Face Mask Yolov4 detector - Nvidia Jetson Nano

Citations

This is a project for the Jetson Community and couldn't be possible without the effort of other developers. All the YoloV4 / Darknet code and documentation can be found here:

- Pjreddie - Darknet
- AlexeyAB/darknet

Index

Previous tips

During the development of this project I use SSH, SCP and VNC Viewer for controlling and file-transfer from my PC to Jetson Nano board. You can control it directly from the board with a keyboard and mouse but this way is more unconfortable from my point of view:

- Setup VNC server on the Jetson developer kit
- How To Use SSH To Connect To A Remote Server In Linux Or Windows
- How to Use SCP Command to Securely Transfer Files

Dataset

For this project I used the Kaggle's Face Mask Detection dataset with 853 images belonging to 3 classes. Each image has one or many bounding boxes.

The classes are:

- With mask
- Without mask
- Mask worn incorrectly

I recomend use Google Dataset Search to find any kind of dataset, in this case this dataset maybe its a bit small but I've got a good accuracy, if you want you can make it bigger with your own images or any other dataset.

Conversion to Yolo format

Yolo needs an specific notation for train the model and .jpg file format, so first of all you have to go to images folder and run:

```
$> sudo apt-get install imagemagick

$> #mogrify -format jpg *.png
# create new folder for output images
$> mkdir ../obj & mogrify -format jpg -path ../obj *.png
```

```
$> python3 xml_to_yolo.py
```

If you haven't any library just install it with pip/pip3.

After that you will have one .txt per .xml file, train.txt and test.txt, obj folder contain the same format as darknet, now just copy it into darknet/data/ (These file has a split 90/10 of the total of bounding boxes).

```
cp -r obj ../darknet/data/
cp train.txt ../darknet/data/
cp test.txt ../darknet/data/
cp obj.names ../darknet/data/
cp obj.data ../darknet/data/
```

YoloV4

All the YoloV4 code is develop by AlexeyAB/darknet, there you can find great documentation and examples about how to train, metrics, etc.

Compiling YoloV4 on Nvidia Jetson Nano

First of all you have to clone AlexeyAB repository

```
$ git clone https://github.com/AlexeyAB/darknet.git
$ cd darknet
```

Edit the Makefile with:

```
GPU=1
CUDNN=1
CUDNN_HALF=1
OPENCV=1
AVX=0
OPENMP=1
LIBSO=1
ZED_CAMERA=0
ZED_CAMERA_v2_8=0

......

USE_CPP=0
DEBUG=0

ARCH= -gencode arch=compute_53,code=[sm_53,compute_53]

......

NVCC=/usr/local/cuda/bin/nvcc
```

And run make:

```
$ make
```

Testing YoloV4 with COCO

After that the project is compiled and just need the trained weights to run it. I recommend to use Tiny-Yolo if you want a higher FPS performance. You can download both from AlexeyAB repository:

- yolov4.weights

- yolov4-tiny.weights

To run darknet just:

```
./darknet detector demo cfg/coco.data \
                          cfg/yolov4-tiny.cfg \
                          yolov4-tiny.weights \
                          -c 0
```

The `-c 0` means using the camera (V4L2) device at `/dev/video0`.

Training the Mask Detector

To train a new YoloV4-Tiny model just follow AlexeyAB steps or use my files and .weights. It takes about 20 hours to finish the 6000 steps (2000x3 classes).

To run with my trainning:

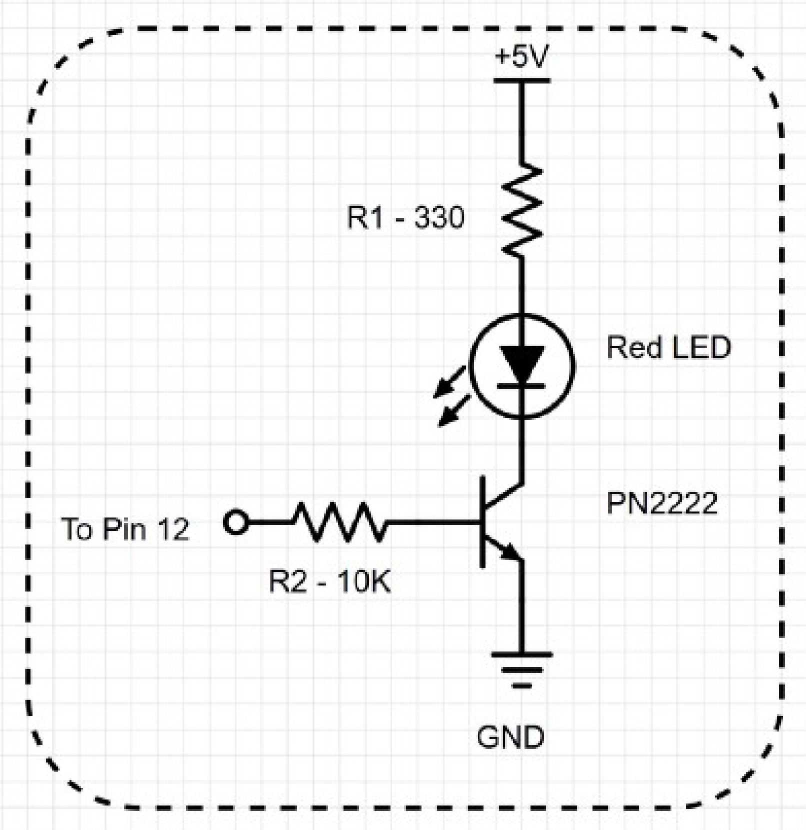
```
./darknet detector demo cfg/obj.data \
                          cfg/yolov4-tiny-masks.cfg \
                          yolov4-tiny-obj_last.weights \
                          -c 0
```

Led Control

To finish the project I wanted to use this detections to create "traffic lights", that's just a silly experiment but the possibilities are endless...

Once you have the model loaded you can run it from darknet_video.py or darknet_images.py, on this case I use darknet_images.py `import RPi.GPIO as GPIO` and added an if-else statement to control detections and set high-low values to the pins output.

The circuit I created if this one with 2 leds and 2 PN2222 transistors one for the green led and the other one for the red.



The pins are mapped this way:

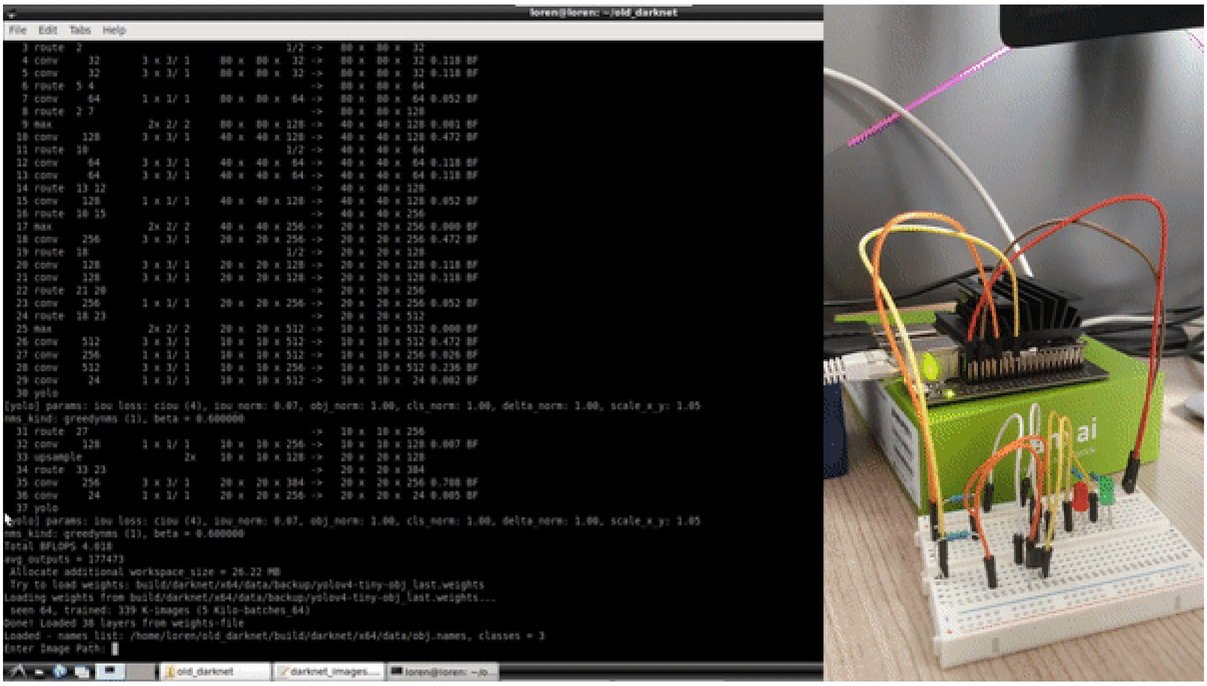
```
pinGreen =  18 #Green led -> Pin 12 on the board
pinRed = 24 #Red led -> Pin 18 on the board
```

Thats an useful image to see how BCM maps works:

Jetson Nano Dev-Board Expansion Header

Alt Function	Linux(BCM)	Board Label			Board Label	Linux(BCM)	Alt Function
DAP4_DOUT	78(21)	D21	40	39	GND		
DAP4_DIN	77(20)	D20	38	37	D26	12(26)	SPI2_MOSI
UART2_CTS	51(16)	D16	36	35	D19	76(19)	DAP4_FS
		GND	34	33	D13	38(13)	GPIO_PE6
LCD_BL_PWM	168(12)	D12	32	31	D6	200(6)	GPIO_PZ0
		GND	30	29	D5	149(5)	CAM_AF_EN
		D1/ID_SC	28	27	D0/ID_SD		
SPI1_CS1	20(7)	D7	26	25	GND		
SPI1_CS0	19(8)	D8	24	23	D11	18(11)	SPI1_SCK
SPI2_MISO	13(25)	D25	22	21	D9	17(9)	SPI1_MISO
		GND	20	19	D10	16(10)	SPI1_MOSI
SPI2_CS0	15(24)	D24	18	17	3.3V		
SPI2_CS1	232(23)	D23	16	15	D22	194(22)	LCD_TE
		GND	14	13	D27	14(27)	SPI2_SCK
DAP4_SCLK	79(18)	D18	12	11	D17	50(17)	UART2_RTS
		RXD/D15	10	9	GND		
		TXD/D14	8	7	D4	216(4)	AUDIO_MCLK
		GND	6	5	SCL/D3		
		5V	4	3	SDA/D2		
		5V	2	1	3.3V		

Live demo:



>