

CMS COOKBOOK 21–22

Table of Contents:

1. [Introduction](#)
2. [Centos7](#)
 - a. [Prerequisites](#)
 - b. [OpenCV](#)
 - c. [Cuda and Cudnn - GPU](#)
 - d. [Darknet](#)
 - i. [Without GPU](#)
 - ii. [With GPU](#)
 - e. [CSMF Project Installation](#)
3. [Ubuntu](#)
 - a. [Prerequisites](#)
 - b. [OpenCV from the Source](#)
 - c. [Cuda and Cudnn - GPU](#)
 - d. [Darknet](#)
 - i. [Without GPU](#)
 - ii. [With GPU](#)
 - e. [CSMF Project Installation](#)
4. [Application Overview](#)
5. [Training Documentation](#)
6. [Video Links](#)

1. Introduction

For the CMS experiment at CERN, Geneva a substantial number of HGCal sensor modules are manufactured at advanced laboratories across the globe. Each sensor module comprises around 675 checkpoints for visual inspection, making manual inspection practically unfeasible. In the industrial environment of manufacturing these sensor modules, this task is awfully difficult due to diverse defect appearances, ambiguous intraclass, and interclass distances. Due to recent technological advances, there has been a rise in automated visual inspections and intelligent quality assurance systems in manufacturing. In order to simplify this, we propose a deep-learning-based approach for defect diagnosis and subsequent quality assurance. [You only look once \(YOLO\)](#) is an extremely fast and accurate state-of-the-art, real-time object detection system. More information can be found [here](#).

2. CentOS7

a. Prerequisites

Requires admin privileges

```
sudo yum update -y
sudo yum groupinstall -y "Development Tools"
sudo yum install -y epel-release centos-release-scl
devtoolset-4 wget emacs vim emacs vim openssh-clients zip
python3-pip python3-setuptools hdf5 opencv opencv-devel
opencv-python
```

a. OpenCV

- i. Install the [build tools](#) and dependencies:

Requires admin privileges

```
sudo yum install epel-release git gcc gcc-c++ cmake3
qt5-qtbase-devel \
python python-devel python-pip cmake python-devel
python34-numpy \
gtk2-devel libpng-devel jasper-devel openexr-devel
libwebp-devel \
libjpeg-turbo-devel libtiff-devel libdc1394-devel
tbb-devel numpy \
eigen3-devel gstreamer-plugins-base-devel
freeglut-devel mesa-libGL \
mesa-libGL-devel boost boost-thread boost-devel
libv4l-devel
```

- ii. Clone the OpenCV's and OpenCV contrib repositories:

```
mkdir ~/opencv_build && cd ~/opencv_build
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

- iii. Once the download is complete, create a temporary build directory, and [navigate](#) to it:

```
cd ~/opencv_build/opencv && mkdir build && cd build
```

Set up the OpenCV build with CMake:

```
cmake3 -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_C_EXAMPLES=ON \
```

```
-D INSTALL_PYTHON_EXAMPLES=ON \  
-D OPENCV_GENERATE_PKGCONFIG=ON \  
-D  
OPENCV_EXTRA_MODULES_PATH=~/.opencv_build/o  
pencv_contrib/modules \  
-D BUILD_EXAMPLES=ON ..
```

The **output** will look something like below:

```
-- Configuring done  
-- Generating done  
-- Build files have been written to:  
/home/linuxize/opencv_build/opencv/build
```

- iv. Start the compilation process:

```
make -j8
```

Modify the -j flag according to your processor. If you do not know the number of cores in your processor, you can find it by typing nproc. The compilation may take several minutes or more, depending on your system configuration.

- v. Install OpenCV with:

Requires admin privileges

```
sudo make install
```

- vi. To verify the installation, type the following commands and you should see the OpenCV version.

- a. C++ bindings:

```
pkg-config --modversion opencv4
```

If this command is not able to find the path then perform the below step given

- b. Set the Path Globally

```
export  
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:xxx/opencv_build/op  
encv/build/lib
```

Here xxx will start from root directory Windows

c. Cuda and Cudnn(only for accelerating performances {GPU})

Requires admin privileges

i. **Use the following link to check compatibility between Tensorflow, cuDNN and CUDA**

1. <https://www.tensorflow.org/install/source#gpu>
2. https://www.tensorflow.org/install/source#tested_build_configurations
3. <https://stackoverflow.com/a/50622526>

ii. **Download and install the compatible CUDA version**

(<https://developer.nvidia.com/cuda-toolkit-archive>)

1. Select operating system
2. Select Architecture
3. Select Linux Flavour as centos 7/8
4. Select installer type and follow instructions
5. Additional step for ubuntu once you have installed the files of cuda

```
sudo nano ~/.bashrc
```

Add the following at the end of `~/.bashrc`

```
export PATH=/usr/local/cuda-11.2/bin${PATH:+:${PATH}}
export
LD_LIBRARY_PATH=/usr/local/cuda-11.2/lib64${LD_LIBRARY_PATH:+
:${LD_LIBRARY_PATH}}
export CUDA_HOME=/usr/local/cuda
```

b. Download cuDNN (<https://developer.nvidia.com/cuDNN>)

i. Procedure:

1. Go to: NVIDIA cuDNN home page.
2. Click Download.
3. Complete the short survey and click Submit.
4. Accept the Terms and Conditions. A list of available download versions of cuDNN displays.
5. Select the cuDNN version you want to install. A list of available resources is displayed.

ii. Installation on Centos:

1. Tar File:
 - a. Navigate to the directory containing the cuDNN tar file.
 - b. Unzip the cuDNN package

```
tar -zxvf cudnn-x.x-linux-x64-v8.x.x.x.tgz
```

- c. Link the downloaded lib64 to local environmental variables. First, open `~/.bashrc` by run the following.

```
cd ~  
sudo gedit ~/.bashrc
```

Add the following at the end of `~/.bashrc`. **Note: replace xxx with your own path.**

```
export  
LD_LIBRARY_PATH=xxx/cuda/lib64:$LD_LIBRARY_PATH
```

Save the file, and run by the following at the terminal.

```
source ~/.bashrc
```

- d. Copy the following files into the CUDA Toolkit directory

```
cd xxx/cuda/include  
sudo cp *.h /usr/local/cuda/include/  
cd ..  
sudo cp -r xxx/cuda/lib64/* /usr/local/cuda/lib64
```

2. Debian:

- a. Navigate to the directory containing the cuDNN Debian file.
b. Install the runtime library

```
sudo dpkg -i libcudnn8_x.x.x-1+cudax.x_amd64.deb
```

- c. Install the developer library

```
sudo dpkg -i libcudnn8-dev_8.x.x.x-1+cudax.x_amd64.deb
```

- d. Install the code samples and the cuDNN library documentation.

```
sudo dpkg -i libcudnn8-samples_8.x.x.x-1+cudax.x_amd64.deb
```

d. Darknet ([Repository](#))

Requirements:

- OpenCV >= 2.4

- a. Clone Darknet from <https://github.com/AlexeyAB/darknet>

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

- b. Navigate to the directory
- c. Adjust flags in the 'MakeFile' as required.

Change link number 87, 102 in **MakeFile** of darknet from

```
CFLAGS+= -DOPENCV
```

to

```
CFLAGS+= -DOPENCV -std=gnu11
```

- d. The following options can be set before using **make** :

- i. GPU=0
- ii. CUDNN=0
- iii. CUDNN_HALF=0
- iv. **OPENCV=1**
- v. DEBUG=0
- vi. OPENMP=0
- vii. **LIBSO=1**
- viii. ZED_CAMERA=0

```
cd darknet  
make
```

With GPU

Requirements:

- OpenCV >= 2.4
- CUDA >= 11.2
- cuDNN >= 8.1

- GPU with CC >= 3.0
- Clone Darknet from <https://github.com/AlexeyAB/darknet>

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

- Navigate to the directory
- Adjust flags in the 'MakeFile' as required.

Change link number 87, 102 in **MakeFile** of darknet from

```
CFLAGS+= -DOPENCV
```

to

```
CFLAGS+= -DOPENCV -std=gnu11
```

- The following options can be set before using **make** :
 - GPU=1**
 - CUDNN=1**
 - CUDNN_HALF=1**
 - OPENCV=1**
 - DEBUG=1**
 - OPENMP=1**
 - LIBSO=1**
 - ZED_CAMERA=1**

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

Example Makefile flags

Now, for example, if OpenCV is to be used, set OPENCV = 1

```
cd darknet
make
```

e. CSMF Project Installation

1. Pre-requirements
 - a. Python > 3.7
 - b. Setup of Opencv
 - c. Setup of Darknet
2. Installing Virtual environment
 1. Installing virtualenv (Requires admin privileges)

```
pip3 install virtualenv
```

2. Creating a virtualenv

```
virtualenv <virtualenv name>
```

3. Activating virtualenv

```
source <virtualenv name>/bin/activate
```

1. Clone the git repository in your desired location [link](#)

```
git clone https://github.com/swara07/CMSF
```

2. Download the weight file [link](#)

```
https://drive.google.com/drive/folders/1EmKPJpzaRg4zcVHP1k6GSSu  
k-nxN4pVq?usp=sharing
```

3. Open the cloned repository path in your terminal

4. Install pyqt5 in your system (Requires admin privileges)

```
sudo yum -y install python36-qt5-base python36-qt5-devel
```

5. Run the requirements.txt in the terminal

```
pip3 install -r requirements.txt
```

6. Copy dataset.names and dataset.data file from clone repository of CMSF into
xxx/darknet/data

7. Run main.py in the terminal

```
python3 main.py
```


3. Ubuntu

a. Prerequisites

```
sudo apt-get update  
sudo apt-get upgrade -y
```

```
sudo apt-get install -y build-essential cmake unzip pkg-config libxmu-dev libxi-dev  
libglu1-mesa libglu1-mesa-dev libjpeg-dev libpng-dev libtiff-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev libx264-dev libgtk-3-dev  
libopenblas-dev libatlas-base-dev liblapack-dev gfortran libhdf5-serial-dev graphviz  
python3-dev python3-tk python-imaging-tk linux-image-generic  
linux-image-extra-virtual linux-source linux-headers-generic
```

b. OpenCV from the Source

i. Install the [build tools](#) and dependencies:

```
sudo apt install build-essential cmake git pkg-config libgtk-3-dev \  
libavcodec-dev libavformat-dev libswscale-dev libv4l-dev \  
libxvidcore-dev libx264-dev libjpeg-dev libpng-dev libtiff-dev \  
gfortran openexr libatlas-base-dev python3-dev python3-numpy \  
libtbb2 libtbb-dev libdc1394-22-dev libopenexr-dev \  
libgstreamer-plugins-base1.0-dev libgstreamer1.0-dev
```

ii. Clone the OpenCV's and OpenCV contrib repositories:

```
mkdir ~/opencv_build && cd ~/opencv_build  
git clone https://github.com/opencv/opencv.git  
git clone https://github.com/opencv/opencv\_contrib.git
```

iii. Once the download is complete, create a temporary build directory, and [navigate](#) to it:

```
cd ~/opencv_build/opencv  
mkdir -p build && cd build
```

Set up the OpenCV build with CMake:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \ -D  
CMAKE_INSTALL_PREFIX=/usr/local \ -D INSTALL_C_EXAMPLES=ON \  
-D INSTALL_PYTHON_EXAMPLES=ON \ -D  
OPENCV_GENERATE_PKGCONFIG=ON \ -D  
OPENCV_EXTRA_MODULES_PATH=~/opencv_build/opencv_contrib/modules \
```

```
-D BUILD_EXAMPLES=ON ..
```

The **output** will look something like below:

```
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/vagrant/opencv_build/opencv/build
```

iv. Start the compilation process:

```
make -j8
```

Modify the -j flag according to your processor. If you do not know the number of cores in your processor, you can find it by typing nproc.

The compilation may take several minutes or more, depending on your system configuration.

v. Install OpenCV with:

```
sudo make install
```

vi. To verify the installation, type the following commands and you should see the OpenCV version.

i. C++ bindings:

```
pkg-config --modversion opencv4
```

If this command is not able to find the path then perform the below step given

ii. Set the Path Globally

```
export  
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:xxx/opencv_build/opencv/build/lib
```

Here xxx will start from root directory

c. Cuda and Cudnn(only for accelerating performances {GPU})

i. Use the following link to check compatibility between Tensorflow, cuDNN and CUDA

- a. <https://www.tensorflow.org/install/source#gpu>
- b. https://www.tensorflow.org/install/source#tested_build_configurations
- c. <https://stackoverflow.com/a/50622526>

ii. **Download and install the compatible CUDA version**

(<https://developer.nvidia.com/cuda-toolkit-archive>)

- a. Select operating system
- b. Select Architecture
- c. Select Linux Flavour or Windows Version
- d. Select installer type and follow instructions
- e. Reference image and steps to install cuda in ubuntu 20.04 [link](#)
- f. Additional step for ubuntu once you have installed the files of cuda

```
sudo nano ~/.bashrc
```

Add the following at the end of `~/.bashrc`

```
export PATH=/usr/local/cuda-11.2/bin${PATH:+:${PATH}}
export
LD_LIBRARY_PATH=/usr/local/cuda-11.2/lib64${LD_LIBRARY_PATH:+
:${LD_LIBRARY_PATH}}
export CUDA_HOME=/usr/local/cuda
```

b. Download cuDNN (<https://developer.nvidia.com/cuDNN>)

i. Procedure:

1. Go to: NVIDIA cuDNN home page.
2. Click Download.
3. Complete the short survey and click Submit.
4. Accept the Terms and Conditions. A list of available download versions of cuDNN displays.
5. Select the cuDNN version you want to install. A list of available resources is displayed.

ii. Installation on Linux:

1. Tar File:

- a. Navigate to the directory containing the cuDNN tar file.

- b. Unzip the cuDNN package

```
tar -zxvf cudnn-x.x-linux-x64-v8.x.x.x.tgz
```

- c. Link the downloaded lib64 to local environmental variables. First, open `~/.bashrc` by run the following.

```
cd ~  
sudo gedit ~/.bashrc
```

Add the following at the end of `~/.bashrc`. **Note: replace xxx with your own path.**

```
export  
LD_LIBRARY_PATH=xxx/cuda/lib64:$LD_LIBRARY_PATH
```

Save the file, and run by the following at the terminal.

```
source ~/.bashrc
```

- d. Copy the following files into the CUDA Toolkit directory

```
cd xxx/cuda/include  
sudo cp *.h /usr/local/cuda/include/  
cd ..  
sudo cp -r xxx/cuda/lib64/* /usr/local/cuda/lib64
```

2. Debian:

- a. Navigate to the directory containing the cuDNN Debian file.
b. Install the runtime library

```
sudo dpkg -i libcudnn8_x.x.x-1+cudax.x_amd64.deb
```

- c. Install the developer library

```
sudo dpkg -i libcudnn8-dev_8.x.x.x-1+cudax.x_amd64.deb
```

- d. Install the code samples and the cuDNN library documentation.

```
sudo dpkg -i libcudnn8-samples_8.x.x.x-1+cudax.x_amd64.deb
```

d. Darknet ([Repository](#))

Requirements:

- OpenCV >= 2.4

- a. Clone Darknet from <https://github.com/AlexeyAB/darknet>

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

- b. Navigate to the directory

- c. The following options can be set before using **make** :

- ix. GPU=0
- x. CUDNN=0
- xi. CUDNN_HALF=0
- xii. OPENCV=1**
- xiii. DEBUG=0
- xiv. OPENMP=0
- xv. LIBSO=1**
- xvi. ZED_CAMERA=0

```
cd darknet  
make
```

With GPU

Requirements:

- OpenCV >= 2.4
- CUDA>=11.2
- cuDNN >= 8.1
- GPU with CC >= 3.0

- a. Clone Darknet from <https://github.com/AlexeyAB/darknet>

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

- b. Navigate to the directory

- c. The following options can be set before using **make** :

- ix. **GPU=1**
- x. **CUDNN=1**
- xi. **CUDNN_HALF=1**
- xii. **OPENCV=1**
- xiii. **DEBUG=1**
- xiv. **OPENMP=1**

- xv. LIBSO=1
- xvi. ZED_CAMERA=1

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

Example Makefile flags

Now, for example, if OpenCV is to be used, set OPENCV = 1

```
cd darknet
make
```

e. CSMF Project Installation

1. Pre-requirements
 - a. Python > 3.7
 - b. Setup of Opencv
 - c. Setup up of Darknet
2. Installing Virtual environment
 - a. Installing virtualenv (Requires admin privileges)

```
pip3 install virtualenv
```

- b. Creating a virtualenv

```
virtualenv <virtualenv name>
```

- c. Activating virtualenv

```
source <virtualenv name>/bin/activate
```

3. Setup
 - a. Clone the git repository in your desired location [link](#)

```
git clone https://github.com/swara07/CSMF
```

- b. Download the weight file [link](#)

```
https://drive.google.com/drive/folders/1EmKPJpzaRg4zcVH
```

Plk6GSSuk-nxN4pVq?usp=sharing

- c. Open the cloned repository path in your terminal
- d. Install pyqt5 in your system

```
sudo apt-get install python3-pyqt5
sudo apt install pyqt5-dev-tools pyqt5-dev
sudo apt-get install python3-pyqt5.qtsql
```

- e. Run the requirements.txt in the terminal

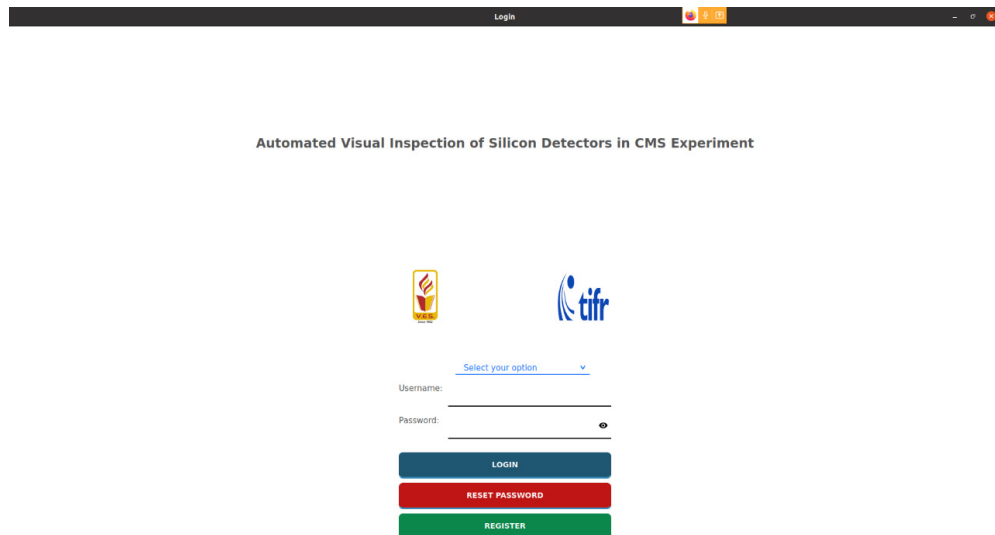
```
sudo apt install python3-pip
pip3 install -r requirements.txt
```

- f. Copy dataset.names file from clone repository of CMSF into xxx/darknet/data
- g. Run main.py in the terminal

```
python3 main.py
```

4. Application Overview

- a. You will see the below screen

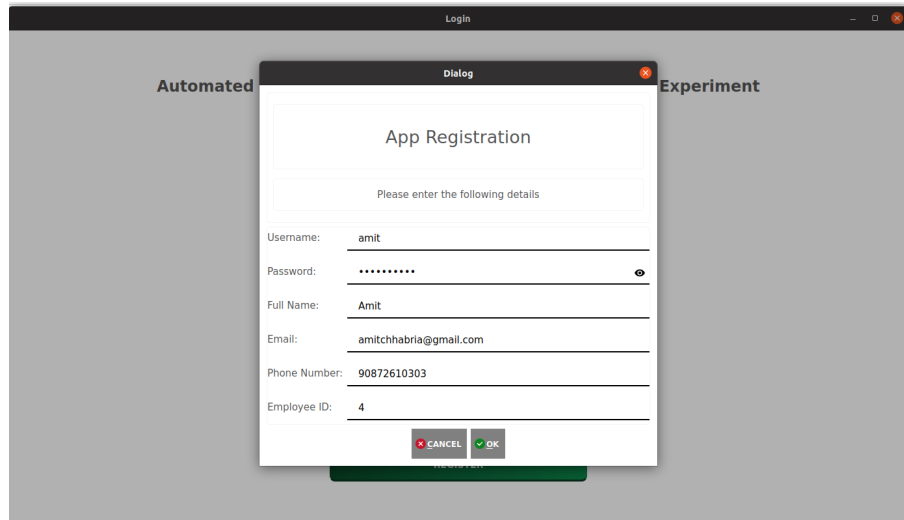


- b. Click on register

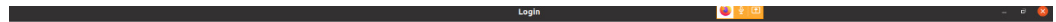
NOTE:

Password should contain the following:

1. Length should be greater than 8 characters
2. It should contain at least one lowercase letter ,one uppercase and one digit
3. Special characters can only be '@' , '\$' , '_'



- c. Try to login in with your credentials → Select from the drop down Training and Annotation

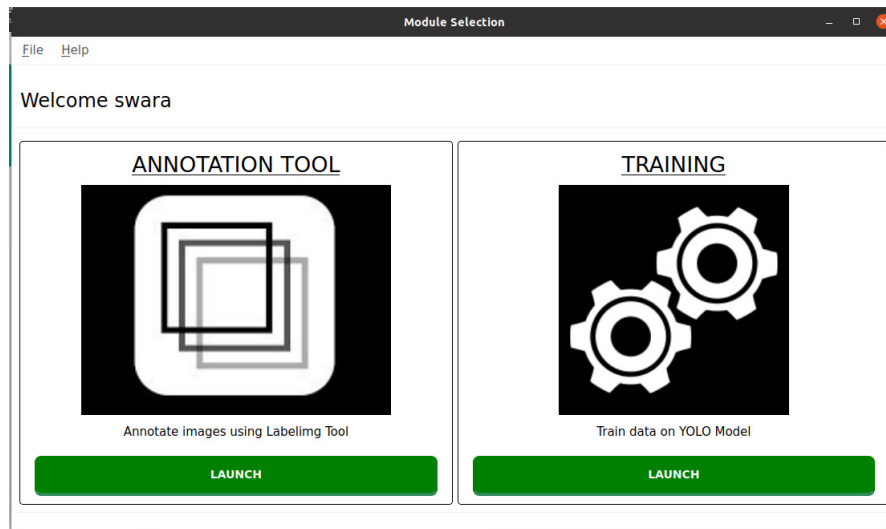


Automated Visual Inspection of Silicon Detectors in CMS Experiment

The screenshot shows the login page of the application. At the top, there are two logos: a yellow one on the left and a blue 'tifr' logo on the right. Below the logos is a dropdown menu labeled 'Training and Annotation'. The login form consists of the following elements:

- Username field: swara
- Password field: (masked with dots)
- LOGIN button (blue)
- RESET PASSWORD button (red)
- REGISTER button (green)

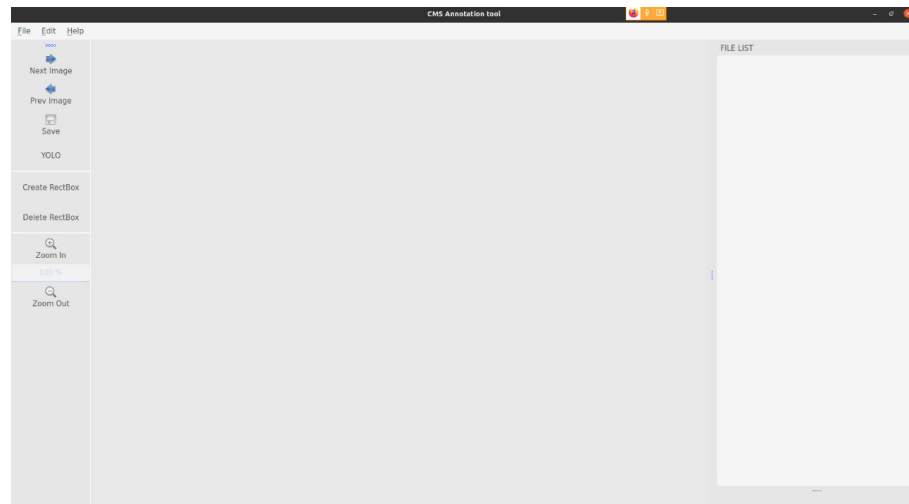
d. Select Training Or Annotation



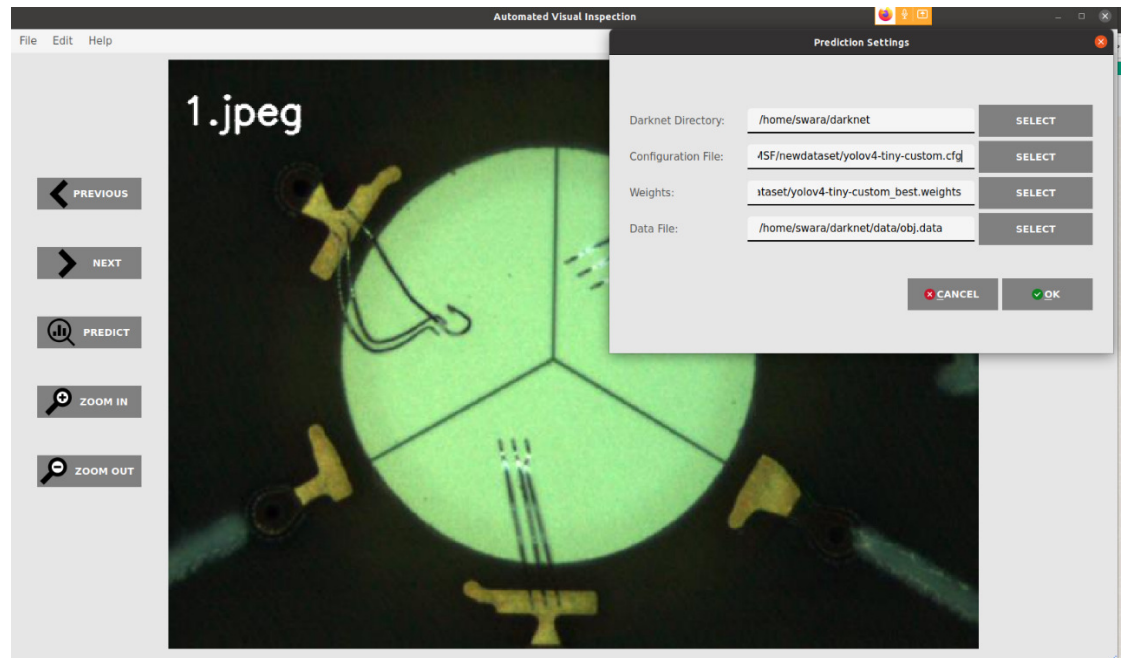
e. Training Screen

Training									
<div>← New Training Start Training Delete Training</div>									
	NAME	START DATE	START TIME	END DATE	END TIME	HOURS ELAPSED	PROCESS ID	STATUS	REATED B
1	<input type="checkbox"/> train1	--	--	--	--	0	--	Not Starte...	

f. Annotation Screen



g. Logout → Login again for Testing → Testing Screen → Select Edit and Configure darknet Folder and path



5. Training Documentation

Steps to start training

a. Labeled Custom Dataset - This should be done using annotation tool

b. Custom cfg file

Download the ****yolov4-tiny-custom.cfg**** file from *****darknet/cfg***** directory, make changes to it, and upload it to the *****yolov4-tiny***** folder on your drive .

You can also download the custom config files from the official [AlexeyAB Github](<https://www.github.com/AlexeyAB/darknet>)

You need to make the following changes in your custom config file:

1. change line batch to batch=64
2. change line subdivisions to subdivisions=16
3. change line max_batches to (classes*2000 but not less than number of training images, but not less than number of training images and not less than 6000), f.e. max_batches=6000 if you train for 3 classes
4. change line steps to 80% and 90% of max_batches, f.e. steps=4800,5400
5. set network size width=416 height=416 or any value multiple of 32
6. change line classes=80 to your number of objects in each of 2 [yolo]-layers
7. change [filters=255] to filters=(classes + 5)x3 in the 2 [convolutional] before each [yolo] layer, keep in mind that it only has to be the last [convolutional] before each of the [yolo] layers.

So if classes=1 then it should be filters=18. If classes=2 then write filters=21.

c. dataset.data and dataset.names files

dataset.data

```
classes = 2
train   = data/train.txt
valid   = data/test.txt
names   = data/dataset.names
backup  = /mydrive/yolov4-tiny/training
```

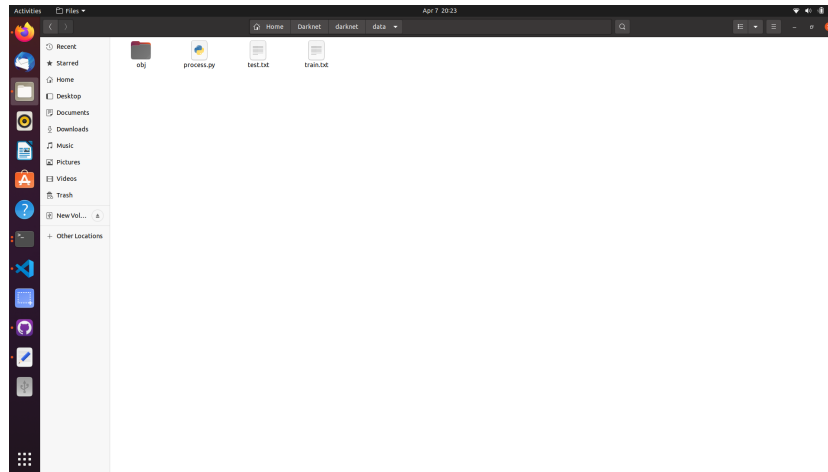
dataset.names

```
Open [v] [+]
1 three
2 not_three
```

d. process.py file (to create train.txt and test.txt files for training)

This **process.py** script creates the files **train.txt** & **test.txt** where the **train.txt** file has paths to 90% of the images and **test.txt** has paths to 10% of the images.

e. Keep all the above files in darknet → data folder



f. Download the weight file from this [link](#)

6. Video links:

https://drive.google.com/file/d/1qy_2ccuTs6TQ6XKJJHyzo1JNm4lEXT_/view?usp=sharing