Note: This is the user manual for the latest version at the time of the paper submission. For the most updated user manual and video tutorials, please visit: https://smartctsc.github.io/SmartCS

Homepage:

What type of citizen science mobile app do you want to create?

Select from the options below

App for Object Detection

App for Object Classification

Mobile apps created/under development using this platforms

RipSnap

A citizen science mobile app for real-time rip current detection and data collection.

Demo video link

Website: [here](#)

Vehicle Object Detector
Object Detection App Creation:

Step 1: Label Your Data
- Create and Label Training Dataset
- Label your training data following the instruction in this link.

Step 2: Train Your Model
- Train Model on Cloud Notebook
  - Option 1: Train your model on the cloud using a Google Colab notebook.
- Train Model on Local or Virtual Machine
  - Option 2: Train your model on your personal computer or virtual machine on a cloud computing service following the instructions.

Step 3: Create Your App
- Select iOS App template
- Select Android App template
- Download the code for the app and build it following the instructions in this link.

Data Labeling:

Label Data

Select a labeling tool to label data
- makesense.ai
  - makesense.ai is a free web-based tool for image annotation
- Labeling
  - Labeling is a free and open-source graphical tool for image annotation on a local machine
Model Training:

Train Model using Colab

Select a model to train

- EfficientDet-Lite (v0/v1/v2)
- SSD-Mobilenet (v1/v2)

App Template Selection and Creation:

iOS App Templates

Select a template for your app from the template gallery below

All templates has instructions about how to use and customize

Template 1

Template 2
SmartCS App Studio Desktop Version
Overview (Anonymized)

Main Menu:

![Main Menu Image]

Main Menu - Object Detection App:

![Main Menu Object Detection Image]

**Step 1: Label your data**
- Import image files
- Create label list
- Label image files

**Step 2: Train your model**
- Create .tfrecord datafiles
- Select and Train Model
- Convert to mobile model

**Step 3: Create your app**
- Build Android App
- Build iOS App (macOS only)
Labeling Tool (Labellmg):

- Click to open the “images” directory (either from “train” or “test”)
- Click to select the “annotations” directory (from “train” or “test”, as selected previously)
- Use PascalVOC format

Output Console (Training Progress):

```
I0913 16:17:33.826492 4539039232 learning.py:512] global step 2267
  : loss = 3.6457 (6.282 sec/step)
I0913 16:17:40.187733 4539039232 learning.py:512] global step 2268
  : loss = 4.8808 (6.288 sec/step)
I0913 16:17:46.391788 4539039232 learning.py:512] global step 2269
  : loss = 4.2819 (6.282 sec/step)
INFO:tensorflow:global step 2270: loss = 3.2933 (6.934 sec/step)
I0913 16:17:53.326368 4539039232 learning.py:512] global step 2270
  : loss = 3.2933 (6.934 sec/step)
INFO:tensorflow:global step 2271: loss = 4.9687 (6.716 sec/step)
I0913 16:18:00.043579 4539039232 learning.py:512] global step 2271
  : loss = 4.9687 (6.716 sec/step)
INFO:tensorflow:global step 2272: loss = 2.6909 (6.515 sec/step)
I0913 16:18:06.559284 4539039232 learning.py:512] global step 2272
  : loss = 2.6909 (6.515 sec/step)
  : loss = 3.4397 (6.791 sec/step)
INFO:tensorflow:global step 2274: loss = 2.8017 (7.114 sec/step)
  : loss = 2.8017 (7.114 sec/step)
```
App Templates:

a) iOS – portrait mode
b) iOS – landscape mode
c) Android – portrait mode
d) Android – landscape mode
SmartCS App Studio Quick Start Guide
for Object Detection App Creation

How to use the desktop version?
Download the desktop version from https://smartctsc.github.io/SmartCS as a
Unzip and go to the main directory.
Setup the software by running python setup.py command
Run the software by running python run.py command
Select Object Detection App from the Top Menu.

It’ll open the object detection app creation menu. The results of running the app will be shown in the terminal/console.
Step 1

Select **Import image files** to import the images you want to use for the Machine Learning (ML) model training. You can use this option multiple time to import images from different locations.

You can extract images from multiple video files as well. You can define the step size of the frames to be extracted.

Images will be imported to a directory named **images** under the **dataset** directory.
Once the image importing is done, use the **Create label list** button to create the list of the exact name of the labels to be used. This will update the label map (list) in multiple files that’ll be used in different stages of the app creation process. Enter the label names separated by commas.

After creating the labelmap, select the **Label image files** to label the images by using an integrated labeling tool named labelImg. labelImg will automatically open your imported images and load the labelmap.
Label the imported images by drawing bounding boxes around the objects you want your ML model to learn to detect. Save the images using the PascalVOC format so that the labels are saved as .xml. The annotations will be saved in a directory named **annotations**.

**Step 2**

Select the **Create .tfrecord datafile** option to create datafiles for training. The main dataset will be split into three parts, train, test, and eval, with a defined ratio, which you can change.

![Dataset split ratio]

After the datafiles are created, you can select a ML model from a given list and run the training. The training will stop after certain number of steps or based on the loss function value, which you can define.

![Select ML model]

ML model can be decided based on your requirements and the information provided in the table below.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Inference Speed (ms)</th>
<th>mAP for COCO objects</th>
<th>TFLite model size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD MobileNet v1</td>
<td>48</td>
<td>29.1</td>
<td>5</td>
</tr>
<tr>
<td>SSD MobileNet v2</td>
<td>39</td>
<td>28.2</td>
<td>5</td>
</tr>
<tr>
<td>EfficientDet D0</td>
<td>39</td>
<td>33.6</td>
<td>6</td>
</tr>
<tr>
<td>EfficientDet D1</td>
<td>54</td>
<td>38.4</td>
<td>8</td>
</tr>
<tr>
<td>EfficientDet D2</td>
<td>67</td>
<td>41.8</td>
<td>11</td>
</tr>
</tbody>
</table>
Once the training is completed, the Convert to mobile model option can be selected to convert the trained model to .tflite format. It’ll be automatically copied to the proper location of the default app templates.

A GUI-based utility is provided to test the trained model on your PC before adding it to the smartphone app. You can use your webcam or pre-recorded videos for the testing.

**Step 3**

In step 3, you can choose to create your app for iOS or Android platform, or both. Selecting each option will open the default templates with either XCode (only on MacOS) or Android Studio. You can download other templates from this link (link removed for anonymity) and replace the default templates.
Each template has more guidelines about how to customize and use the templates. Various GUI components, such as, icon, banner, logo, UI color, etc., can be changed.
How to use the web version?

The web version can be used by loading the website (link removed for anonymity) using any standard web browser. Instructions for using the website is provided there on each of the steps.
Optional ML model training Guide for Advanced Users

**Note:** No programming/coding skills are needed for these (optional) steps. However, advance computer usage skill or experience may require for successfully complete these steps.

These steps are recommended to be used with the GitHub repo [https://github.com/SmartCtSc/CitizenScienceTFL](https://github.com/SmartCtSc/CitizenScienceTFL) or creating the directory structure

1. **Dataset creation and annotation**

Datasets are labeled using a tool called [labelImg](https://smartctsc.github.io/SmartCS). Detailed instructions can be found here: [https://smartctsc.github.io/SmartCS](https://smartctsc.github.io/SmartCS)

This repository has no dataset included in it. You need to put your labeled dataset (from step 1) in the "train" and "test" directory under the "dataset" directory. You also need to update (using a text editor) the "label_map.pbtxt" file in the dataset "directory" with the id and name of your classes.

For example, if you have three classes named apple, orange, and banana, then the "label_map.pbtxt" file should look like,


**(Optional) Recommended preinstallation setup**

Install miniconda
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
Create a new virtual environment for tensorflow 1

conda create -n tf1 python=3.7
conda activate tf1

2. Setting up Tensorflow and Dependencies

Install tensorflow 1.15

pip install tensorflow==1.15
(Optional) If you have GPU, you can install Cuda, CudNN and tensorflow-gpu to run the training much faster

conda install cudatoolkit=10.0
conda install cudnn=7.3.1
pip install tensorflow-gpu==1.15
Install these dependencies:

pip install numpy==1.19
pip install Cython
pip install contextlib2
pip install pillow
pip install lxml
pip install jupyter
pip install matplotlib
pip install tf_slim
pip install pycocotools
pip install scipy
Install Protobuf appropriate for your operating system: https://grpc.io/docs/protoc-installation/

Clone the object detection models repository

git clone https://github.com/tensorflow/models.git
Compile Protobuf from "models/research" folder

protoc object_detection/protos/*.proto --python_out=. 
In the \models\research\slim directory run

python setup.py build
python setup.py install
Update PYTHONPATH variable:

export PYTHONPATH=$PYTHONPATH:/research:/research/slim' OR export PYTHONPATH=$PYTHONPATH:pwd:pwd/slim
Install object detection. From within "models/research/" folder
cp object_detection/packages/tf1/setup.py
python -m pip install .
Test the installation of object detection API by running the command below from "models/research" folder

python object_detection/builders/model_builder_tf1_test.py

3. Generating TFRecord

First, run the following commands in the dataset directory to generate "test.csv" and "train.csv" respectively.

python xml_to_csv_test.py
python xml_to_csv_train.py

Update the generate_tfrecord.py file at line 3 with your class names. For example, if you have three classes named apple, orange, and banana, then it should look like,

```python
def class_text_to_int(row_label):
    if row_label == 'apple':
        return 1
    if row_label == 'orange':
        return 2
    if row_label == 'banana':
        return 3
    else:
        return None
```

Then, generate the TFRecord files by running the following Python script inside the dataset directory,

```bash
generate_tfrecord.py --csv_input=train.csv --output_path=train.record --image_dir=train/images
generate_tfrecord.py --csv_input=test.csv --output_path=test.record --image_dir=test/images
```

4. Selecting a Pre-trained model

We used a pretrained model as our initial checkpoint ssd mobilenet v2. It can be downloaded from [http://download.tensorflow.org/models/object_detection/ssd.mobilenet_v2.quantized_300x300.coco.2019_01_03.tar.gz](http://download.tensorflow.org/models/object_detection/ssd.mobilenet_v2.quantized_300x300.coco.2019_01_03.tar.gz)

Extract the .tar.gz file to the folder named "pretrained_model". (It should be pretrained_model/ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03)

Update the "pipeline.config" in inside the extracted folder as follows,

- update the number of classes in line 3
- update the paths of the pretrained model in line 157

- update the path of the dataset in line 162, 164, 174, 178

```python
157    fine_tune_checkpoint = "../pretrained_model/ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03/model.ckpt"
158    from_detection_checkpoint = true
159    num_steps = 20000000
160 }
161 train_input_reader {
162    label_map_path = "../dataset/label_map.pbtxt"
163    tf_record_input_reader {
164        input_path = "../dataset/train.record"
165    }
166 }
167 eval_config {
168    num_examples = 8000
169    metrics_set = "coco_detection_metrics"
170    use_averaging = true
171    include_metrics_per_category = true
172 }
173 eval_input_reader {
174    label_map_path = "../dataset/label_map.pbtxt"
175    shuffle = false
176    num_readers = 1
177    tf_record_input_reader {
178        input_path = "../dataset/test.record"
179    }
180 }
```

(Tip: you can use an editor such as Atom, VS Code, etc to see the line numbers.)

5. Training the model

Train the model using the pretrained model as our initial checkpoint. Use the trained_model directory as the training folder. Run the following script inside "models/research/object_detection/legacy/" directory.

```
python train.py --logtostderr --train_dir=<path to "trained_model"> --pipeline_config_path=<path to pipeline.config file>
```

Run the training until converge and then run the checkpoint for the next step. It is considered by many literatures that the model converged when the loss is below 2.

6. Converting the model to .tflite

Convert the TensorFlow model and generates a TensorFlow Lite model using the instruction from the link below https://www.tensorflow.org/lite/convert

For example, if you are using the 5000th checkpoint, run the following command from "models/research/object_detection/" directory.
python export_tflite_ssd_graph.py --pipeline_config_path=<path to pipeline.config file> --trained_checkpoint_prefix="../trained_model/model.ckpt-5000" --output_directory="../trained_model/tflite" --add_postprocessing_op=true

run the following command from "trained_model" directory

tflite_convert \
--graph_def_file=tflite/tflite_graph.pb \n--output_file=tflite/model.tflite \n--output_format=TFLITE \n--input_shapes=1,300,300,3 \n--input_array=normalized_input_image_tensor \n--output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1','TFLite_Detection_PostProcess:2','TFLite_Detection_PostProcess:3' \n--inference_type=QUANTIZED_UINT8 \n--mean_values=128 \n--std_dev_values=127 \n--change_concat_input_ranges=false \n--allow_custom_ops

This command will create a file named "model.tflite" in a directory named "tflite". Here, create a text file with your class names in separate lines and name it "labelmap.txt". For example, if you have three classes named apple, orange, and banana, then the "labelmap.txt" file should look like,

apple
orange
banana

Now, run the following command to include the labelmap as metadata with your .tflite file,

python metadata_writer.py

It'll create a file named "detect.tflite"

7. Building and running the Android/iOS app with the .tflite.

Add an additional line "???" to your "labelmap.txt". For example,

???
apple
orange
banana

Now, use your "detect.tflite" and "labelmap.txt" with your Android/iOS app development project in Android Studio or Xcode to build and run the app following the instructions here,

https://github.com/SmartCtSc/CitizenScienceTFL