UNIVERSITY OF SOUTH CAROLINA

PROJECT MILESTONE #2

UNIVERSITY OF SOUTH CAROLINA

COMPUTER SCIENCE AND ENGINEERING

CSCE-585: Machine Learning Systems
Project Proposal: Visual ML System for Identifying and Minimizing Civilian Fatality in Urban War Zones

---

Authors:

Walter Pach (github.com/waltster),
Vera Svensson (github.com/Lux-Vera)

November 3, 2022

1 Project Repository

The repository for our team can be found at https://github.com/csce585-mlsystems/MilCiv. This repository is used for collaborating, issue tracking, and updating/version control. This milestone can be identified by the GitHub tag: **Milestone2**.

2 Introduction

As AI systems are increasingly deployed in battlefield environments, there is a need for efficient models that distinguish between civilian and military targets. Our project aims to create a ML system that is able to provide this functionality.

3 Problem Statement

Our goal is to develop an AI model that is enabled to differentiate between civilian and military vehicles. The problem presented includes both the issue of accuracy of characterization and the ability to audit the success of the system with a clearly defined confidence metric.

4 Technical Approach

- The data set was pre-labelled and we developed a Jupyter notebook to convert the CSV and images into TensorFlow Record format.
- We created a Jupyter Notebook to download and configure the dependencies for the project and format their configuration appropriately.
- The model's environment is configured from the Common Objects in Context model with TensorFlow object detection.
- The object detection library for TensorFlow is used and trained on the data set.
- Different parameters and training parameters were adjusted to achieve the fastest detection with the highest average accuracy.

To lint the code we have created a workflow in Github Actions. This makes the code easier to work with and help us find errors. The workflow is activated on every push to the repository to make sure unstructured code and possible errors are eliminated as early as possible.

We are using Super-Linter in our workflow, which is a combination of linters that has support for Python, among other languages [1]. However, it does not have support for ipynb files. Thus those files will have to be converted to Python before running the linter [2]. This is also done by the workflow.

To make the model more user friendly we've developed a front-end. The user can upload an image, and the model will perform classification and present a result.

HTML and CSS are used to create the visual components and styling of the page, while JavaScript is used to process the input, feed it to the model, and then transform and send back the model's output. These languages were chosen for their popularity. We were familiar with using them which made it easy to quickly set up a working front-end.

The front-end UI is powered by a custom HTTP model server that handles the processing and validation of input, as well as the decoding/encoding of images, processing of the model, and transmission of results. This requires the lightweight *server.py* to be running when the UI is used.

5 Intermediate/Preliminary Approach and Results

Currently, our project supports cross-platform building and installation with automation for the download, configuration, and training process. Our team believes that we have found the idea parameters for MobileNet in this case, but we are still experimenting with them and different training options and comparing results.

In the second iteration of the project, we aimed to develop more infrastructure around the model in support of its configuration and testing phases. Between milestone one and two we also developed a front-end user interface for uploading and processing photos

The front-end is still under development and is currently works locally on the users machine. This is powered by a custom model server that handles the processing of images.

At this time, users upload images from their local filesystem which are then compressed and transmitted over HTTP to the server as Base64 encoded byte streams. These are then received by the server, decoded, and passed to the model. The result is a robust, checksummed image that the client receives.

We are currently working on getting hold of, and presenting, the relevant information from the model's output regarding accuracy and improvements. Currently, the user is given information about the time that it took to process the image as well as which classifications were detected in the image.

Going forward, we will implement a suite of scripts to automate testing and accuracy reading and tweak our model configuration more. With our teammate dropping the project, we were unable to develop this part of the project yet.

In 1 the latest version of the UI is shown.

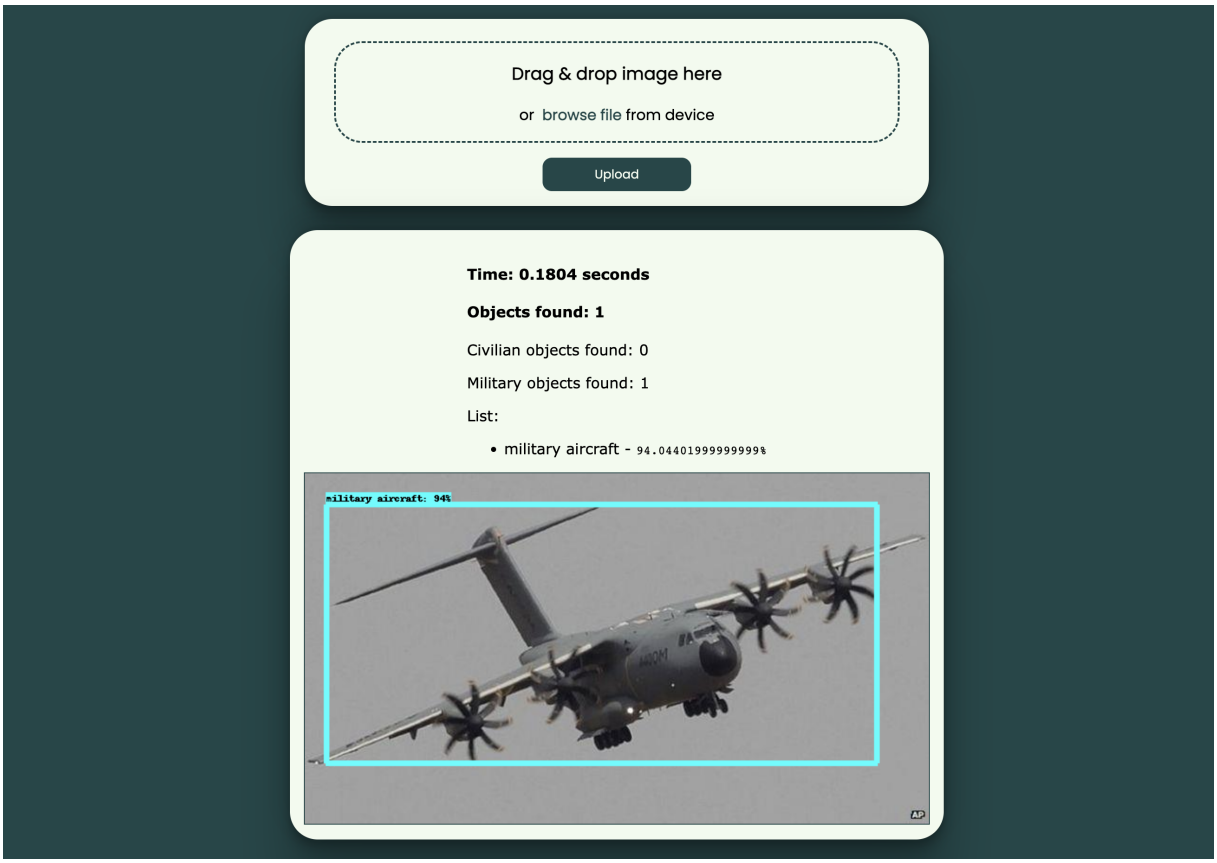

Figure 1: Screen shot of the latest version of the front-end. Both real- and mock statistics are shown.

References

[1] GitHub. Super-Linter. GitHub;. [cited 2 October 2022]. Available at `https://github.com/github/super-linter#supported-linters`. pages 2

[2] notebooks with GitHub Actions L. Donagh Horgan. Donagh Horgan; 2021. [cited 2 October 2022]. Available at `https://donagh.io/2021/05/10/linting-notebooks-with-github-actions.html`. pages 2