Collaborative Teaching Laboratory - AR Development

University of Birmingham
Jack Shilton
2022-03-31
Version 2.1

Abstract

This document describes the workflow, capabilities, and limitations of the `TensorFlow` and `Google MLKit` based approach to AI training that is currently being adopted by the Collaborative Teaching Laboratory (CTL) for Augmented Reality software at the University of Birmingham.

This report is to be used to assist developers in how to recreate, maintain, and update the systems currently in place. This report is to be kept up-to-date to reflect the current state of this project.

Contents

1 Introduction

1.1 Background of the Project

This project was started in order to prove the hypothesis that it is possible for students to work to produce meaningful products that could be used in the *development* and *enhancement* of the learning experience of their peers.

→ Section 1.2

You can find detailed information about the project goals in Section 1.2.

During initial discussions of the project's focus, there came a consensus of producing a companion application to be used by students in order to *identify*, *label*, and *display* appropriate information based on machinery used within the Collaborative Teaching Laboratory through the use of *augmented reality* (AR).

Developer: The student chosen to lead initial development of the project was Jack Shilton, a second year student enrolled in the BSc Computer Science course at the University of Birmingham.

They would act as the lead developer for this project during it's conception and development to a *minimum viable product* (MVP). During this they would work to streamline the process of expanding the application, as to ensure it could be properly maintained and developed, after the MVP has been completed.

Managers: Overseeing the project was Edwin Kyi, the Multimedia Learning Experience Developer at the CTL, and Dr Christopher Avins. They would work to maintain and develop the product after the MVP has been finalised.

1.2 Project Objectives

The final product that is to be created from this project will take the form of an application that can be run from a mobile device or tablet. This application would have a multitude of AI models that have been trained to identify components of a specific machine that is present within the lab.

The requirements for this application are as follows:

- Design should make the app easy to use.
- Different models should be able to be selected and switched to from within the application.
- Models should be trained to such a standard that there is no difficulty in them recognising the components.
- Object recognition should be implemented in real-time, there should be minimal delay in processing the video from the camera.
- Information given about components should be easily accessible, clear, and concise.
- Should require no knowledge of AI in order to use.

Using this, students should be able to approach various machines in the CTL and learn how to safely configure and operate it without the assistance of one of the lab staff. [1]

[1] This application is **not** designed to be a full replacement for lab staff, but instead a complimentary component of teaching. This should be made clear.

2 AI Training

2.1 Overview

When creating a model there are certain steps that we need to take. These can be split into the following processes:

1. Dataset Collection
2. Image Labelling
3. Model Training
4. Evaluation

We will step through each of these processes and provide in-depth information on how the perform these, as well as best practices.

2.2 Dataset Collection

This process begins by selecting a machine that you wish to add to the application.

2.2.1 Component Selection

At this stage, you should decide how you wish to split the machine into its components. When doing this there are certain features that you want to have:

- Distinct silhouette
- Clear shapes
- Unique from other components
- Avoid reflections and transparency

Following these guidelines, here is an example of two *good* components and two *bad* components:

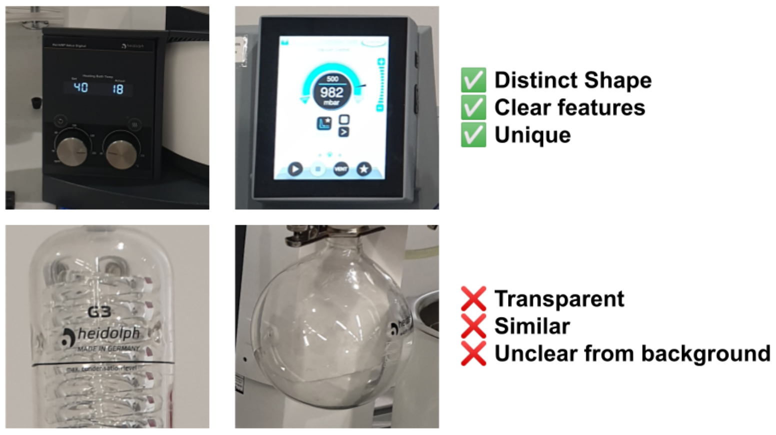

Figure 1: Examples of good and bad component selections

2.2.2 Taking Photos

When collecting photos that are to be used for training a model you want to be able to create a good variety of images, whether this be different angles, distance, lighting, states, etc. [2] The more variety you have in your initial dataset, the better the final model will be in identifying these components.

Selecting a dataset size will vary from one machine to the next, the more images that are available to train off of the better the final result, however that will also mean that more time is spent in labelling. A general rule of thumb for this is $\sim$ 100 pictures per object, however getting more images is always beneficial.

2.3 Image Labelling

`labelImg` [3] is used to label images from the dataset. This process is the most time consuming, taking approximately an hour per 100 images per person.

2.3.1 Workflow

With every image the components must have a RectBox drawn around them and provided with the associated label. The workflow should move as follows:

1. Open image directory
2. Select "Create RectBox"
3. Highlight area
4. Select/Create associated label
5. If more objects, return to 2
6. Save (Ctrl + S)
7. Move to next image (A for previous image, D for next image)
8. Repeat until complete

Ensure export type is **PascalVOC**

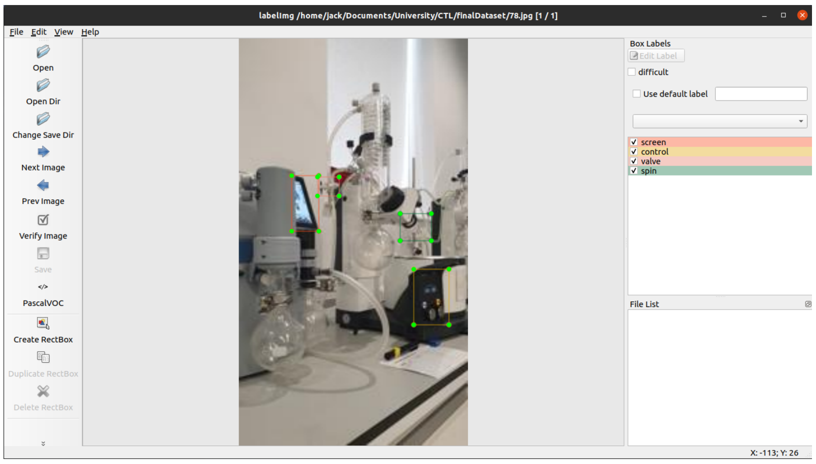

Figure 2: Example of labelled image in labelImg

[2] A high-quality camera is **not** required, nowadays most phone cameras are sufficient.
[3] tzutalin/labelImg - GitHub

2.3.2 Criteria

Ensure all labels match the following criteria:

- Highlighted areas are accurate
- Object is not obscured
- Angle is not too sharp
- Item is easily identifiable (e.g. If item is seen through a transparent object, do not highlight)
- Given label is correct

2.4 Model Training

2.4.1 Dataset Processing

To start training the dataset should be split into their respective categories, e.g. All of the `Spin` images should be placed into a folder named `"spin"`. There is no need to split them into **training** and **validation**, like the previous version, as this is now handled programmatically by TensorFlow. Therefore, the final dataset should look as follows:

```
modelData
    >> spin      >> 2-138.jpg
                 >> 6-729.jpg
                 >> 7-663.jpg
                 >> ...

    >> screen    >> 3-309.jpg
                 >> 6-727.jpg
                 >> 8-574.jpg
                 >> ...
    >> ...
```

This can be performed by running the following script:

```python
import sys
import re
import os
import csv
from PIL import Image

def getNum(target, content):
  tempString = re.findall(r'<' + target  + '>[0-9]*</' + target + '>', content)
  tempFinal = re.findall(r'\d+', tempString[0])
  return list(map(int, tempFinal))

def getString(target, content):
  tempString = re.findall(r'<'+target+'>(.*?)</'+target+'>', content, flags=re.S|re.M)
  return tempString

def genLine(type, location, name, point1, point2):
  return type+","+location+","+name+","+str(point1[0])+","+str(point1[1])+",,,"+str(point2[0])+","+str(point2[1])+",,"

path = sys.argv[1]
total = len(os.listdir(path))
lines = []
count = 0
for filename in os.listdir(path):
  if not filename.endswith(".xml"):
    continue
  print(filename)
```

```
27   f = open(path + "/" + filename)
28   content = f.read()
29 
30   width = getNum('width', content)[0]
31   height = getNum('height', content)[0]
32 
33   objects = getString('object', content)
34   objectList = []
35 
36   for x in objects:
37     name = getString('name', x)[0]
38     xmin = getNum('xmin', x)[0]
39     ymin = getNum('ymin', x)[0]
40     xmax = getNum('xmax', x)[0]
41     ymax = getNum('ymax', x)[0]
42     final = [name, xmin, ymin, xmax, ymax]
43     objectList.append(final)
44 
45   for x in objectList:
46     if not os.path.exists(x[0]):
47       os.makedirs(x[0])
48     count += 1
49     img = Image.open(path + "/" + filename.replace(".xml",".jpg"))
50     img = img.transpose(Image.ROTATE_270)
51     img = img.crop((x[1],x[2],x[3],x[4]))
52     img.save(str(x[0]) + "/" + filename.replace(".xml","-"+str(count)+".jpg"))
```

Jack-Development/CTLMachineLearning/Scripts/cropImages.py - GitHub

If the images are not being output properly, remove line 50, this issue is caused by some systems, which will automatically rotate an image if they contain certain metadata.

NOTE: In this format, the images are labelled in the following format: [FileName]-[LabelCount].jpg and the folders will be output in the directory of execution.

2.4.2 TensorFlow Lite Model Maker

Now the dataset is prepared, it can be fed into the script for the `TensorFlow Lite Model Maker`.

```
import os
import sys

import numpy as np

import tensorflow as tf
assert tf.__version__.startswith('2')

from tflite_model_maker import model_spec
from tflite_model_maker import image_classifier
from tflite_model_maker.config import ExportFormat
from tflite_model_maker.config import QuantizationConfig
from tflite_model_maker.image_classifier import DataLoader

import matplotlib.pyplot as plt

image_path = sys.argv[1]
data = DataLoader.from_folder(image_path)
train_data, test_data = data.split(0.9)

model = image_classifier.create(train_data, batch_size=8, epochs=100)
loss, accuracy = model.evaluate(test_data)
```

```

model.export(export_dir=".")
```

Jack-Development/CTLMachineLearning/Scripts/trainModel.py - GitHub

The settings for the model can be edited by altering line 21, previously the best results have been found by setting the epochs to 100 and the batch size to 8, training using the raw high quality pictures.

During the process of the script the console will output information on the progress of the model and then the evaluation of the model, once it has completed.

3 Implementation

3.1 Android Studio Project

The current version of the project is hosted at Jack-Development/ CTL-MachineLearning - GitHub. This repository contains all the required files for creating the .apk file for the application, as well as the ability to add custom models.

The recommended IDE for developing with this project is Android Studio.

3.2 Model Preparation

Folder Formatting

Once the model has been trained, it should be formatted so it corresponds with the requirements of the program. This is done in the following way:

```
[modelName]
    >> images   >> [label1].jpg
                >> [label2].jpg
                >> [label3].jpg
                >> ...
    >> labels.json
    >> model.tflite
```

In this format, "[modelName]" should be replaced with the name that the model will be referred to as in all other elements of the program. "[label1]", "[label2]", etc, should correspond to each of the label names that were created in `LabelImg`. "model.tflite" should be the model that was trained and should be renamed to "model.tflite". An example implementation of this formatting can be seen as follows:

```
rotaryEvaporator
    >> images   >> control.jpg
                >> power.jpg
                >> screen.jpg
                >> spin.jpg
                >> valve.jpg
    >> labels.json
    >> model.tflite
```

labels.json

This file should provide the details for each label, which is split into three components `shortName`, `name`, and `description`. `shortName` should correspond to the label name that was created previously in `LabelImg`. `name` refers to the title that will be displayed on the information screen. Finally, `description` will be the information that is shown on the same screen. An example of a labels.json file is as follows:

```
[
  {
    "shortName": "spin",
    "name": "Spin",
    "description": "This is the spinning component, it spins very well."
  },
  {
    "shortName": "control",
    "name": "Controller",
    "description": "This does the controlling, it controls the stuff."
  },
  {
    "shortName": "valve",
    "name": "Valve",
    "description": "This is a valve, it does valving."
  },
  {
    "shortName": "screen",
    "name": "Screen",
    "description": "This is a screen. It shows information."
  },
  {
    "shortName": "power",
    "name": "Power",
    "description": "This is the power button. Have you tried turning it off and on again?"
  }
]
```

3.3 Application Importing

In order to add this new model to the application, simply add the previously created folder into the assets folder of the project, and edit the `setup.json` to implement the model and the threshold it requires. A sample for how the file should be formatted is as follows:

```
[
  {
    "modelName": "rotaryEvaporator",
    "threshold": "0.2"
  },
  {
    "modelName": "rotaryEvaporator",
    "threshold": "0.65"
  }
]
```

Within this file the `modelName` should be the same as the name of the folder that was added to the assets folder. The `threshold` value should be tweaked and set based on observations of the model performance, in this case the 0.65 shown here is the production threshold and a 0.2 option has been added as a presentation option in order to show the application without the need for a machine on sight.

With each added option to `setup.json` a new option will be shown on the main menu of the application, and can be selected within the application.

4 APK Upload

4.1 Developer Options

In order to upload an `.apk` to an Android device, a phone with `USB debugging` enabled is required. This can be done with any Android device, instructions on how to do so can be found here.

4.2 Uploading

Once the application is prepared, plugging the Android device into the computer will cause the device to request permission for `USB debugging`, it will need to be accepted to continue.

This should cause the device to appear as a platform to run the application in `Android Studio`. When the application is started in the IDE it should build the `.apk`, upload it to the device, then begin the process.

The device can now be disconnected from the computer and used independently, however the terminal output will only be available when plugged in to the computer.

5 Debugging

In order to help fix issues that may appear in tracking and identification, a debug mode was implemented.

To toggle the debug mode, the variable will need to be changed in the `ModelSelect.java` file, specifically here.

Editing this variable will propagate through all the other files and will change how the application functions. These are all the changes that can be expected:

- Model selection will display threshold
- Camera will show current model name and threshold
- Bounding boxes will display confidence

Appendix

A Additional Comments

A.1 Google Colab

Google Colab is designed for research and provides access to high performance resources at no cost. This is due to the fact that these resources are not unlimited or guaranteed.

The use of Google Colab was considered for this project, and at one point was a core component of the main workflow, however, with the switch to the TensorFlow Lite Model Maker, processing time was reduced massively. Due to this, this workflow can be performed on most computers.

However, if the dataset is too large and time is an issue, then it may be beneficial to use Google Colab to attempt to reduce the processing time.

Index