions

    Output:

    File logging and user visible error messages.

Database Description

To be able to host the application online, a cloud-based database is required. A workflow with Azure ensures that the image recognition system has a scalable and organized method for managing both image data and associated metadata. Microsoft's Azure offers solutions that store the image as a long string known as a Blob. The Azure SQL Database with Blob Storage will begin by uploading the image files to Azure Blob Storage. Each image will be assigned a unique URL or path, which will be used during the model's training and inference stages.

Next, a table will be created in Azure SQL Database to store relevant metadata about each image. This table will include columns for the image ID, the URL linking to the image in Blob Storage, labels or classifications, predictions, and timestamps for when the images were uploaded. This metadata will provide easy access to both the image location and important details about the image's content and model output. Once the images are stored in Blob Storage and the metadata is organized in SQL Database, a connection between the two will be established. The SQL Database will serve as the central hub for querying image information, and the stored URLs will be used to retrieve and load the images into the AI system using PyTorch for tasks such as training and prediction.

Implementation

The programming language selected for handling the application logic is Python 3.12.4, chosen for its extensive libraries, flexibility, and performance in AI and machine learning tasks. Pythons Pytorch will be a critical resource for the implementation and model training of SeedID. The Python code will be developed using the PyCharm Community Edition IDE, which seamlessly supports Python development. For database management, the cloud-based solutions provided by Microsoft Azure will serve as the primary data storage platform. Widely supported by the previously mentioned tools, Windows 11, is the chosen OS platform to host the project's development and presentation.

On the hardware end, the image learning processing will be run by the latest in personal computing technology- a 13th Gen Intel Core i9-13980HX CPU and a NVIDIA GeForce RTX 4070 GPU. With 64GB of localized memory, the processing and training should be relatively quick and fully capable of ramping up the pixelation, weights, and epochs required for effective and more accurate model training.

Some of the images used in the model training will require taken-from-the-field photos. SeedID is in the possession of a professional camera setup with proper macro lenses and needed light diffusion. The camera used will be a Sony A7IV with a Sony 90mm lens. Photos will be stacked and edited in post-production software such as Adobe's Lightroom.

The combination of Python's ease of use, Azure's cloud solutions, and the cutting-edge hardware allows for efficient processing, training, and deployment of SeedID. This setup ensures the program's overall scalability, speed, and accuracy in identifying seeds based on images provided by users or external datasets.

The Seed Data

The primary dataset for this project consists of seed images, sourced from three distinct origins. Initially, images will be obtained from publicly available online sources that provide stock footage of simple, easily identifiable seeds such as wheat grains, corn kernels, and almonds. These readily accessible images will serve as the foundation for early-stage model development.

The second phase will involve collecting images directly from fieldwork, capturing seeds that are more complex and less commonly recognized, such as dandelion seeds or mango seeds from local trees. These field-sourced images will add diversity and will scale up the challenge level in the dataset by introducing natural variability.

Finally, images will be sourced from research labs, taken by qualified staff in academic or conservation institutions. These lab-sourced images often include measurement markers and multiple seeds within a single image, which may introduce challenges during model training. However, they represent a more realistic application scenario for SeedID, simulating the conditions under which the software is expected to perform in practical, real-world contexts.

Examples of Online Sourced Seed Data

Corn. Image 1

Wheat. Image 2

Almond. Image 3

Examples of Lab Sourced Seed Data

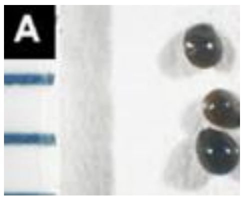


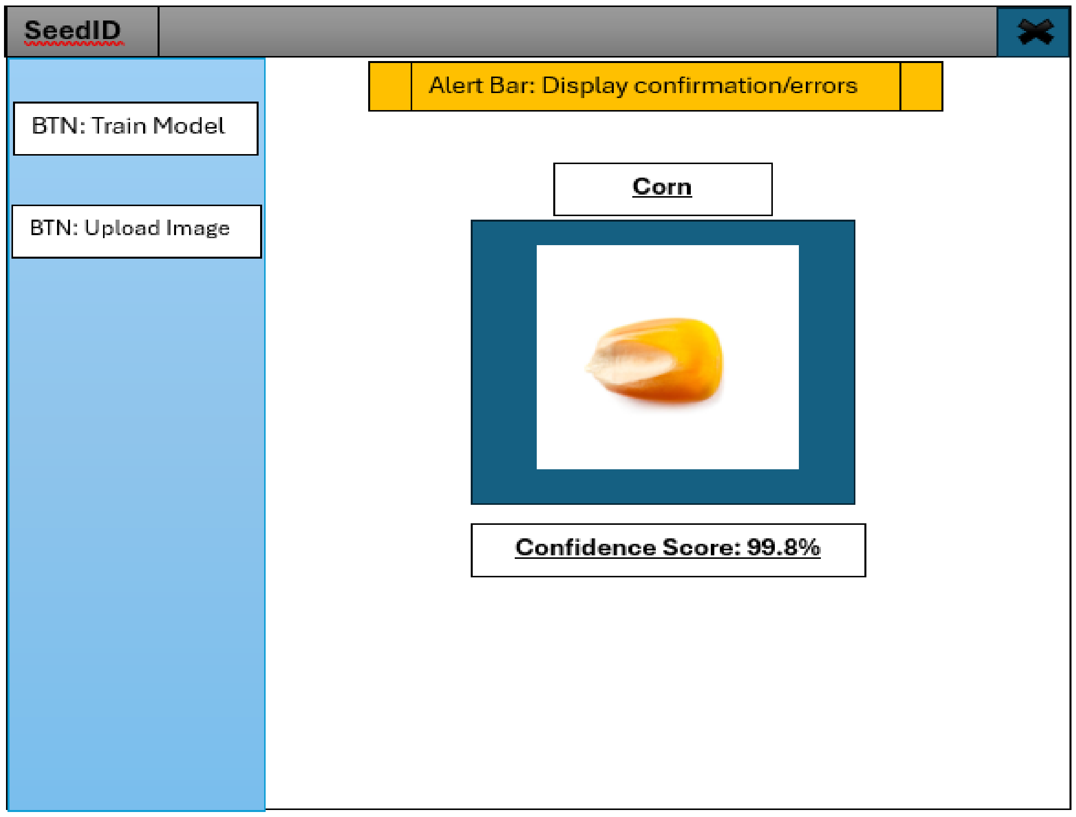
Image 4

Image 5

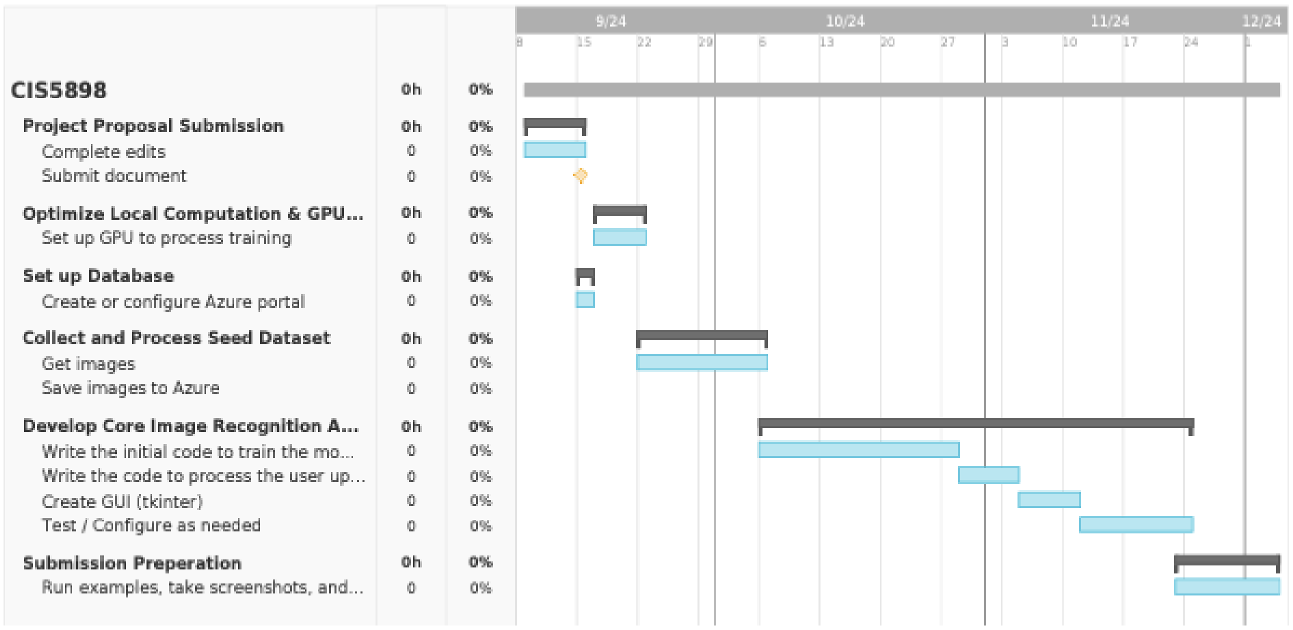
Image 6

Graphical User Interface

The following outline provides an overview of the anticipated GUI layout for the application. The top bar will feature the tool's name along with window control options, including the close button. A panel on the left-hand side will contain the user interface buttons, allowing for easy navigation and interaction. The central panel will display the user-selected image, accompanied by the predicted confidence score and the label indicating the identified seed's name.



Schedule (Gantt Chart)



CIS5898	**0h**	**0%**
Project Proposal Submission	**0h**	**0%**
Complete edits	0	0%
Submit document	0	0%
Optimize Local Computation & GPU...	**0h**	**0%**
Set up GPU to process training	0	0%
Set up Database	**0h**	**0%**
Create or configure Azure portal	0	0%
Collect and Process Seed Dataset	**0h**	**0%**
Get images	0	0%
Save images to Azure	0	0%
Develop Core Image Recognition A...	**0h**	**0%**
Write the initial code to train the mo...	0	0%
Write the code to process the user up...	0	0%
Create GUI (tkinter)	0	0%
Test / Configure as needed	0	0%
Submission Preperation	**0h**	**0%**
Run examples, take screenshots, and...	0	0%

The chart breaks down the project into specific, time-bound phases, such as submitting the project proposal, setting up local computation and GPU resources, configuring the Azure database, collecting and processing seed images, developing the core image recognition application, and preparing the final submission. The timeline spans from mid-September to late December, with tasks like the project proposal expected to finish early, while more complex tasks such as developing the core application span a longer period.

Gray bars represent the planned duration of each task, while blue bars indicate active or ongoing work. The chart also includes dependency markers, showing that some tasks rely on the completion of others. For example, setting up the database is a prerequisite for collecting and processing the seed dataset.

References

Chollet, F. (2018). Deep learning with Python. Manning Publications.

FreePNGImg. (n.d.). Almond PNG image. FreePNGImg. https://freepngimg.com/png/9490-almond-png-image

Gantt Chart. (n.d.). TeamGantt. https://www.teamgantt.com

iStock. (n.d.). Corn seeds [Photographs]. iStock. https://www.istockphoto.com/photos/corn-seeds

PNG Hunter. (n.d.). Wheat. PNG Hunter. https://pnghunter.com/png/wheat-49/

PyTorch. (n.d.). PyTorch documentation. PyTorch. Retrieved September 11, 2024, from https://pytorch.org/docs/

University of Florida. (n.d.). Images 4-6.