

## Content

|   |    |
|---|----|
| Revision history .....                                    | 3  |
| Model training.....                                       | 4  |
| Environment prerequisites.....                            | 4  |
| Prepare darknet application & Download YOLO-Fastest ..... | 4  |
| Dataset preparation (from COCO dataset).....              | 5  |
| Download COCO dataset.....                                | 5  |
| Train & valid image list file .....                       | 6  |
| Annotation file .....                                     | 7  |
| Class description file .....                              | 8  |
| training material list file .....                         | 8  |
| Model configuration file .....                            | 8  |
| Final readiness COCO dataset for YOLO .....               | 10 |
| Start training .....                                      | 11 |
| Test model.....   | 12 |
| Model conversion .....                                    | 13 |
| Convert weight to Keras model.....                        | 13 |
| Convert Keras model to TensorFlow Lite model .....        | 13 |
| Integrate model to WE2.....                               | 14 |
| Prerequisite .....  | 14 |
| Vela Compilation .....                                    | 14 |
| Convert to C Byte Array.....                              | 14 |
| Add code to SDK.....                                      | 14 |
| Yolo_coco_vela.cc .....                                   | 15 |
| app_main.mk .....   | 15 |
| app_algo.cc .....   | 15 |
| Linker file.....  | 17 |
| Reference .....   | 19 |
| Network(model).....                                       | 19 |
| Yolo (you only look once) official .....                  | 19 |
| Tensorflow.....   | 19 |
| Himax Yolo-Fastest .....                                  | 19 |
| Keras-YOLOv3-model-set.....                               | 19 |
| Dataset reference.....                                    | 19 |
| COCO dataset .....  | 19 |
| Tools .....   | 19 |
| COCO_to_YOLO annotation translator .....                  | 19 |

|              |    |
|--------------|----|
| Netron ..... | 19 |
|--------------|----|

# Revision history

| Version | Description                                     |
|---------|---|
| v1.0    | Initial version                                 |
| v1.1    | Update training process by COCO person dataset. |

# Model training

This section introduces how to train a model with COCO dataset based on *Yolo-Fastest*. You can use different dataset to train your model accordingly.

**<Note> WE-I has very limited flash, please consider reducing model size like decreasing input image resolution. The Himax YOLO COCO model uses 160\*160 resolution with only 1 class(person), more model examples (model\_zoo/\*.cfg) can be found in HIMAX Yolo-Fastest Github.**

## Environment prerequisites

|             |       |
|-------------|-------|
| Ubuntu:     | 22.04 |
| Python:     | 3.8.5 |
| OpenCV:     | 4.xx  |
| Tensorflow: | 2.4   |
| CUDA:       | 11.0  |
| CUDNN:      | 8.0   |

**<Note> We suggest you use GPU to make sure overall training is completed successfully. For the environment of Tensorflow, please check [https://www.tensorflow.org/install/source\\_windows?hl=zh-tw](https://www.tensorflow.org/install/source_windows?hl=zh-tw) for more details.**

## Prepare darknet application & Download YOLO-Fastest

1. Download **darknet** repository.  
\$ `git clone https://github.com/AlexeyAB/darknet.git`
2. Modify makefile.  
\$ `cd Yolo-Fastest`  
\$ `vi makefile`  
Change settings based on your environment.  
`GPU=1`  
`CUDNN=1`  
`CUDNN_HALF=0`  
`OPENCV=1`

```
AVX=0
```

```
OPENMP=0
```

```
LIBSO=0
```

**<Note>** This guide uses GPU darknet to accelerate training, you may need to install corresponding CUDA, CUDANN to meet GPU darknet requirement.

3. Build darknet application

```
$ make
```

If no error during building, you can get **darknet** application under the same folder.

4. Download Yolo-Fastest repository for following training tasksd.

```
$ git clone https://github.com/HimaxWiseEyePlus/Yolo-Fastest.git
```

Now we have **darknet** application and **Yolo-Fastest** environment. The next step, let's prepare training dataset from COCO with YOLO format.

## Dataset preparation (from COCO dataset)

### Download COCO dataset

Download the COCO dataset from this [link](#). Please download

**2017 Train/Val images** [18GB, 1GB] and

**2017 Train/Val annotations** [241MB].




### Images

2014 Train images [83K/13GB]  
2014 Val images [41K/6GB]  
2014 Test images [41K/6GB]  
2015 Test images [81K/12GB]  
2017 Train images [118K/18GB]  
2017 Val images [5K/1GB]  
2017 Test images [41K/6GB]  
2017 Unlabeled images [123K/19GB]

### Annotations

2014 Train/Val annotations [241MB]  
2014 Testing Image info [1MB]  
2015 Testing Image info [2MB]  
2017 Train/Val annotations [241MB]  
2017 Stuff Train/Val annotations [1.1GB]  
2017 Panoptic Train/Val annotations [821MB]  
2017 Testing Image info [1MB]  
2017 Unlabeled Image info [4MB]

Extract them and the contents look like:

| Name   |
|--|
|  annotations_trainval2017 |
|  train2017                |
|  val2017                  |

The content structure looks like:

```
-- annotations_trainval2017 # annotation file for train & valid images
-- annotations
-- instances_train2017.json # annotations of train images
-- instances_val2017.json # annotations of valid images
-- train2017 # train images
-- xxxxxxxxxxxxxxxx.jpg
-- val2017 # valid images
-- xxxxxxxxxxxxxxxx.jpg
```

We're going to train with person images only, select images having person and put them into train & valid folder respectively in **COCO\_dataset**.

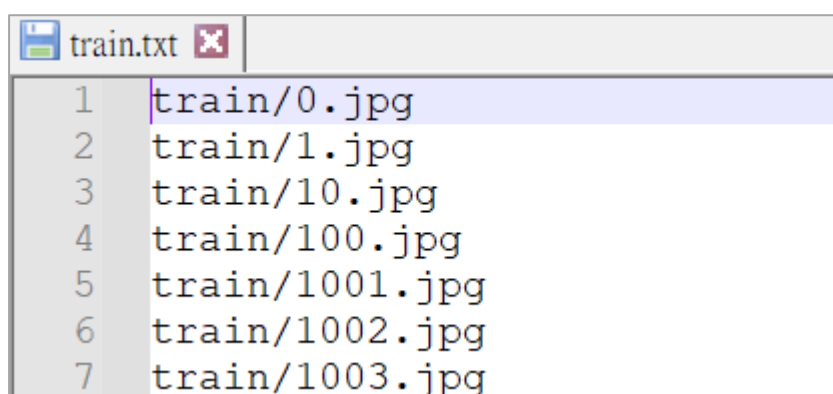
```
-- COCO_dataset
-- train
-- valid
```

## Train & valid image list file

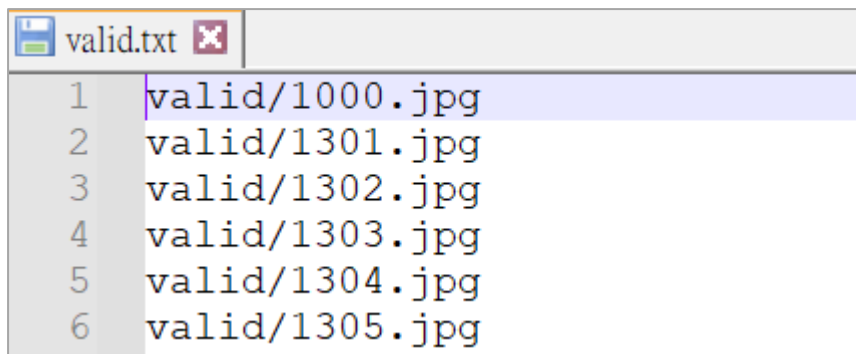
For train & test dataset, you should prepare two files **train.txt** & **valid.txt** to describe image path belonging to them. The format should follow the rule:

**One line for one image path.** e.g.

**train.txt**



**valid.txt**



**<Note>** The path defined by your file locations.

### Annotation file

Annotation files (\*.txt) describes object's bounding box (**object class & bounding box position**) on each image, which should follow YOLO format:

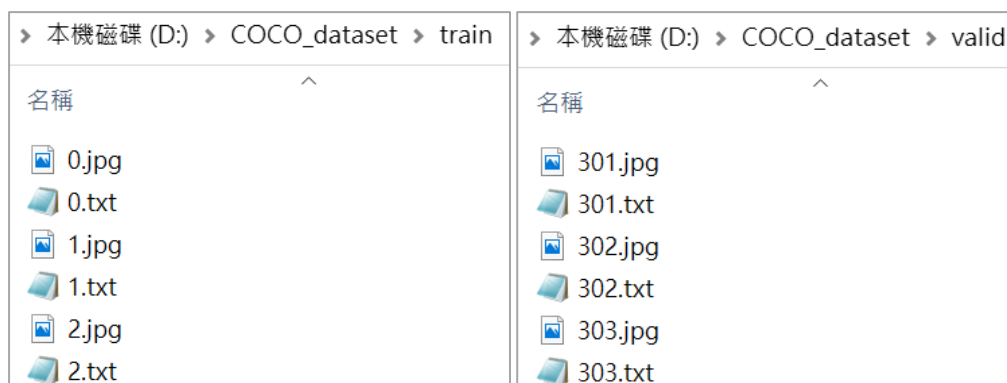
**<obj\_class> <x\_center> <y\_center> <width> <height>**

**<Note>** All values are separated by space.

e.g.

```
0 0.549219 0.483333 0.123438 0.204167
0 0.547656 0.468750 0.129688 0.216667
0 0.550000 0.484375 0.131250 0.214583
```

Each image has own annotation file with **the same name**. Make sure that images and their annotation files are in the same folder and **one image has its annotation txt file**. The result should look like:

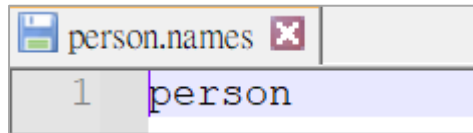


**<Note>** Some [python script](#) can help converting from raw COCO annotation file(.json) to each image's annotation file(.txt).

## Class description file

Copy file from `Yolo_Fastest/Modelzoo/yolo-fastest-1.1_160_person/person.names` to `COCO_dataset` folder.

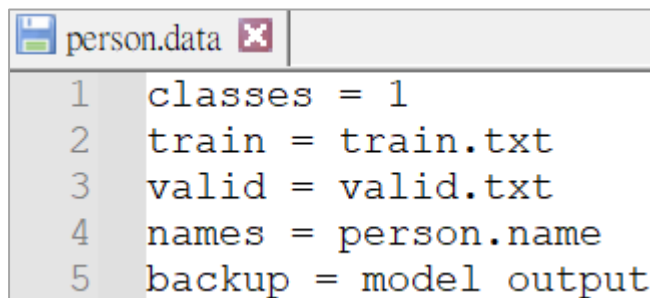
This file describes class mapping. This tutorial only uses 1 class(person).



**<Note>** The original COCO dataset has 80 classes.

## training material list file

Copy file from `Yolo_Fastest/Modelzoo/yolo-fastest-1.1_160_person/person.data` to `COCO_dataset` and modify it with your configurations.



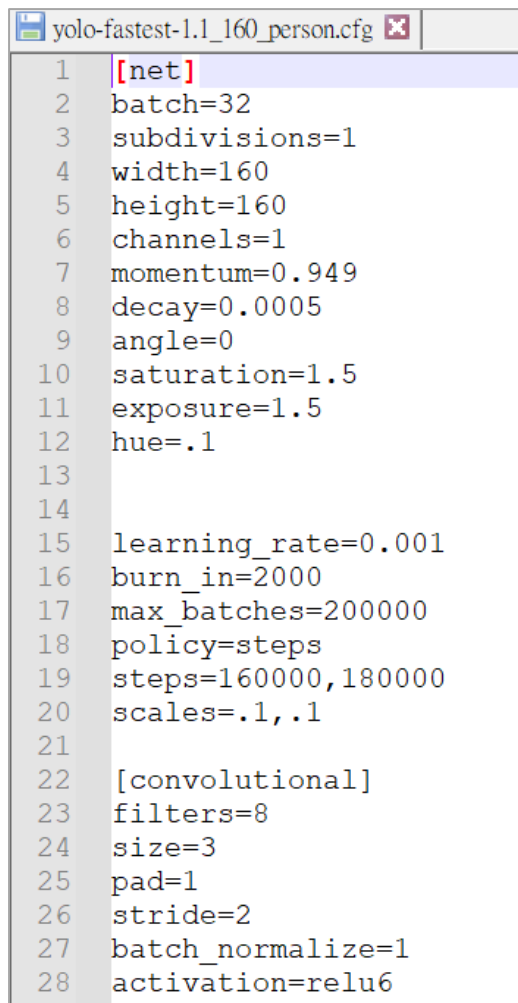
`classes`    *# number of class*  
`train`      *# train image list file*  
`valid`      *# valid image list file*  
`names`      *# class description file*  
`backup`     *# Folder to put trained model*

## Model configuration file

This file describes model configurations & structure. Copy file from `Yolo_Fastest/Modelzoo/yolo-fastest-1.1_160_person/yolo-fastest-1.1_160_person.cfg` to `COCO_dataset`.

The model configuration file looks like:





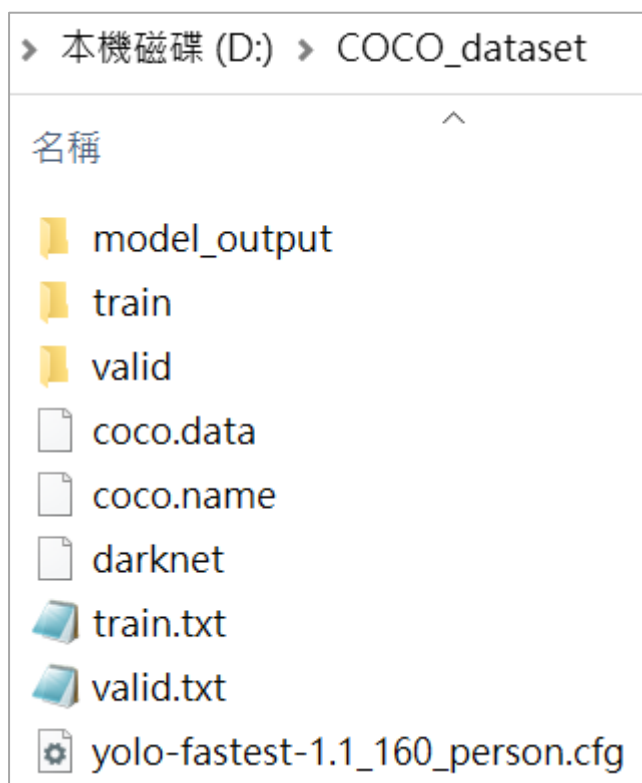
```
1 [net]
2 batch=32
3 subdivisions=1
4 width=160
5 height=160
6 channels=1
7 momentum=0.949
8 decay=0.0005
9 angle=0
10 saturation=1.5
11 exposure=1.5
12 hue=.1
13
14
15 learning_rate=0.001
16 burn_in=2000
17 max_batches=200000
18 policy=steps
19 steps=160000,180000
20 scales=.1,.1
21
22 [convolutional]
23 filters=8
24 size=3
25 pad=1
26 stride=2
27 batch_normalize=1
28 activation=relu6
```

**<Note1>** [Netron](#) can visualize model configuration file.

**<Note2>** There are more different model configuration files in the folder **Yolo-Fastest/cfg**, check them for more information.

## Final readiness COCO dataset for YOLO

After above steps, the folder **COCO\_dataset** should have the following structure. Now you're ready to start training model.



**<Note>** You can use own dataset by the same preparation steps.

## Start training

Make sure all following contents are available and follow the [structure](#). If not, go back to [Dataset preparation](#) and do it again.

1. YOLO training tool:

**darknet**

2. Images & annotation files:

**train/\*\*\*.jpg & \*\*\*.txt**

**valid/\*\*\*.jpg & \*\*\*.txt**

3. Train & Valid image list file:

**train.txt**

**valid.txt**

4. Training material list file:

**person.data**

5. Class description file:

**person.name**

6. model configuration file:

**yolo-fastest-1.1\_160\_person.cfg**

### Steps:

1. Change directory to **COCO\_dataset** and run command to start training

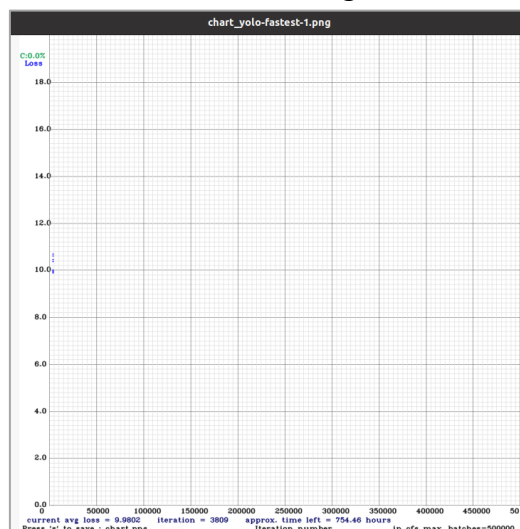
```
$ ./darknet detector train \
```

```
coco.name \
```

```
yolo-fastest-1.1_160_person.cfg \
```

```
-gpus 0
```

The training process will start, and status diagram is shown as well.



During training process, the **current** trained weight is saved periodically in `COCO_dataset/model_output/` until the end of training.

The latest trained model is named `****_last.weights`.

The final trained model is named `****_final.weights`.

## Test model

You can use **darknet** to verify trained model *by* image or video.

```
$ ./darknet detector test \  
person.data \  
yolo-fastest-1.1_160_person.cfg \  
model_output/xxxx.weights \  
test_image.jpg \  
-i 0 -thread 0.25
```

The result image will be popped up and saved as **predictions.jpg** which includes object-detection bounding box, class and confidence score. The following image is a test result.

e.g.



**<Note>** Feel free to test your model with different images/video to see if model works correctly.

## Model conversion

### Convert weight to Keras model

To deploy model into WE-I, we need to convert weight from `*.weight` to `*.h5` (Keras format) via `converter.py` @GitHub [keras-YOLOv3model-set](#).

```
$ python keras-YOLOv3-model-set/tools/model_converter/fastest_1.1_160/convert.py \  
yolo-fastest-1.1_160_person.cfg \  
model_output/xxxxxx.weights \  
model_output/yolo_coco.h5
```

**<Note>** `.h5` is Keras-format file.

### Convert Keras model to TensorFlow Lite model

Convert model from Keras to tflite format and quantization by following command:

```
$ python keras-YOLOv3-model-set/tools/model_converter/fastest_1.1_160/post_train_quant_convert_demo.py \  
--keras_model_file model_output/yolo_coco.h5 \  
--output_file model_output/yolo_coco.tflite \  
--annotation_file train.txt
```

**<Note>** `train.txt` is your image list file of train dataset.

# Integrate model to WE2

This section describes how to integrate trained model(\*.cc) to SDK.

## Prerequisite

### 1. *yolo\_coco.cc*

We got this TensorFlow Lite for microcontroller C-array file( \*.cc) from previous section. You can read [Model Training](#) to know how to get this file.

### 2. *Himax SDK*

Himax official software development kit.

## Vela Compilation

### 1. *Install Vela Module*

```
$ pip install ethos-u-vela
```

### 2. *Use Vela Compiler to compile yolo\_coco.tflite*

```
$ vela --accelerator-config ethos-u55-64 --config himax_vela.ini --system-config  
My_Sys_Cfg --memory-mode My_Mem_Mode_Parent .\yolo_coco.tflite
```

### 3. *Got the vela compiled tflite file [output/yolo\\_coco\\_vela.tflite](#)*

An vela compilation example is in `sdk_root\_Documents\vela_compile`, it shows how to convert yolo\_coco.tflite to yolo\_coco\_vela.tflite.

## Convert to C Byte Array

The final step is converting model to C byte array(\*.cc).

```
$ xxd -i yolo_coco_vela.tflite > yolo_coco_vela.cc
```

Now you're ready to deploy this model to WiseEye.

## Add code to SDK

This section uses scenario *sample\_code\_app* of SDK to explain how to deploy model step-by-step.

## Yolo\_coco\_vela.cc

Location: `\sdk_root\`

`WE2_CM55M_APP_S\app\scenario_app\sample_code_app\src\yolo_fastest\yolo_coco_vela.cc`

Replace original `yolo_coco.cc` with yours and change some declaration used in SDK as follows.

```
extern const unsigned char yolo_fastest_tflite [];  
extern const int yolo_fastest_tflite_len;  
const unsigned char yolo_fastest_tflite[]  
__attribute__((section(".tflite_model"), aligned(16))) =  
{  
    0x20, 0x00, 0x00, 0x00, 0x54,  
    ...  
};  
  
extern const int yolo_fastest_tflite_len = xxxxx;
```

**<Note>** Himax SDK use these variables by default.

## app\_main.mk

Location: `\sdk_root\`

`WE2_CM55M_APP_S\app\scenario_app\sample_code_app\`

# Set the algorithm definition.

`TFLITE_ALGO = YOLO_FASTEST`

## app\_algo.cc

Location: `\sdk_root\`

`WE2_CM55M_APP_S\app\scenario_app\sample_code_app\src\yolo_fastest\`

This the main model inference code.

### ■ Global definitions

**# Add extern declarations defined in `yolo_coco_vela.cc`.**

```
extern const unsigned char yolo_fastest_tflite[];  
extern const int yolo_fastest_tflite_len;
```

**# The tensor arena size.**

```
constexpr int tensor_arena_size = 399*1024; # 399 bytes
```

**# Modify model image input size. Himax *yolo coco* model uses 160\*160 resolution, you should change it based on your model.**

```
#define HIMAX_INPUT_SIZE_X (160)
```

```
#define HIMAX_INPUT_SIZE_Y (160)
```

**<Note>** *Increase input resolution will increase model size as well.*



## ■ *app\_algo\_init ()*

**# Get model array defined in yolo\_coco\_vela.cc**

```
model = ::tflite::GetModel(yolo_fastest_tflite);
```

**# Operation resolver setup.**

This example model uses 12 operators, make sure the value is the same as the number of operators used.

```
static tflite::MicroMutableOpResolver<1> micro_op_resolver;  
micro_op_resolver.AddEthosU();
```

## ■ *app\_algo\_run ()*

This function has main inference from image input. Generally, you don't need to do any modifications.

model\_inference:

```
TfLiteStatus invoke_status = interpreter->Invoke();
```

The major topic you need to take care is how to process the model output (**interpreter->output**). This guide provides an example to process output with 1 class and 3 anchor boxes. If you use customized model, please change code respectively to fit model output.

## Linker file

Location: *sdk\_root\*

*WE2\_CM55M\_APP\_S\app\scenario\_app\sample\_code\_app\*

*TrustZone\_S\_ONLY.ld*

When you try to integrate model into WE-I, you may find that model size is too large to put. In this case, you should try to run model in flash instead of SRAM(default) by adding code in linker file as follows.

```
#ifdef FLASH_AS_SRAM  
#if defined (TFLITE_ALGO_ENABLED)
```

```
.model : ALIGN(4)
{
    * (.tflite_model)
} > CM55M_S_APP_FLASH1
#endif
#endif
```

# Reference

## Network(model)

**Yolo (you only look once) official**

<https://pjreddie.com/darknet/>

**Tensorflow**

<https://www.tensorflow.org/install?hl=zh-tw>

**Himax Yolo-Fastest**

<https://github.com/HimaxWiseEyePlus/Yolo-Fastest.git>

**Keras-YOLOv3-model-set**

<https://github.com/david8862/keras-YOLOv3-model-set>

## Dataset reference

**COCO dataset**

<https://cocodataset.org/#home>

## Tools

**COCO\_to\_YOLO annotation translator**

Translate raw COCO dataset to YOLO format

[https://github.com/Weifeng-Chen/DL\\_tools/blob/main/coco2yolo.py](https://github.com/Weifeng-Chen/DL_tools/blob/main/coco2yolo.py)

<https://github.com/qwirky-yuzu/COCO-to-YOLO.git>

Coco2yolo.py

**Netron**

Network visualization tool

<https://netron.app/>