**Research Labs Inventory**

Sumedha Bhattacharyaa

**machine learning model subsystem final report**

REVISION – Final

5 December 2024

Subsystem final report

for

Research Lab Inventory

Prepared by:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Author Date

Approved by:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Leader Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

John Lusher, P.E. Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

T/A Date

**Change Record**

| **Rev.** | **Date** | **Originator** | **Approvals** | **Description** |
| --- | --- | --- | --- | --- |
| **1** | 12/05/2024 | Sumedha Bhattacharyaa |  | Final Release |

**Table of Contents**

[**1. Introduction 6**](#_heading=h.3dy6vkm)

[**2. Development Environment and Tools 6**](#_heading=h.3dy6vkm)

[**3. Data Preprocessing 6**](#_heading=h.1t3h5sf)

[3.1 Data Collection 6](#_heading=h.4d34og8)

[3.2 Data Transformation 7](#_heading=h.2s8eyo1)

[3.3 Dataset Splitting 7](#_heading=h.17dp8vu)

[**4. Model Architecture 8**](#_heading=h.3rdcrjn)

[4.1 Overview of the CNN Design 8](#_heading=h.26in1rg)

[4.2 Flow of Data 9](#_heading=h.lnxbz9)

[**5. Training Phase 10**](#_heading=h.35nkun2)

[5.1 Training Data 10](#_heading=h.1ksv4uv)

[5.2 Training Pipeline 10](#_heading=h.44sinio)

[5.3 Performance Metrics 10](#_heading=h.2jxsxqh)

[**6. Validation and Testing 10**](#_heading=h.z337ya)

[6.1 Validation Process 10](#_heading=h.3j2qqm3)

[6.2 Testing Phase 10](#_heading=h.1y810tw)

[6.3 Testing Metrics 11](#_heading=h.a6tjaon6jhmy)

[**7. Challenges and Solutions 12**](#_heading=h.4i7ojhp)

[7.1 Issues Encountered 12](#_heading=h.2xcytpi)

[7.2 Mitigation Strategies 12](#_heading=h.1ci93xb)

[**8. Integration Plans 13**](#_heading=h.3whwml4)

[8.1 Future Integration with the Android App 13](#_heading=h.2bn6wsx)

[8.2 Deployment Considerations 13](#_heading=h.qsh70q)

[**9. Future Improvements 13**](#_heading=h.3as4poj)

[**10. Conclusion 13**](#_heading=h.1pxezwc)

[**11. References 14**](#_heading=h.iizvsk1puna6)

**List of Tables**

No table entries.

**List of Figures**

[Figure 1. Dataset Organization 8](#_heading=h.oo6tpggw0lyv)

[Figure 2. Comprehensive Model Architecture Diagram 9](#_heading=h.r5ox53ag1j90)

[Figure 3. False Positive (Incorrect Negative Classification) Rate 11](#_heading=h.3l98bhry82je)

[Figure 4. Classification Report of Measured Metrics from Testing 11](#_heading=h.icdajcamj8zl)

[Figure 5. Confusion Matrix 12](#_heading=h.c95z6k7fomld)

# 1. Introduction

The machine learning (ML) subsystem is designed to automate the identification and classification of lab items from images to allow mobile users to check in their items back into the lab’s inventory. Using a Convolutional Neural Network (CNN), this subsystem processes input images and predicts their corresponding categories, providing a streamlined solution for inventory management. Currently, the system functions as a standalone implementation, but integration with an Android mobile app is planned for next semester. This integration will allow users to upload images directly via the app and receive real-time predictions from the ML model, enabling efficient inventory tracking and an easy way to check in items.

# 2. Development Environment and Tools

The development of the ML subsystem was carried out using a combination of tools and libraries. **Visual Studio Code (VS Code)** was used as the primary integrated development environment (IDE) for writing and debugging the code, offering an efficient and organized workspace. The implementation was done in **Python** because it’s well-suited for machine learning tasks with its many libraries and frameworks. The core model was built using **PyTorch**, a widely used deep learning framework that simplifies the process of defining and training neural networks. Together, these tools provided a robust development environment for building this subsystem.

# 3. Data Preprocessing

## 3.1 Data Collection

The datasets used for this project were taken from **Kaggle**, which is a website with several datasets made for deep learning models. Each image was put in a folder with its respective class name and the classes consisted of resistors, screws, and PCB (Printed Circuit Board) components. Each class had around 1200 images in total. These categorized images form the foundation of the training and validation datasets. There was also a data set called ‘misclassified’ which contained images that intentionally did not belong to the predefined categories. For example, in the ‘screws’ category, the misclassified version included images of unrelated objects, such as wires or laptops. The purpose of this dataset was to measure the model's false positive rate, which indicates how often the model incorrectly classifies non-category images as belonging to a specific class.

## 3.2 Data Transformation

The preprocessing pipeline ensures that all input data is standardized and suitable for the model. Images of varying dimensions were resized to 64x64 pixels, providing uniformity and reducing computational load during training. The pixel values were normalized to a range of [0, 1], which improves numerical stability. To enhance the diversity of the dataset, data augmentation techniques, including random rotations, flipping, and cropping, were applied. This augmentation helped the model generalize better and reduces the likelihood of overfitting.

## 3.3 Dataset Splitting

The preprocessed dataset was divided into two parts: 75% was used for training the model, 15% was used to validate the model, and 10% was used to test the model. This split allows the model to learn patterns from the training data while using the validation set to evaluate its generalization performance.

A screenshot of a computer

Description automatically generated

### *Figure 1. Dataset Organization*

# 4. Model Architecture

## 4.1 Overview of the CNN Design

The CNN (Convolutional Neural Network) model used in this subsystem is composed of multiple layers designed to extract features and classify images. The first layer is the input layer, which accepts resized 64x64 RGB images. This is followed by three convolutional layers that progressively extract more complex features from the images. Each convolutional layer is followed by a Rectified Linear Unit (ReLU) activation function and a MaxPooling layer. ReLU is a mathematical function which sets all negative values to zero while keeping positive values unchanged. This activation introduces non-linearity into the model, allowing it to learn complex patterns and relationships in the data. MaxPooling layers downsample the feature maps by selecting the most prominent features, reducing spatial dimensions and complexity.

*Equation 1. The Rectified Linear Unit (ReLU) Activation Function.*

After the convolutional layers, the output is flattened into a one-dimensional vector to prepare it for the fully connected layers. The first fully connected layer (FC1) contains 256 neurons and applies ReLU activation to further process the features. A dropout layer follows FC1, randomly deactivating 50% of its neurons during training to prevent overfitting and improve generalization. The final fully connected layer (FC2) outputs a vector of probabilities corresponding to the number of classes, enabling the classification of the input image.

## 4.2 Flow of Data

The input image is first passed through Conv1 with ReLU activation and MaxPooling. This process is repeated for Conv2 and Conv3, extracting progressively higher-level features. The resulting feature maps are flattened into a vector and passed through FC1, where Dropout is applied. Finally, the output from FC2 provides the predicted class label for the image.

A diagram of a flowchart

Description automatically generated

### *Figure 2. Comprehensive Model Architecture Diagram*

# 5. Training Phase

## 5.1 Training Data

The training dataset consisted of preprocessed images categorized into their respective class folder. These images were used to teach the model to identify patterns and distinguish between different categories of lab items.

## 5.2 Training Pipeline

During training, the input images were passed through the network in a forward pass, where predictions were generated at the output layer. The loss function, CrossEntropyLoss, calculates the difference between predicted and true labels. This loss was then backpropagated through the network to compute gradients, which were used to update the model’s weights using the Adam optimizer. This iterative process continues for several epochs (10) until the model achieved satisfactory performance on the training dataset.

## 5.3 Performance Metrics

The model’s training performance is evaluated using loss, which quantifies the difference between predicted and true labels. This metric provides insights into how well the model is learning and generalizing to the data. A low loss number indicates that the model is learning the data well. The loss that this model had achieved after 10 epochs, was 0.0049, indicating a low loss.

# 6. Validation and Testing

## 6.1 Validation Process

The validation dataset was used to monitor the model’s performance on unseen data. The validation metric that was recorded was accuracy. If the validation accuracy is high, while the training loss is also high, there is an indication of overfitting data. However, since this model achieved a standard validation accuracy of 96.35% with the aforementioned low loss number, it’s safe to say this model is not overfitting.

## 6.2 Testing Phase

After training and validation, the final model was tested on a separate test set to evaluate its generalization ability. Additionally, the 'misclassified' dataset was processed through the model to calculate the number of false positives. The Correct Negative Classification Rate was 91.90% and the False Positive Rate was 8.10% indicating that the model has few false positives.

A black background with white text

Description automatically generated

### *Figure 3. False Positive (Incorrect Negative Classification) Rate*

## 6.3 Testing Metrics

Key metrics such as **precision**, **recall**, and the **F1 score** were computed during the testing phase to provide deeper insights into the model's performance for each class. For instance, the model achieved a **precision** of **97%**, a **recall** of **96%**, and an **F1-score** of **96%**, averaged across all classes. These results indicate that the model is effective at making correct predictions and consistent in identifying instances of each class. Additionally, a confusion matrix for the testing phase was created. The x-axis variations (numbers outside the highlighted diagonal) are false positives – images that were predicted to be a class when they’re not from that class. The y-axis variations indicate false negatives – images that were classified as not being from that class when they are from that class. There were 17 false positives and 0 false negatives.

A screenshot of a computer screen

Description automatically generated

### *Figure 4. Classification Report of Measured Metrics from Testing*

A diagram of a confused matrix

Description automatically generated

### *Figure 5. Confusion Matrix*

# 7. Challenges and Solutions

## **7.1 Issues Encountered**

The project faced challenges such as data imbalance, where certain classes (screws) had fewer examples, leading to potential bias.

## **7.2 Mitigation Strategies**

To address these challenges, data augmentation techniques were employed to artificially increase the dataset's diversity. Dropout layers were also used during training to prevent overfitting by randomly deactivating neurons in the fully connected layers.

# 8. Integration Plans

## **8.1 Future Integration with the Android App**

In the next semester, the machine learning model will be integrated into an Android mobile app. The app, developed by a teammate, will allow users to upload images of lab items. These images will be sent to a backend server where the model will process them and return predictions. The app will display the predicted class label, providing real-time feedback to users.

## **8.2 Deployment Considerations**

To achieve seamless integration, the trained model will be saved in a server-compatible format, such as PyTorch or ONNX. A RESTful API (Representational State Transfer Application Programming Interface) will be developed to facilitate communication between the app and the server. This API will handle tasks such as receiving image uploads, preprocessing them, and passing them to the model for classification. Testing will focus on ensuring minimal latency and accurate predictions, for a smooth user experience.

# 9. Future Improvements

In the future, the model’s performance can be enhanced by experimenting with more advanced architectures, such as ResNet, which may work more effectively. Expanding the dataset to include additional classes and larger datasets will make the system even more versatile and practical for inventory management tasks.

# 10. Conclusion

The machine learning subsystem is a critical component of this project, enabling efficient classification of lab items once fully integrated. While currently demonstrating intended performance, its planned integration with the Android app will significantly enhance its usability. The system holds great potential for further development, paving the way for a comprehensive and user-friendly inventory management solution.

# 11. References

[1] Kaggle. "Kaggle Datasets." Available: https://www.kaggle.com/datasets, Accessed: September 2024.

[2] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," PyTorch Documentation. [Online]. Available: https://pytorch.org/docs. Accessed: September, 2024.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.