GSR COOKBOOK 21-22

Introduction

YOLO-App is a YOLO training and inference system that can be used to label images containing typewritten/handwritten english, train YOLO models and get inference from trained models. **Download here.**

What is YOLO?

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.

Requirements

- Python 3.6 and above
- CUDA and cuDNN (For Training on GPU)
- Darknet (For Training and Testing)

Python Libraries Required

The libraries can be installed in a python virtual environment using the provided ***requirements.txt*** or manually.

The required libraries are:

- PyQt5
- lxml
- bcrypt
- psutil
- python-dateutil
- qt-material
- qtwidgets
- opencv-python
- numpy
- autocorrect
- wheel (required for building autocorrect and qtwidgets)
- Gingerit (required for grammar and spelling correction)
- Layoutparser (required for block detection)

Steps to get started

1. Python

Python can be downloaded and installed from here for Windows based systems. It is already installed on Linux distros.

2. CUDA and cuDNN

This step can be skipped if the GPU is not going to be used for training and testing.

Use the following link to check compatibility between Tensorflow, cuDNN and CUDA
(This is required only for TensorFlow)

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

a. NVIDIA-SMI installation steps and link

➔ **Installation Steps for NVIDIA-SMI**

- In case any previous NVIDIA installations are there to get rid of it add:

```
sudo apt-get purge nvidia*
//To ensure no other nvidia is meddling
OR ALTERNATIVELY
sudo apt remove nvidia*
OR ALTERNATIVELY
sudo apt autoremove
```

- To add the NVIDIA PPA, run the following command. This is the official package repository for installing NVIDIA drivers:

```
sudo add-apt-repository ppa:graphics-drivers/ppa
```

- Update System:

```
sudo apt-get update

sudo apt-get upgrade
```

- Install NVIDIA driver:

```
ubuntu-drivers devices

//the above code is to find out which driver is suitable for
the PC’s graphics card* (add example)

sudo apt-get install nvidia-driver-460

sudo ubuntu-drivers autoinstall

sudo reboot
```

*Find out apt driver from first command

- Check GPU: Check the NVIDIA driver’s version and CUDA version

```
nvidia-smi
```

b. Download and install the compatible CUDA version (11.3.0)

(CUDA Toolkit 11.3 Downloads | NVIDIA Developer)

(OR, directly follow the commands given below for CUDA Toolkit 11.3 for Ubuntu 20.04)

- Select operating system
- Select Architecture
- Select Linux Distribution or Windows Version
- Select installer type, follow instructions and execute the provided commands

```
wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2
004/x86_64/cuda-ubuntu2004.pin
```

```
sudo mv cuda-ubuntu2004.pin
/etc/apt/preferences.d/cuda-repository-pin-600
```

```
wget
https://developer.download.nvidia.com/compute/cuda/11.3.0/local_installers/cuda-repo-ubuntu2004-11-3-local_11.3.0-465.19.01-1_amd64.deb
```

```
sudo dpkg -i
cuda-repo-ubuntu2004-11-3-local_11.3.0-465.19.01-1_amd64.deb
```

```
sudo apt-key add
/var/cuda-repo-ubuntu2004-11-3-local/7fa2af80.pub
```

```
sudo apt-get update
```

```
sudo apt-get -y install cuda
```

c. Download cuDNN

Download Link
Installation Video for reference

◆ Procedure

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

◆ Installation on Linux

- **Tar File**
    - Navigate to the directory containing the cuDNN tar file.
    - Open a terminal and type the following to get superuser privileges

      ```
      $ sudo su
      ```

    - Unzip the cuDNN package

```
$ tar -xzvf cudnn-11.4-linux-x64-v8.2.4.15.tgz
```

- Copy the following files into the CUDA Toolkit directory

```
$ sudo cp cuda/include/cudnn*.h
/usr/local/cuda/include

$ sudo cp -P cuda/lib64/libcudnn*
/usr/local/cuda/lib64

$ sudo chmod a+r /usr/local/cuda/include/cudnn*.h
/usr/local/cuda/lib64/libcudnn*
```

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

```
sudo dpkg -i
libcudnn8_8.2.4.15-1+cuda11.4_amd64.deb
```

  - Install the developer library

```
sudo dpkg -i
libcudnn8-dev_8.2.4.15-1+cuda11.4_amd64.deb
```

  - Install the code samples and the cuDNN library documentation.

```
sudo dpkg -i
libcudnn8-samples_8.2.4.15-1+cuda11.4_amd64.deb
```

- **RPM**
  - Download the rpm package libcudnn*.rpm to the local path.
  - Install the rpm package from the local path. This will install the cuDNN libraries.

```
rpm -ivh libcudnn8-*.x86_64.rpm

rpm -ivh libcudnn8-devel-*.x86_64.rpm
rpm -ivh libcudnn8-samples-*.x86_64.rpm
```

- **Package Manager**

  1. Enable the repository. The following commands enable the repository containing information about the appropriate cuDNN libraries online

Option	**Description**

Ubuntu 16.04	`wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-ubuntu1604.pin`
	
	`sudo mv cuda-ubuntu1604.pin /etc/apt/preferences.d/cuda-repository-pin-600`
	
	`sudo apt-key adv --fetch-keys http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub`
	
	`sudo add-apt-repository "deb https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/ /"`
	
	`sudo apt-get update`

Ubuntu 18.04 and 20.04
 	
Where ${OS} is ubuntu1804 or ubuntu2004.
	`wget https://developer.download.nvidia.com/compute/cuda/repos/${OS}/x86_64/cuda-${OS}.pin`
	
	`sudo mv cuda-${OS}.pin /etc/apt/preferences.d/cuda-repository-pin-600`
	
	`sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/re`

	pos/${OS}/x86_64/7fa2af80.pub sudo add-apt-repository "deb https://developer.download.nvidia.com/compute/cuda/repos/${OS}/x86_64/ /" sudo apt-get update

2. Install the cuDNN library

(Differs from PC to PC. Also, refer to official documentation here.)

```
sudo apt-get install libcudnn8=${cudnn_version}-1+${cuda_version}

sudo apt-get install libcudnn8-dev=${cudnn_version}-1+${cuda_version}

Where:
  ● ${cudnn_version} is 8.1.1.*
  ● ${cuda_version} is cuda10.2 or cuda11.2
```

◆ **Verify Installation**

- Copy the cuDNN samples to a writable path

```
$cp -r /usr/src/cudnn_samples_v8/ $HOME
```

- Go to the writable path

```
$ cd  $HOME/cudnn_samples_v8/mnistCUDNN
```

- Compile the mnistCUDNN sample

```
$make clean && make
```

- Run the mnistCUDNN sample

  ```
  $ ./mnistCUDNN
  ```

- If cuDNN is properly installed and running on your Linux system, you will see a message similar to the following:

  ```
  Test passed!
  ```

◆ Installation on Windows

Before issuing the following commands, you'll need to replace x.x and 8.x.x.x with your specific CUDA and cuDNN versions and package date.

Where:

- The CUDA directory path is referred to as C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x
- The cuDNN directory path is referred to as <installpath>

1. Navigate to your <installpath> directory containing cuDNN.
2. Unzip the cuDNN package.

   ```
   cudnn-x.x-windows-x64-v8.x.x.x.zip
   ```

3. Copy the following files into the CUDA Toolkit directory
   a. Copy <installpath>\cuda\bin\cudnn*.dll to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x\bin.
   b. Copy <installpath>\cuda\include\cudnn*.h to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x\include.

   c. Copy <installpath>\cuda\lib\x64\cudnn*.lib to C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x\lib\x64.

4. Set the following environment variables to point to where cuDNN is located. To access the value of the $(CUDA_PATH) environment variable, perform the following steps:

   a. Open a command prompt from the Start menu.

   b. Type Run and hit Enter.

   c. Issue the control sysdm.cpl command.

   d. Select the Advanced tab at the top of the window.

   e. Click Environment Variables at the bottom of the window.

   f. Ensure the following values are set:

```
Variable Name: CUDA_PATH

Variable Value: C:\Program Files\NVIDIA GPU
Computing Toolkit\CUDA\vx.x
```

5. Include cudnn.lib in your Visual Studio project

   a. Open the Visual Studio project and right-click on the project name.

   b. Click Linker > Input > Additional Dependencies.

   c. Add cudnn.lib and click OK.

3. Darknet (Repository)

Note that the code provides a CPU compiled version of darknet. This can be used directly in the application. However if one has a GPU with NVIDIA drivers, they can use the following steps to use GPU for inference.

A. Requirements

a. CMake >= 3.18: https://cmake.org/download/

b. Powershell (already installed on windows): https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell

c. CUDA >= 10.2: https://developer.nvidia.com/cuda-toolkit-archive (on Linux do Post-installation Actions)

d. OpenCV >= 2.4: use your preferred package manager (brew, apt), build from source using vcpkg or download from OpenCV (on Windows set system variable OpenCV_DIR = C:\opencv\build - where are the include and x64 folders image)

e. cuDNN >= 8.0.2 https://developer.nvidia.com/rdp/cudnn-archive

    i. On Linux copy cudnn.h,libcudnn.so... as described here https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html#installlinux-tar

    ii. On Windows copy cudnn.h,cudnn64_7.dll, cudnn64_7.lib as described here https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html#installwindows

f. GPU with CC >= 3.0: https://en.wikipedia.org/wiki/CUDA#GPUs_supported

B. Compiling on Linux

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

    i. **GPU=1** to build with CUDA to accelerate by using GPU (CUDA should be in /usr/local/cuda)

    ii. **CUDNN=1** to build with cuDNN v5-v7 to accelerate training by using GPU (cuDNN should be in /usr/local/cudnn)

    iii. **CUDNN_HALF=1** to build for Tensor Cores (on Titan V / Tesla V100 / DGX-2 and later) speedup Detection 3x, Training 2x

iv. **OPENCV=1** to build with OpenCV 4.x/3.x/2.4.x - allows to detect on video files and video streams from network cameras or web-cams

v. **DEBUG=1** to build debug version of Yolo

vi. **OPENMP=1** to build with OpenMP support to accelerate Yolo by using multi-core CPU

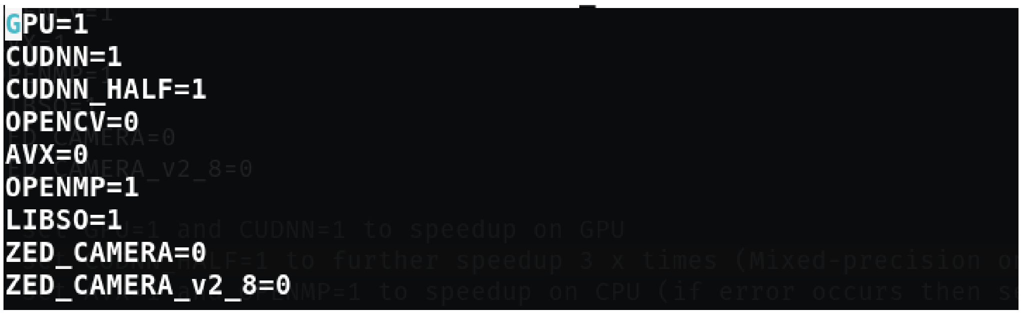
vii. **LIBSO=1** to build a library darknet.so and binary runnable file uselib that uses this library.

viii. **ZED_CAMERA=1** to build a library with ZED-3D-camera support (should be ZED SDK installed)

ix. You also need to specify for which graphics card the code is generated. This is done by setting ARCH=.

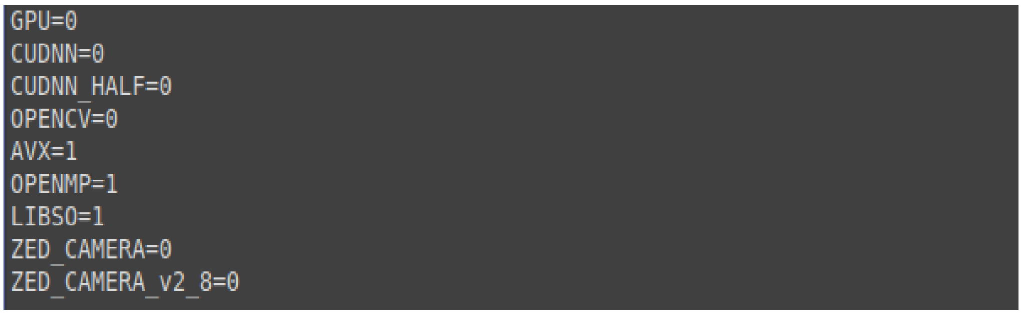
If you use a newer version than CUDA 11 you further need to edit line 20 from Makefile and remove -gencode arch=compute_30,code=sm_30 \ as Kepler GPU support was dropped in CUDA 11.

You can also drop the general ARCH= and just uncomment ARCH= for your graphics card.Now, for example, if OpenCV is to be used, set OPENCV = 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 (Running using GPU)



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

Example Makefile flags (Running using CPU)

b. Clone Darknet from https://github.com/AlexeyAB/darknet

c. Navigate to the directory

d. Adjust flags in the ‘MakeFile’ as required.

e. Open a terminal and run **make** command inside the directory

C. Compiling on Windows using CMake:

a. Requires:

    i. MSVC: https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community

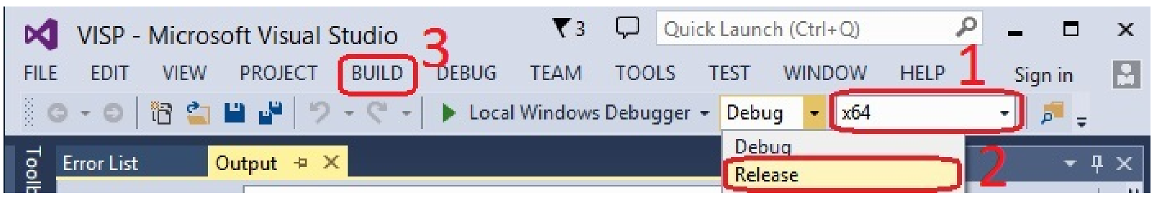
    ii. CMake GUI: Windows win64-x64 Installer https://cmake.org/download/

    iii. Download Darknet zip-archive with the latest commit and uncompress it: master.zip

b. Start (button) -> All programs -> CMake -> CMake (gui) ->

c. look at image In CMake: Enter input path to the darknet Source, and output path to the Binaries -> Configure (button) -> Optional platform for generator: x64 -> Finish -> Generate -> Open Project ->

d. in MS Visual Studio: Select: x64 and Release -> Build -> Build solution



e. Find the executable file darknet.exe in the output path to the binaries you specified

D. Pre/Post Compilation:

This step is necessary and can be performed before or after compiling darknet.

The file **darknet_images.py** present in the cloned repository has to be replaced with a modified version of the same file that can be found **here** for the inferences to be displayed correctly. Not replacing the file might lead to application crashes.

4. App Installation

We recommend creating a virtual environment to install the application.

1. Open a terminal
2. Install Python Virtual Environments if not installed.

   ```
   $ sudo apt install -y python3-venv
   ```

3. Create a virtual environment

   ```
   $ python3 -m venv gsr_env

   where gsr_env is the environment name
   ```

4. Activate the virtual environment

   ```
   $ source gsr_env/bin/activate
   ```

   Your command prompt will now be prefixed with the name of your environment, in this case it is called **gsr_env**.

   ```
   (gsr_env) user@ubuntu:~/TIFR$
   ```

5. Navigate to the directory where the application has been cloned.
6. Run the following command to install all requirements:

   ```
   $ pip install -r requirements_full.txt
   ```

   Note: For some machines detectron2 installation may fail. For this use the command mentioned in the official documentation for a cpu install of detectron2, as shown in the snippet below.

   ```
   $  python -m pip install detectron2 -f \
   ```

```
https://dl.fbaipublicfiles.com/detectron2/wheels/cpu/torch1.10/inde
x.html
```

7. The application also requires a few system wide installations essential for running the Gujarati spelling correction and Clean Image options. The Gujarati spell correction requires a system wide install of the Hunspell Gujarati dictionary, which can be done as follows.

```
$  sudo apt install hunspell-gu
```

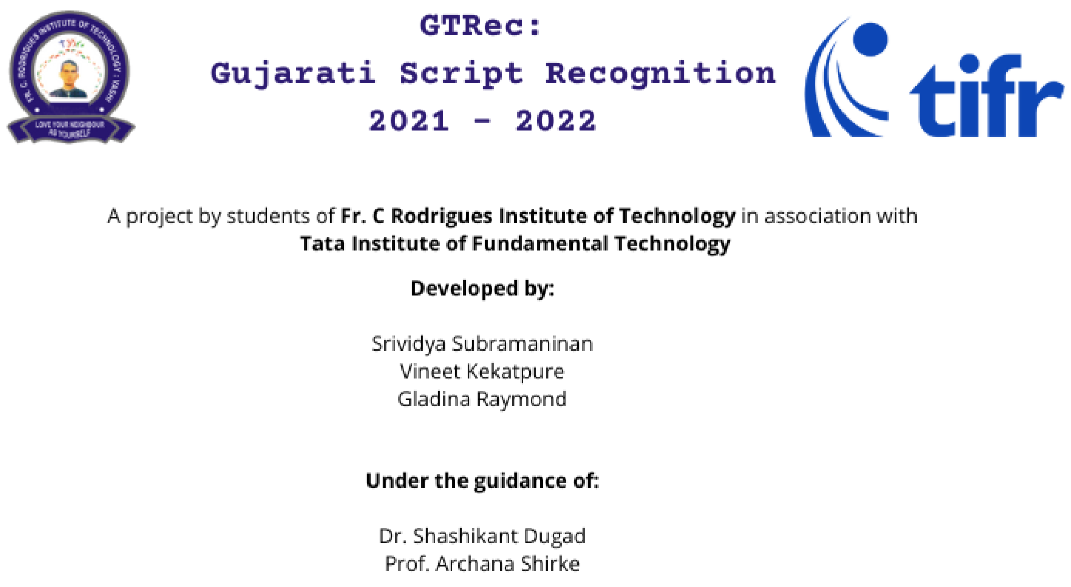
The clean image option requires a system wide install of ImageMagick. This can be done as shown.

```
$  sudo apt install imagemagick
```

8. Launch the application by executing the following command in the application's root directory:

```
$  python3 main.py
```

On successful installation, the certificate screen will be visible.



GTRec:
Gujarati Script Recognition
2021 - 2022

A project by students of **Fr. C Rodrigues Institute of Technology** in association with **Tata Institute of Fundamental Technology**

Developed by:

Srividya Subramaninan
Vineet Kekatpure
Gladina Raymond

Under the guidance of:

Dr. Shashikant Dugad
Prof. Archana Shirke

Pre-Training Requirements (More Information):

1. Connecting to TIFR GPU Server

The TIFR GPU Server is a Pop!_OS Operating System. To connect to it one must ssh int the system, much like one would connect to an AWS Linux Instance. The ssh command provides a secure encrypted connection between two hosts over an insecure network. This connection can be used for terminal access, file transfers, and for tunneling other applications.

Requirements -

1. A terminal session (Linux) / cmd - powershell (Windows)

2. ssh command - OpenSSH comes pre-installed in most operating systems.

However if one sees a ‘Command not found’ error they may install OpenSSH by referring to these guides.

a. Linux Operating Systems - Here.

b. Windows Operating Systems - Here.

Steps -

1. Open a terminal and type in the following command -

```
ssh -p 4748 -tt sipm@gwssh.tifr.res.in
<username>@<ip-address> ssh -p 22 -tt
```

Command Explanation -

1. -p : Port to connect to on the remote host. This can be specified on a per-host basis in the configuration file
2. -tt: Force pseudo-terminal allocation. This can be used to execute arbitrary screen-based programs on a remote machine, which can be very useful, e.g. when implementing menu services. Multiple -t options force tty allocation, even if ssh has no local tty.
3. This command will prompt the user for 2 passwords - One for SIPM and the other for the <user>. Students may get all these credentials from their Internal / External Guides.

4. This command will also lead to another prompt -

   ```
   The authenticity of host '<host>' can't be established.ECDSA key fingerprint is

   SHA256:TER0dEslggzS/BROmiE/s70WqcYy6bk52fs+MLTIptM.

   Are you sure you want to continue connecting (y/n)? y

   Warning: Permanently added 'pc' (ECDSA) to the list of known hosts.
   ```

   As shown in the snippet, one must type in 'yes' in order to establish this server as a trusted one. This prompt will appear only the first time this command is run.

Note -

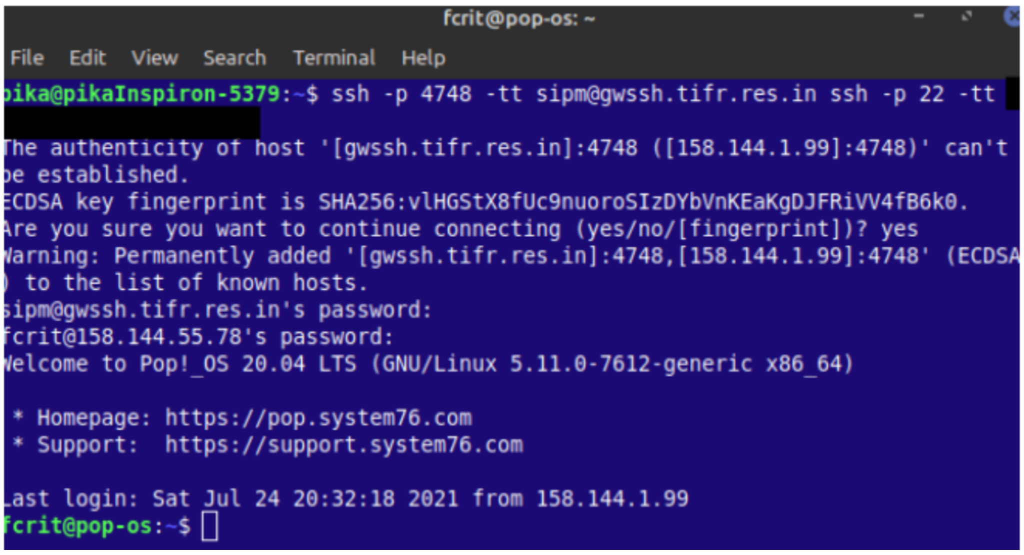
1. Students will not have permission to install any software on the system. In order to do so they must contact their guides.

2. Exercise utmost precaution while modifying files on the server.

3. Only access the directories which have been created for project related work.

4. At a time only one user may access the system.

5. For transferring data to the server students may use gdown.pl.

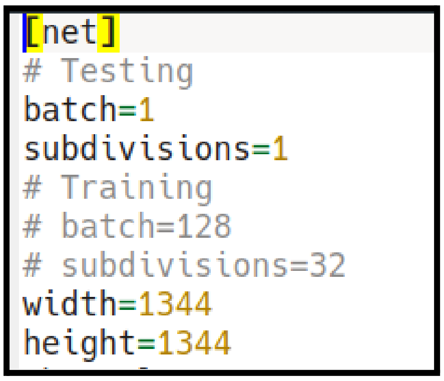
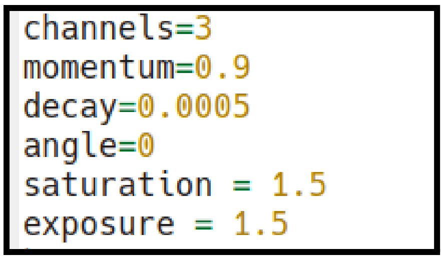


```
pika@pikaInspiron-5379:~$ ssh -p 4748 -tt sipm@gwssh.tifr.res.in ssh -p 22 -tt [redacted]
The authenticity of host '[gwssh.tifr.res.in]:4748 ([158.144.1.99]:4748)' can't
be established.
ECDSA key fingerprint is SHA256:vlHGStX8fUc9nuoroSIzDYbVnKEaKgDJFRiVV4fB6k0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[gwssh.tifr.res.in]:4748,[158.144.1.99]:4748' (ECDSA
) to the list of known hosts.
sipm@gwssh.tifr.res.in's password:
fcrit@158.144.55.78's password:
Welcome to Pop!_OS 20.04 LTS (GNU/Linux 5.11.0-7612-generic x86_64)

 * Homepage: https://pop.system76.com
 * Support:  https://support.system76.com

Last login: Sat Jul 24 20:32:18 2021 from 158.144.1.99
fcrit@pop-os:~$
```

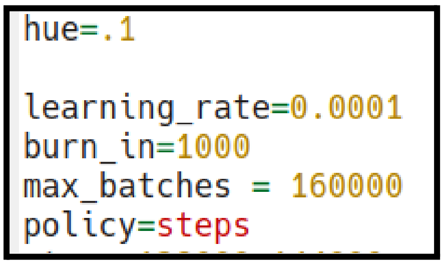
2. Configuration File

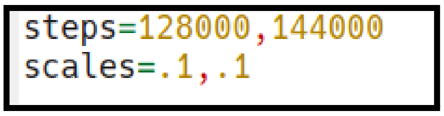
Configuration files for different YOLO versions can be found inside the **cfg** directory inside the cloned Darknet repository.

Understanding the YOLO config file:

Config File	Meaning
 `[net]` `# Testing` `batch=1` `subdivisions=1` `# Training` `# batch=128` `# subdivisions=32` `width=1344` `height=1344`	● **Batch** = number of samples (images) which will be processed in one batch ● **Subdivisions** = number of mini_batches in one batch, size mini_batch = batch/subdivisions, so GPU processes mini_batch samples at once, and the weights will be updated for batch samples (1 iteration processes batch images) ● **Width** = network size (width), so every image will be resized to the network size during Training and Detection ● **Height** = network size (height), so every image will be resized to the network size during Training and Detection
 `channels=3` `momentum=0.9` `decay=0.0005` `angle=0` `saturation = 1.5` `exposure = 1.5`	● **Channels** = network size (channels), so every image will be converted to this number of channels during Training and Detection ● **Momentum** = accumulation of movement, how much the history affects the further change of weights (optimizer) ● **Decay** = a weaker updating of the weights for typical features, it

	eliminates dysbalance in dataset (optimizer) ● **Angle** = randomly rotates images during training (classification only) ● **Saturation** = randomly changes saturation of images during training ● **Exposure** = randomly changes exposure (brightness) during training


`hue=.1`
 	
`learning_rate=0.0001`
`burn_in=1000`
`max_batches = 160000`
`policy=steps`
	● **Hue** = randomly changes hue (color) during training
	● **Learning Rate** = initial learning rate for training
	● **Burn_in** = initial burn_in will be processed for the first 1000 iterations, current_learning rate = learning_rate * pow(iterations / burn_in, power) = 0.001 * pow(iterations/1000, 4) where is power=4 by default
	● **max_batches** = the training will be processed for this number of iterations (batches)
	● **Policy** = policy for changing learning rate: constant (by default), sgdr, steps, step, sig, exp, poly, random (f.e., if policy=random - then current learning rate will be changed in this way = learning_rate * pow(rand_uniform(0,1), power))


`steps=128000,144000`
`scales=.1,.1`
	● **Steps** = at these numbers of iterations the learning rate will be multiplied by scales factor
	● **Scales** = learning_rate * scales[0] * scales[1] = 0.001 * 0.1 * 0.1 = 0.00001

Parameters/Values to be adjusted:

- batch to batch=64
- subdivisions to subdivisions=16
- max_batches to (classes*2000, but not less than the number of training images and not less than 6000), for example. max_batches=6000 if you train for 3 classes
- steps to 80% and 90% of max_batches, f.e. steps=4800,5400
- network size width=416 height=416 or any value multiple of 32
- change line classes=80 to your number of objects in each of 3 [yolo]-layers
- change [filters=255] to filters=(classes + 5)x3 in the 3 [convolutional] before each [yolo] layer, keep in mind that it only has to be the last [convolutional] before each of the [yolo] layers.
- when using [Gaussian_yolo] layers, change [filters=57] filters=(classes + 9)x3 in the 3 [convolutional] before each [Gaussian_yolo] layer

So if classes=1 then there should be filters=18. If classes=2 then write filters=21. (**Do not write in the cfg-file: filters=(classes + 5)x3**)

So for example, for 2 objects, your file yolo-obj.cfg should differ from yolov4-custom.cfg in such lines in each of 3 [yolo]-layers:

```
[convolutional]

filters=21

[region]

classes=2
```

3. Initial Weights

The initial weights required to start training can be downloaded using the following commands:

YOLO Version	Command
YOLO V3	!wget http://pjreddie.com/media/files/darknet53.conv.74
YOLO V4	!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137

The links can also be pasted directly in the browser to download without wget. It should be noted that training can also be carried out without any initial weights. This can lead to better accuracy in cases where the recognition task is significantly different from the information learned in the weights.

4. Name File

After annotating images using Labelimg tool present inside the YOLO App, a 'classes.txt' file will be generated containing all the classes used. This file can be used as the **.names** file required for training.

5. Data File

Create file **gsr.data** in the desired location in the following format:

```
classes = 2

train  = data/train.txt

valid  = data/test.txt

names = data/obj.names

backup = backup/
```

6. train.txt

Create file **train.txt** with filenames of your images, each filename in a new line, with paths. For example:

```
data/obj/img1.jpg

data/obj/img2.jpg

data/obj/img3.jpg
```

7. Running Training on Server GPU

1. Make a complete dataset with annotations
2. Zip it and upload it to a public google drive link.
3. SSH into the pop os server (refer Server connection guide in Section 1)
4. Run the following commands for downloading and extracting uploaded datasets for training and validation. (A possible split could be 80-20 or 75-25)

```
cd fcrit/gsr/gdown.pl

./gdown.pl

mv dataset.tar.gz ~/fcrit/gsr/gujarati_training

tar -xf dataset.tar.gz
```

5. The next task is to run generate_train.txt to get train.txt
    1. Give path to the training dataset which has images and annotations
    2. Give path to gujarati_training/config (where train.txt must be stored)
6. Run generate_valid.txt to get validation.txt
    1. Give path to the validation dataset which has images and annotations
    2. Give path to gujarati_training/config (where validation.txt must be stored)
7. Update spaces.names file – a list of classes for the model to identify
8. Update space.data file -
    1. Number of classes
    2. Path to train.txt file
    3. Path to validation.txt file

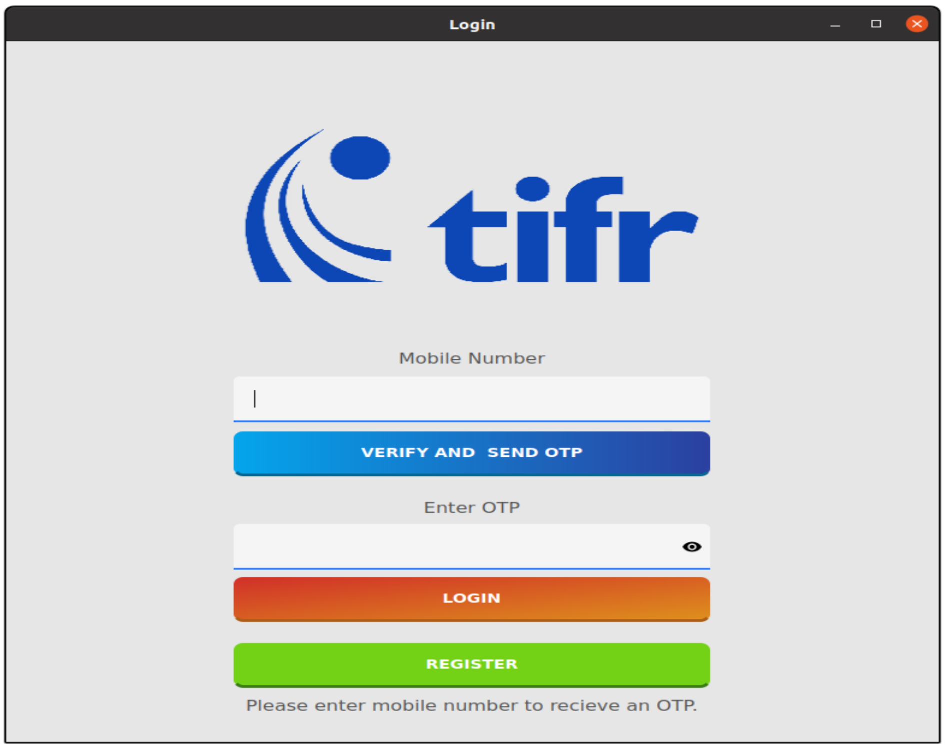
    4. Path to space.names file
    5. Path to store the weights folder (backup)
9. Learn about the cfg file from the documentation -
    1. yolo_v4_427.cfg – for training leave it as it is
    2. yolo_v4_427_test.cfg – for testing uncomment the lines for testing
        1. max_batches = no. of classes * 2000
        2. steps = 80% mb, 90%mb
        3. find classes (3 occurances) = 362
        4. in conv layer above yolo layer filters = (classes + 5) * 3
10. Download initial weights for YOLO with curl – (but put them where?)
11. cd fcrit/gsr/train_darknet/darknet2
12. Edit the train_script_4V.sh
    1. Give path to the initial weights - location where it is downloaded using curl
    2. Give yolov4_427.cfg path in cfg_path
    3. Give space.data path in data_path
    4. Give run_info path for log of details
13. nohup bash train_script_4V.sh &
14. watch -n 1 nvidia-smi
15. Monitoring – once in two hours initially and then once in a day
    1. see value of avg_loss should be as low as possible
    2. see value of validation accuracy – mean average precision

How to Use

A. Creating an account

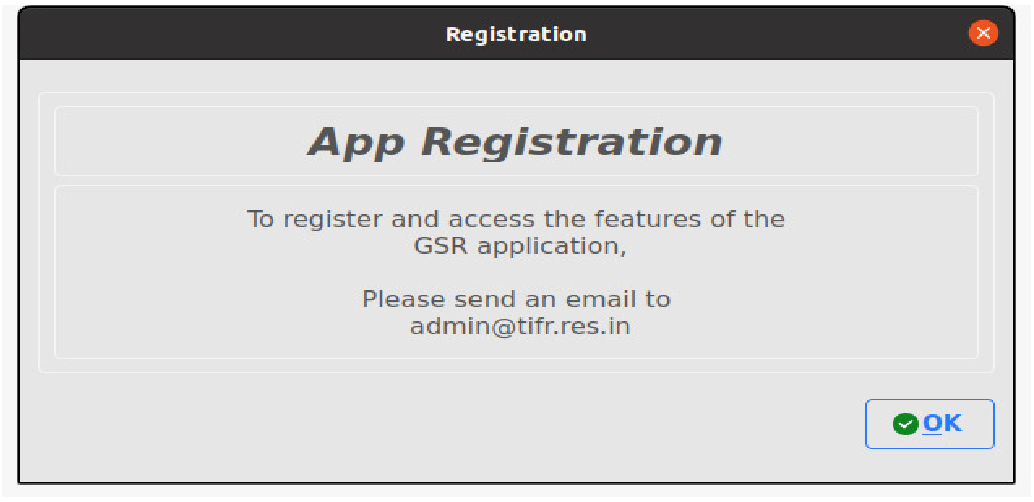
1. Launch the application.

2. Click on the **Register** button.



Please enter mobile number to recieve an OTP.

3. This will open the **User Registration** window.



App Registration

To register and access the features of the GSR application,

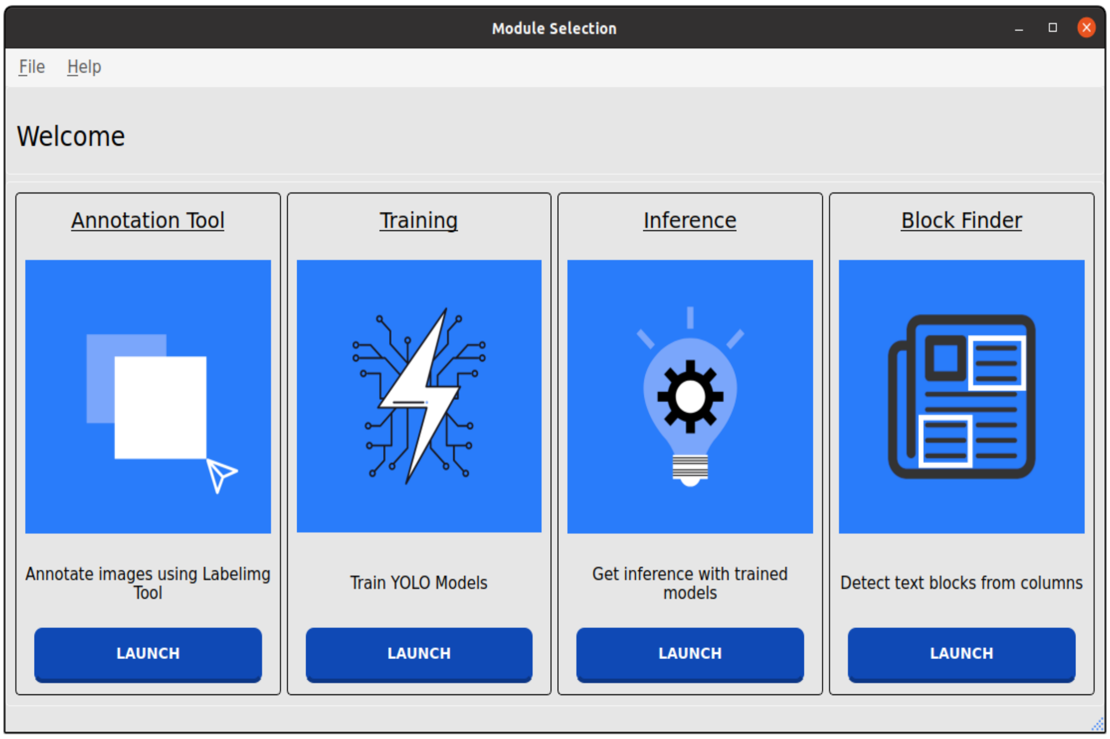
Please send an email to admin@tifr.res.in

4. Enter the required details. (**Username**, **Email** and **Phone Number** must be unique)

5. Click **OK.**

6. Users can login via phone number on verifying it via OTP.

B. Annotation Tool (TIFR Gitlab Repository)



The annotation tool embedded within the application is a fork of the original Labelimg. The dialog box for class selection has been changed in our fork. A standalone version of the tool can be found here.

The annotations are saved in a text file with the same name as that of the image in the directory specified through the toolbar.

The format of the annotations is:

```
<object-class> <x_center> <y_center> <width> <height>
```

where:

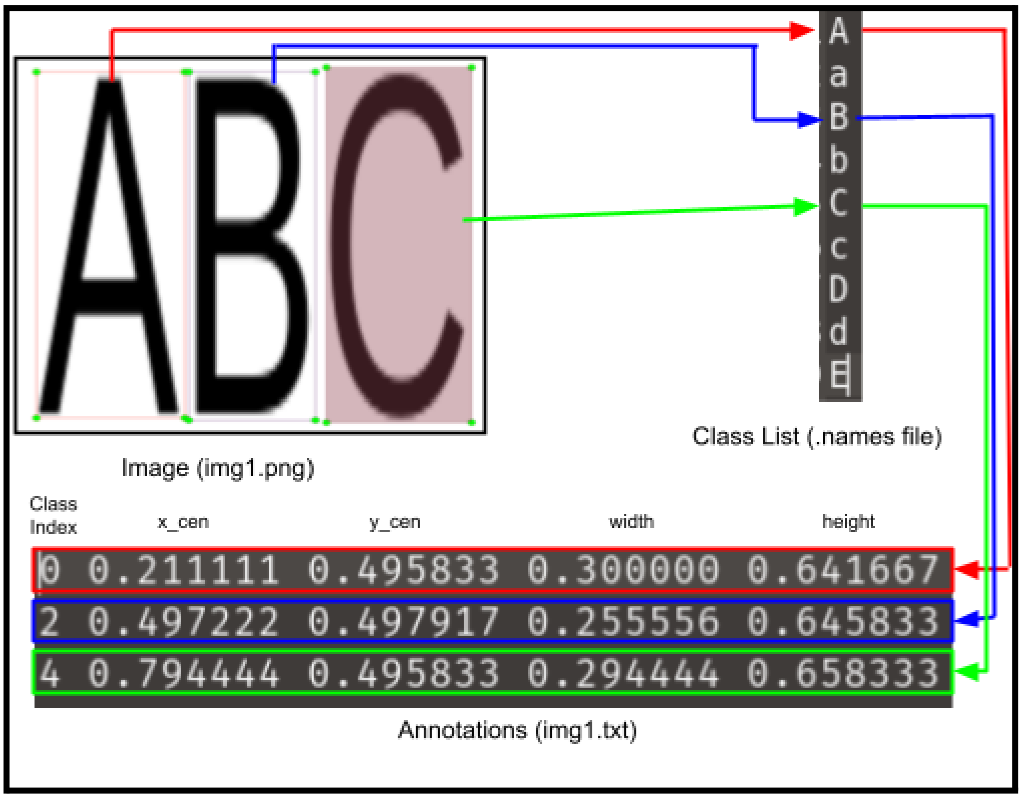
- <object-class> - integer object number from 0 to (classes-1)
- <x_center> <y_center> <width> <height> - float values relative to width and height of image, it can be equal from (0.0 to 1.0]
- for example: <x> = <absolute_x> / <image_width> or <height> = <absolute_height> / <image_height>
- **Note**: <x_center> <y_center> - are center of rectangle (not top-left corner)
- Calculation Formulas:
    - <x_center> = (x_min + x_max) / (2 * W)
    - <y_center> = (y_min + y_max) / (2 * H)
    - <width> = (x_max - x_min) / W
    - <height> = (y_max - y_min) / H

    Where W and H are equal to the image's width and height respectively.

For example for **img1.jpg** you will be created **img1.txt** containing:

```
1    0.716797    0.395833    0.216406    0.147222
0    0.687109    0.379167    0.255469    0.158333
1    0.420312    0.395833    0.140625    0.166667
```

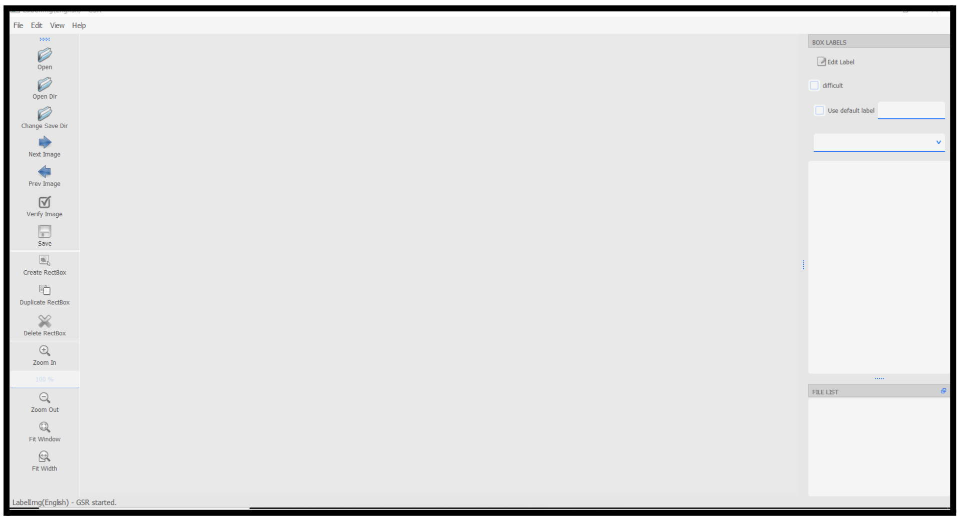
Example Annotation:



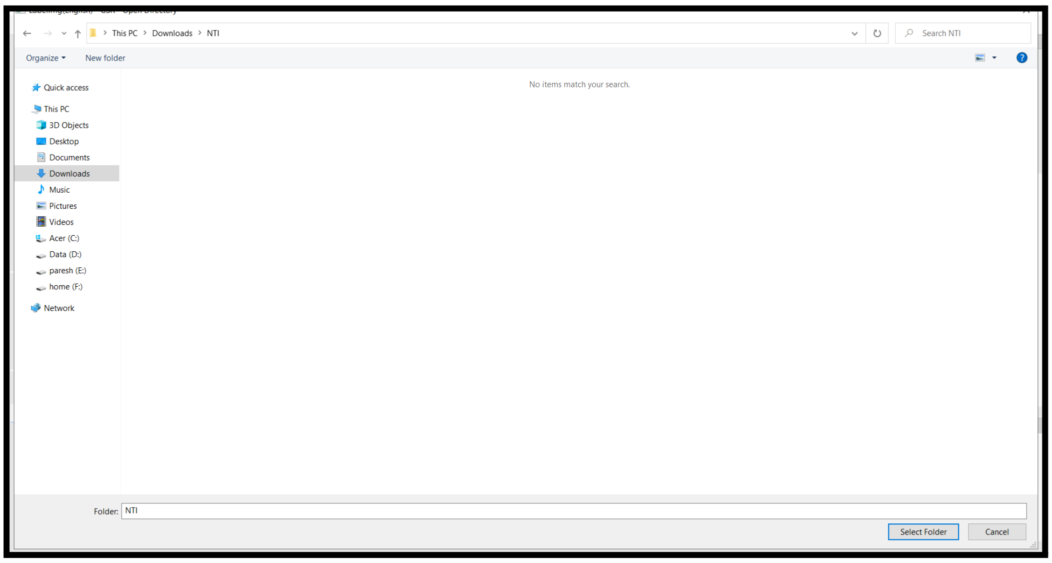
```
0 0.211111 0.495833 0.300000 0.641667
2 0.497222 0.497917 0.255556 0.645833
4 0.794444 0.495833 0.294444 0.658333
```

Steps to use

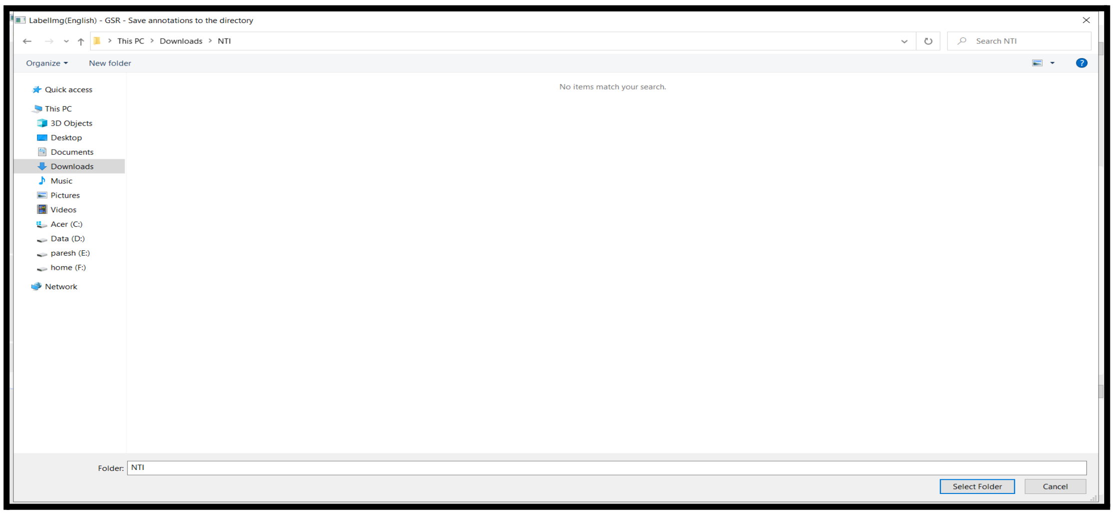
1. On launching our fork of Labelimg tool, the following will be the startup screen.



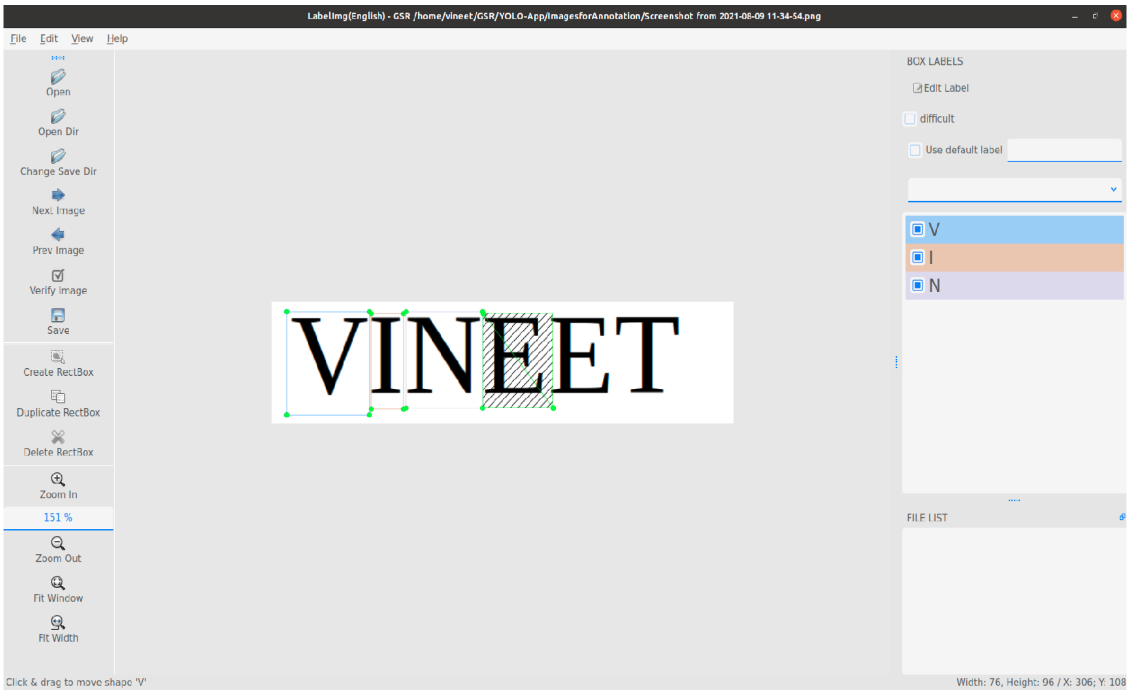
2. Click on the Open Dir button to select the folder where the images to be annotated are stored.



3. Click on the **Change Save Dir** button to select the folder where the annotations are to be saved.

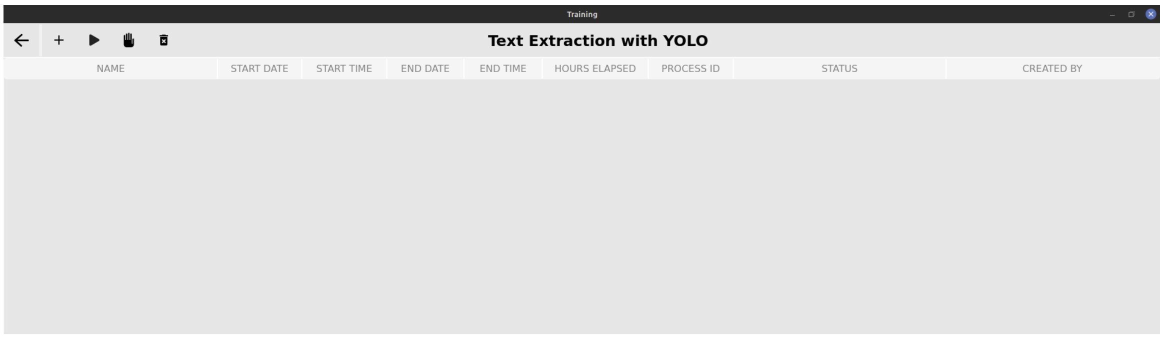


4. Start annotating by clicking and dragging to create a bounding box around the character. On click release, a dialog box will be launched that will allow you to choose the required class. Select class and click **'Ok'**



5. Finally click on the “Save” button to save the annotations.

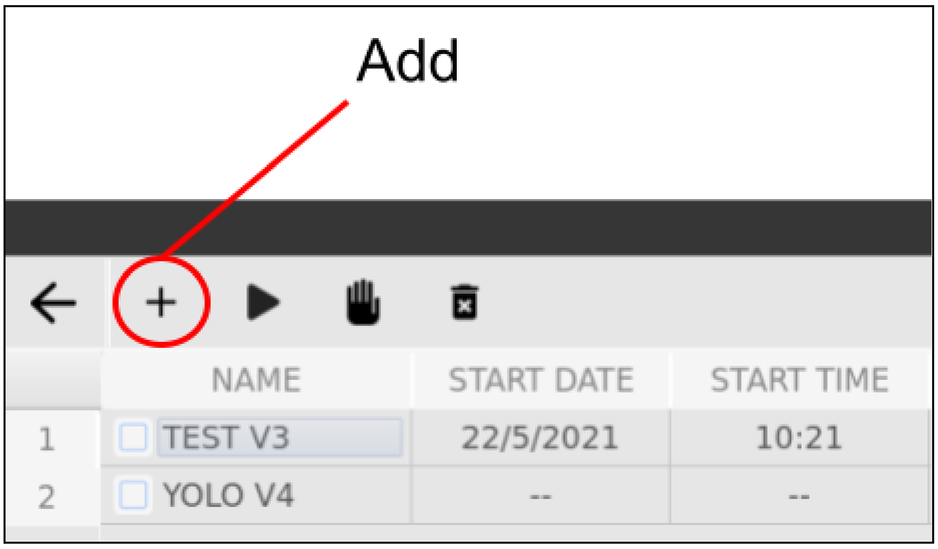
C. Training



Icons	Meaning
+	Create new training session

▶	Start Selected Training(s)
✋	Stop/Pause Selected Training(s)
🗑	Delete Selected Training(s)

1. Create Training Session



	NAME	START DATE	START TIME
1	TEST V3	22/5/2021	10:21
2	YOLO V4	--	--

a. Click on the **New Training** button in the toolbar at the top.

b. Fill out the following details:

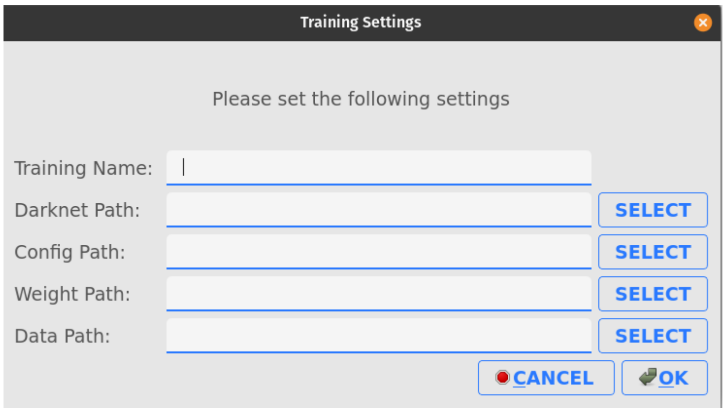
   i. Training Name (Must be unique)

   ii. Compiled Darknet Path

   iii. Weight Path

   iv. Configuration File Path

   v. Data Path

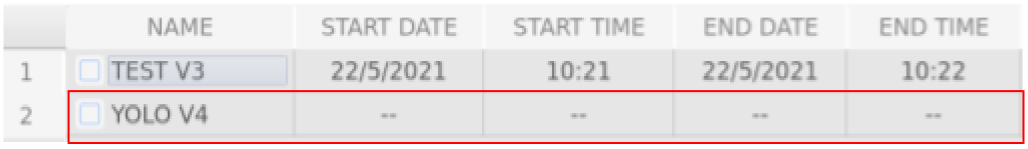


Training Settings

Please set the following settings

Training Name:		
Darknet Path:		SELECT
Config Path:		SELECT
Weight Path:		SELECT
Data Path:		SELECT

c. A new training session will be created and added to the table.

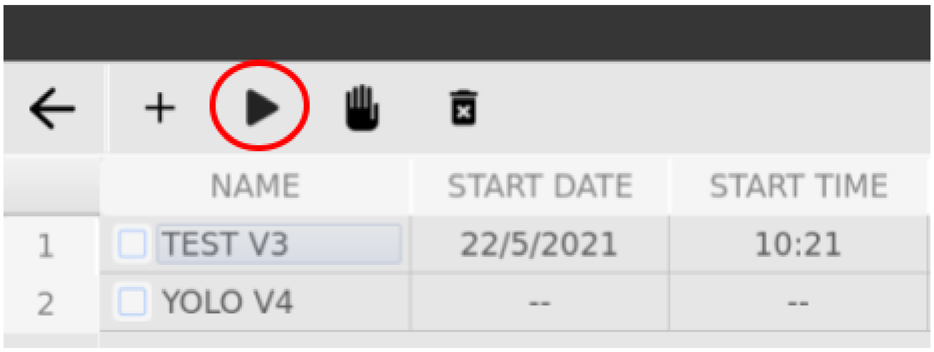


	NAME	START DATE	START TIME	END DATE	END TIME
1	TEST V3	22/5/2021	10:21	22/5/2021	10:22
2	YOLO V4	--	--	--	--

Note: **Training Name** must be unique. Failure to do so will result in the session not being created.

2. Start Training

a. Select the session that is to be started by clicking on the checkbox in the first column i.e the **Name** column.

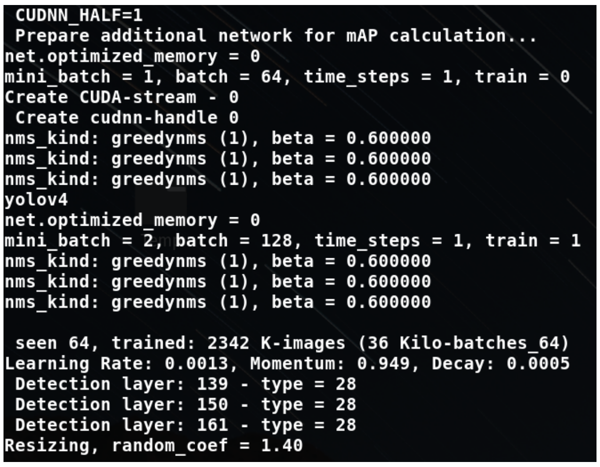


	NAME	START DATE	START TIME
1	TEST V3	22/5/2021	10:21
2	YOLO V4	--	--

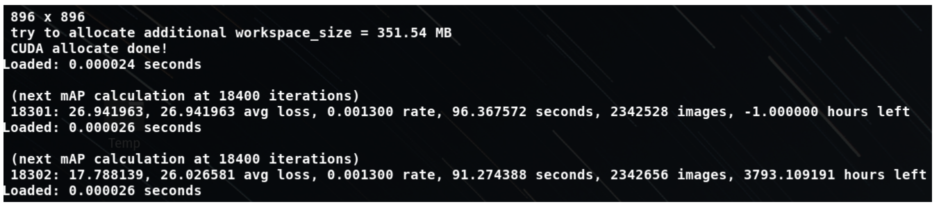
b. Click on the **Start Training** button in the toolbar.

c. A text file will be created in the same directory as that of the data file with trainingname_trainingInformation.txt as the file name. This file will contain all information w.r.t to the training such as iteration loss, average loss, current iteration etc.

Example :



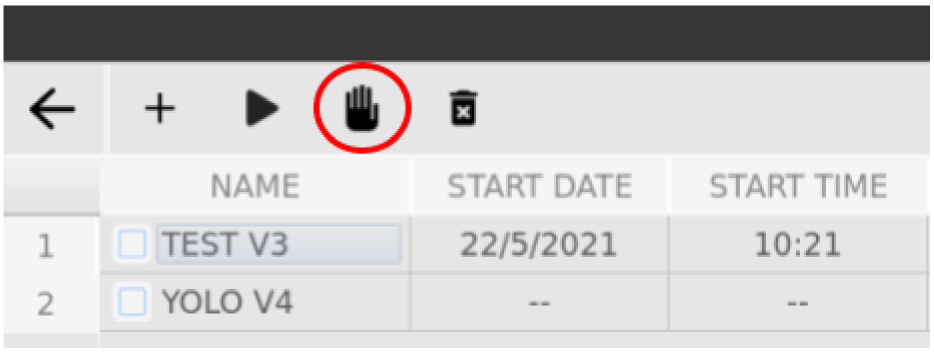
```
 CUDNN_HALF=1
 Prepare additional network for mAP calculation...
net.optimized_memory = 0
mini_batch = 1, batch = 64, time_steps = 1, train = 0
Create CUDA-stream - 0
 Create cudnn-handle 0
nms_kind: greedynms (1), beta = 0.600000
nms_kind: greedynms (1), beta = 0.600000
nms_kind: greedynms (1), beta = 0.600000
yolov4
net.optimized_memory = 0
mini_batch = 2, batch = 128, time_steps = 1, train = 1
nms_kind: greedynms (1), beta = 0.600000
nms_kind: greedynms (1), beta = 0.600000
nms_kind: greedynms (1), beta = 0.600000

 seen 64, trained: 2342 K-images (36 Kilo-batches_64)
Learning Rate: 0.0013, Momentum: 0.949, Decay: 0.0005
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
Resizing, random_coef = 1.40
```



```
 896 x 896
 try to allocate additional workspace_size = 351.54 MB
 CUDA allocate done!
Loaded: 0.000024 seconds

 (next mAP calculation at 18400 iterations)
 18301: 26.941963, 26.941963 avg loss, 0.001300 rate, 96.367572 seconds, 2342528 images, -1.000000 hours left
Loaded: 0.000026 seconds

 (next mAP calculation at 18400 iterations)
 18302: 17.788139, 26.026581 avg loss, 0.001300 rate, 91.274388 seconds, 2342656 images, 3793.109191 hours left
Loaded: 0.000026 seconds
```

3. Stop Training

a. Select the session that is to be stopped by clicking on the checkbox in the first column i.e the **Name** column. (Note: Training has to be running inorder to be stopped)

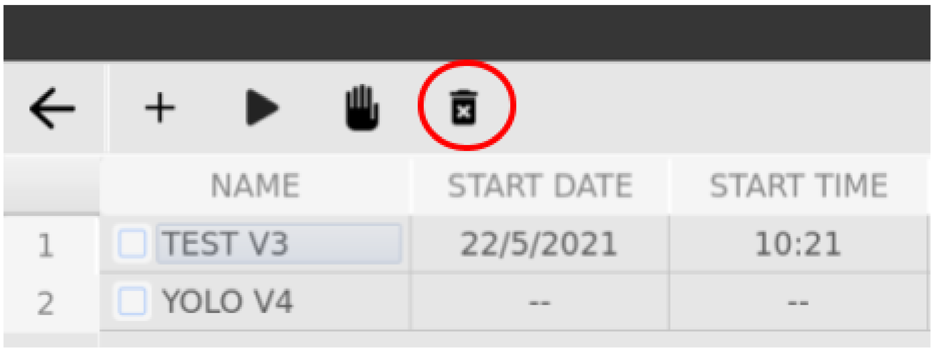


	NAME	START DATE	START TIME
1	☐ TEST V3	22/5/2021	10:21
2	☐ YOLO V4	--	--

b. Click on the **Stop Training** button in the toolbar.

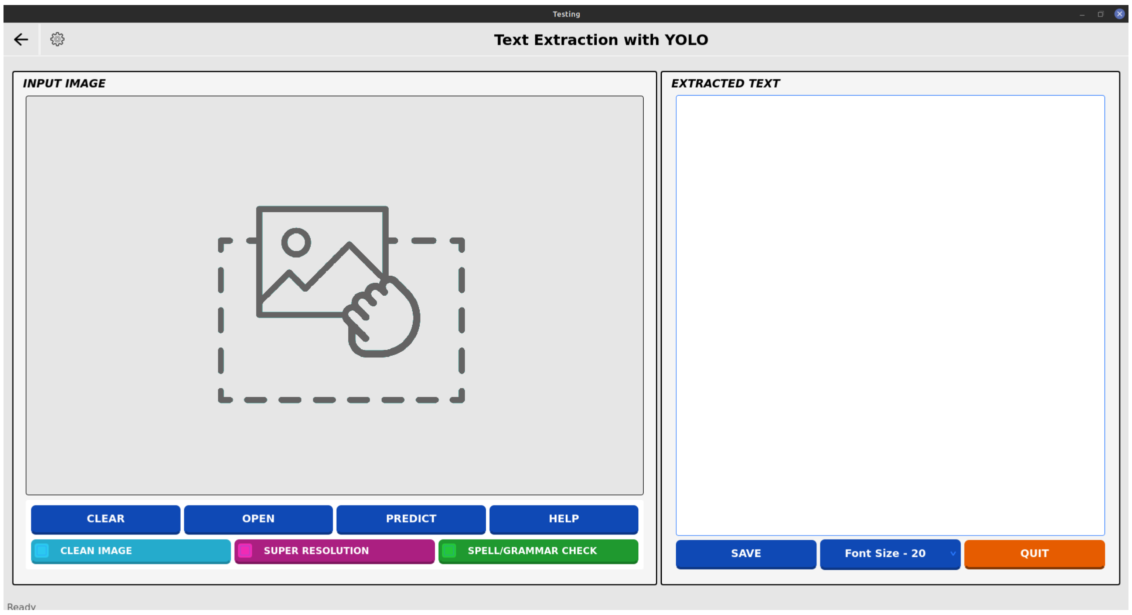
4. Delete Training

a. Select the session that is to be deleted by clicking on the checkbox in the first column i.e the **Name** column.



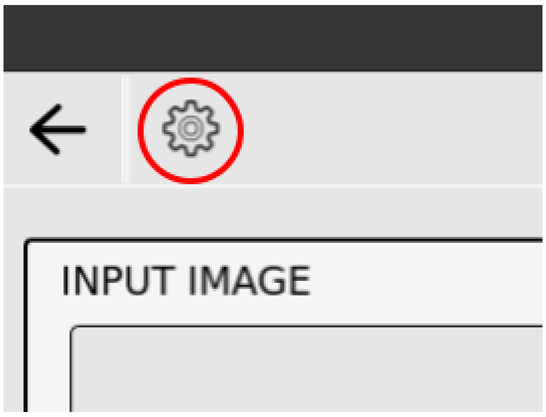
b. Click on the **Delete Training** button in the toolbar.

D. Inference



Icons	Meaning
⚙	Set Various Paths required for Inference

1. Click on the **Inference Settings** button in the toolbar.

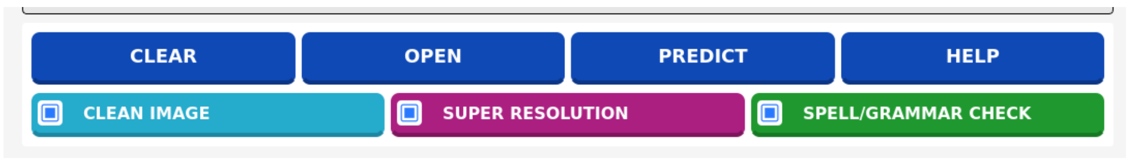


2. Set the following paths:
    a. Darknet Path
    b. Weight File Path
    c. Configuration File Path
    d. Data File Path
3. Open image on which inference is to be made using the **Open** button located at the bottom.
4. Click the **Predict** button and wait for output to be displayed in the **Output Panel**.

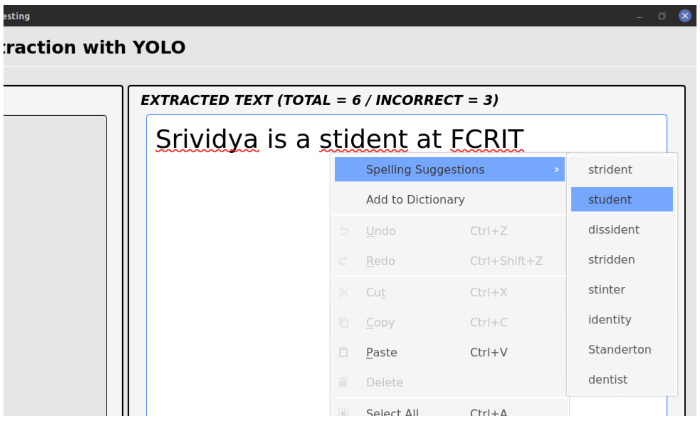
    Note - If one receives an error

    ```
    Couldn't open file: space.names
    ```

    It can be solved by locating the space.data file and setting the names variable to the right path (pointing to the space.names file).
5. The Super Resolution check box can be set for images with a small font size. The Clean Image check box can be set for camera images that have noise. The Grammar check checkbox can be set to include Gingerit for automatic spelling and grammar correction on output (Only for english text).
6. Click on **Help** for user guidance.



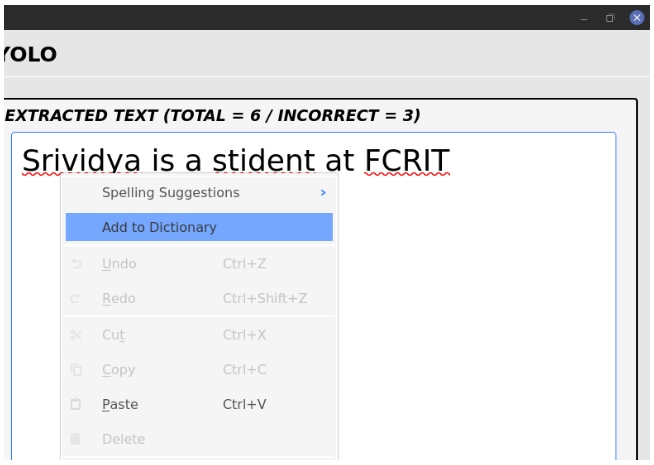
7. The output text area comes with a complete manual spell correction for multiple languages and ability to add words into a Personal Word List.



EXTRACTED TEXT (TOTAL = 6 / INCORRECT = 3)

Srividya is a stident at FCRIT

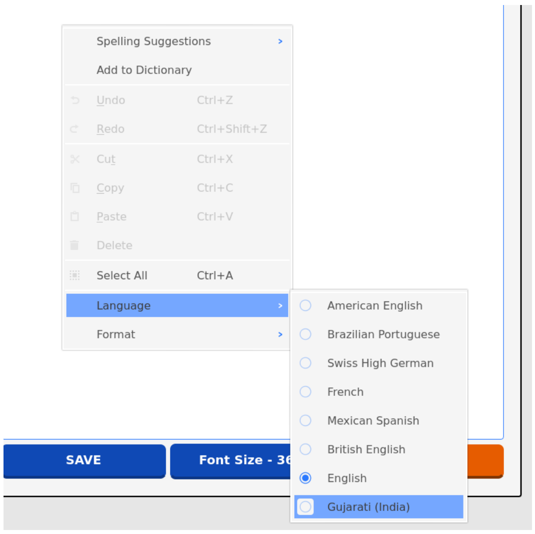
Adding to Personal word list.



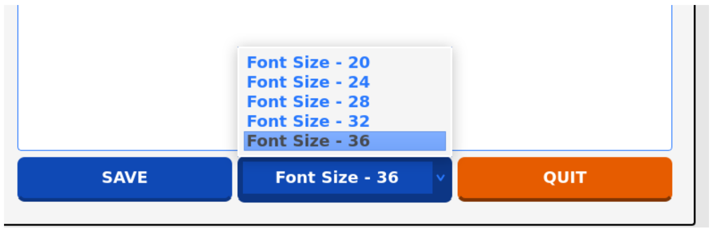
EXTRACTED TEXT (TOTAL = 6 / INCORRECT = 3)

Srividya is a stident at FCRIT

Switching Language.



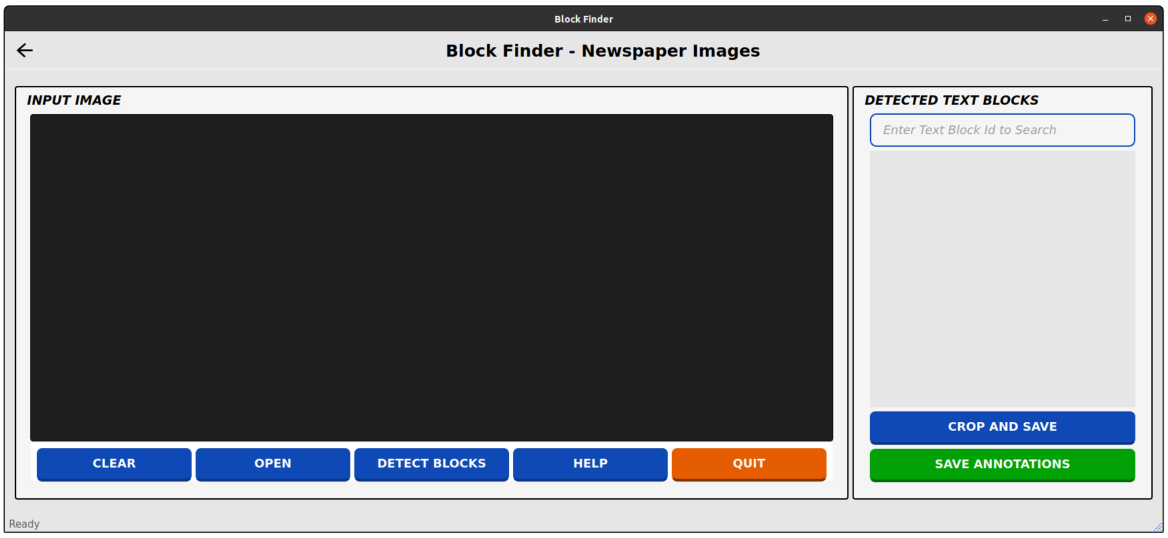
8. The font size of the text can be changed with the drop down.



<u>Note:</u> This setting has to be done only once unless changes are to be made to the paths. Settings are stored for future startups.

E. Block Detector

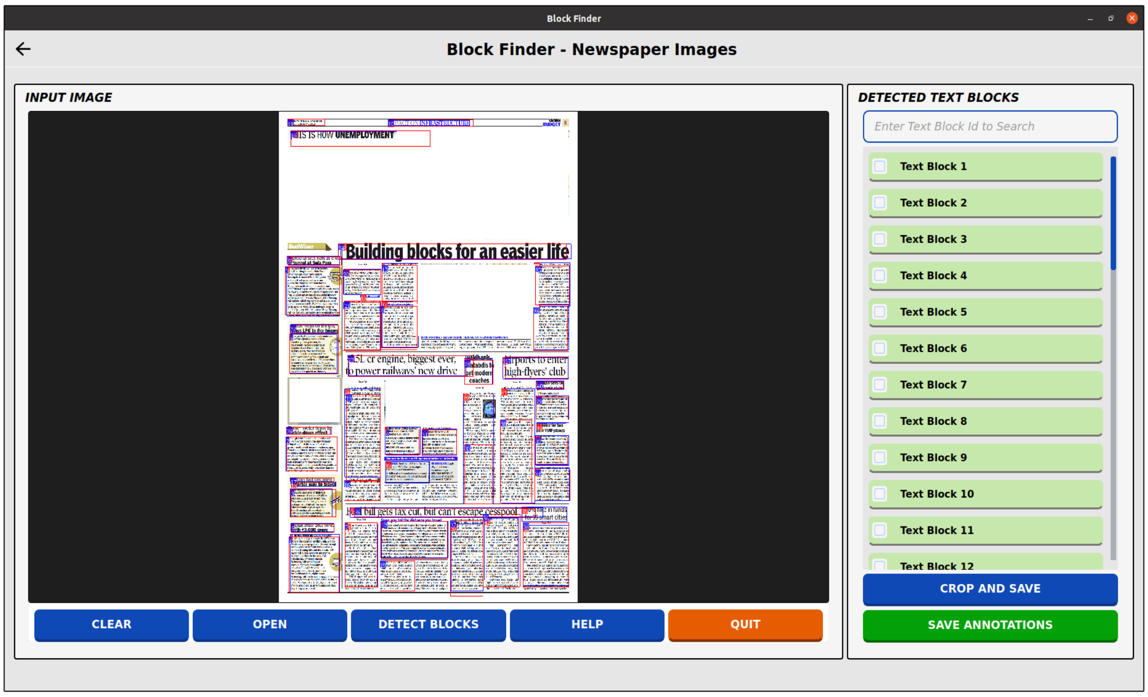
This module detects various blocks in a newspaper image clipping. It draws bounding boxes along text blocks in a newspaper which can further be sent for text extraction. The library used for this functionality can be found at this link and the dataset can be found here.



1. Open image on which block detection is to be done using the **Open** button located at the bottom.

2. Click on **Detect Blocks** and wait for output to be displayed on the output panel on the right.

   Note - The first time Block detector is run, it may take a few minutes to download the PrimaLayout LayoutParser model. Please open the terminal on which the program was run to see the updates regarding the download. But this is only the case in the first run of the Detect Blocks button.



3. Click on **Help** for user guidance.

Autocorrect

The application has a checkbox to apply autocorrect while the output is being generated in the testing module. The library used for this functionality can be found at this link.

GingerIt

The application has a checkbox to apply gingerit which aids in correcting spelling and grammar based on the context of the sentences. The library used for this functionality can be found at this link.

Note - Upon running Gingerit one may face a JSONDecodeError. As mentioned in the link it is a cloudflare antibot which is blocking the request. It can be solved by running the following command -

```
(gsr_env) $ pip install cloudscraper
```

And then locating the gingerit.py file in the path -

```
gsr_env/lib/python3.7/site-packages/gingerit/gingerit.py
```

To this file an import must be added -

```
import cloudscraper
```

And line number 16 should be replaced from

```
session = requests.Session()
```

to

```
session = cloudscraper.create_scraper()
```

Super Resolution

Image super-resolution (SR) is the process of recovering high-resolution (HR) images from low-resolution (LR) images. The application has a checkbox to apply super resolution and the details can be found here.

GSR Training Sessions

2020-21 Batch

This directory contains the following files related to YOLO training's performed by GSR 20-21:

- Configuration Files
- Weight Files
- Data and Name Files
- Datasets

The following sections describe each subdirectory/training session present in the parent directory.

1. Handwritten English (YOLO V3)

This model was trained on images of the dataset available here IAM Handwriting Database as well as images of college assignments to detect handwritten english.

To access the IAM Handwriting Database, **yofilo1086@opetron.com** can be used as username and password.

Training Details

- **Images used**: 111
- **Number of classes:** 79
- **Training time**: 72 Hours +
- **Iterations**: 32885
- **Average Loss**: 13.43

2. YOLO V3

This model was trained to detect typewritten english. The dataset consists of images from digital versions of newspapers, books, college experiments and a few python library generated images.

A **'Space'** class was added to the class list in order to detect spaces between words. This class is not available in the handwritten english detection model. Rest of the classes are the same.

The model was first trained for **160,000 iterations on 443 images**. Hence the weight files have the name as **space_443.weights**. However the dataset size was later increased to **490** and this model continued to train on the newer images.

Training Details

- **Number of Images**: 490 (Earlier 442 Images)
- **Classes**: 80
- **Training Time**: 553 Hours (Old Weights) + 92 Hours
- **Training Stopped**: 8:35 PM 30th March 2021 (Older Weights = 9:47 AM Sunday 14th March)
- **Iterations**: 160000 + 26800
- **Average Loss**: 6.92 (Old Weights = 4.93)

3. YOLO V3 5L

This model was trained to detect typewritten english and as an improvement over the V3 model mentioned above.

The standard **V3** model consists of **3 YOLO regions** (found in config file) whereas the **V3 5L** model consists of **5 YOLO regions**. This model was specifically made to detect small as well as large objects.

Training Details

- **Number of Images**: 490

- **Classes**: 80
- **Training Time**: 563 Hours 15 Minutes
- **Training Stopped**: 10:10 PM 28th April 2021.
- **Iterations**: 19301
- **Average Loss**: 16.06

4. YOLO V4

This model was trained to detect typewritten english and as an improvement over the V3 model mentioned above. This model consists of **3 YOLO regions**.

The dataset for this model was increased from **490 to 525 images**. Using the standard 80-20 training-testing split, **427 images** were used for **training** and **98 for testing.** Splitting was not performed for previous models.

Of all the models, this model performs the best.

Training Details

- **Number of Images**: 427
- **Classes**: 80
- **Training Time**: 1,116.50 Hours (1:45 PM 16th June 2021)
- **Training Stopped**: Not Stopped
- **Iterations**: 105,004
- **Average Loss**: 13.58
- **Accuracy**: 88.78 % (On 98 images)

2021 - 22 Batch

This directory contains the following files related to YOLO training's performed by GSR 20-21:

- Configuration Files
- Weight Files
- Data and Name Files
- Datasets

The following sections describe each subdirectory/training session present in the parent directory.

1. 11 Classes

This model was trained to detect typewritten Gujarati with 11 classes. Out of those 5 are considered simple and 5 complex (visually) and the complex classes have more images in the dataset. The dataset consists of images generated from the TRDG library. This model has the best performance. *It uses pretrained weights to kick start the training.* The details of this model are as follows -

- **Number of Images**: 800
- **Classes**: 11
- **Training Started**: Monday 20 December 2021 11:33:17 PM IST
- **Training Stopped**: 21 December 2021 07:45:34 PM IST
- **Training Time:** 28 hours
- **Iterations**: 6014
- **Average Loss**: 0.37
- **Validation Accuracy**: 99.76 % (On 176 images)

2. 56 Classes

This model was trained to detect typewritten Gujarati with 55 classes. The dataset consists of images generated from the TRDG library. *It does not use pretrained weights to kick start the training.* However it should be noted that this model does not perform well when it comes to Testing accuracy. A large number of images could not be included for validation accuracy because of an OOM error in rank calculation on the server. The details of this model are as follows -

- **Number of Images**: 1252
- **Classes**: 56
- **Training Started**: Thursday 03 February 2022 09:08:25 PM IST
- **Training Stopped**: Friday 04 February 2022 10:37:25 AM IST
- **Training Time:** 13 hours
- **Iterations**: 10013
- **Average Loss**: 0.90
- **Validation Accuracy**: 99.41 % (On 126 images)

3. 124 Classes

This model was trained to detect typewritten Gujarati with 124 classes. The dataset consists of images generated from the TRDG library. *It does not use pretrained weights to kick start the training.* However it should be noted that this model does not perform well when it comes to Testing accuracy. A large number of images could not be included for validation accuracy because of an OOM error in rank calculation on the server. The details of this model are as follows -

- **Number of Images**: 3345
- **Classes**: 124
- **Training Started**: Sunday 13 February 2022 11:01:00 PM IST
- **Training Stopped**: Monday 14 February 2022 09:54:17 PM IST
- **Training Time:** 22 hours
- **Iterations**: 21386
- **Average Loss**: 0.23
- **Validation Accuracy**: 98.66 % (On 126 images)

Additional Weight Files

Some additional weight files which were not downloaded to our laptops for the above mentioned training sessions (except Handwritten English) can be found in the following directories on the TIFR GPU MACHINE (RTX 3090):

2021 - 22 Batch

- **YOLO V3:**
    - **For 443 Images**: `/home/fcrit/work/training/weights/old_weights`
    - **For 490 Images**: `/home/fcrit/work/training/weights/`
- **YOLO V3 5L:** `/home/fcrit/work/training/YOLOV3_5L/weights/`
- **YOLO V4:** `/home/fcrit/work/train_darknet/darknet2/YOLOV4/weights/`

2021 - 22 Batch

- **11 Classes:** `/home/fcrit/fcrit/gsr/train_darknet/darknet2/YOLOV4_GU/weights`
- **56 Classes:** `/home/fcrit/fcrit/gsr/train_darknet/darknet2/YOLOV4_GU_56/weights`
- **124 Classes:** `/home/fcrit/fcrit/gsr/train_darknet/darknet2/YOLOV4_GU_124/weights`
- **362 Classes:** `/home/fcrit/fcrit/gsr/train_darknet/darknet2/YOLOV4_GU_362/weights`

Measuring Model Accuracy

For measuring any YOLO models accuracy, it is recommended to divide the dataset into train-test split using the standard 80-20 ratio or similar other ratios.

The darknet terminal command has a flag/argument named ‘map’. When training with mAP, mAP(Mean Average Precision) calculation is done for each 4 epochs.

For more information, see here

Google Accounts

2020 - 21 Batch

Multiple google accounts were created to perform training on Google Collab before GPU was provided to us. These accounts can now be used for storage purposes.

The accounts IDs are:

- collabtrainingacc@gmail.com
- collabtrainingacc2@gmail.com
- collabtrainingacc3@gmail.com
- collabtrainingacc4@gmail.com

The password for all these accounts is **tifrtraining*9.**

The Google Drive of the first account from the above list contains weight files from all our training sessions as well as the source code.

Alternatively this link (Same as the one in 'GSR 20-21 Training Sessions' section) can be used to access datasets, configurations, data, name files and weight files created by us.

2021 - 22 Batch

All the data and code used in the development of GSR can be found on this gmail account -

gsr.fcrit2022@gmail.com

The password for this account is **gtrec2022**

Alternatively (Same as the one in 'GSR 21-22 Training Sessions' section) can be used to access datasets, configurations, data, name files and weight files created by us.

Important Links

- Darknet and YOLO
- AlexeyAB's Darknet
- Pre/Post Darknet Compilation Replacement (Instructions)
- YOLO-App (Google Drive)
- YOLO-App (TIFR Gitlab)
- LabelImg Standalone Version (Google Drive)
- LabelImg Standalone Version (Gitlab)
- Autocorrect Python Library (Github)
- GSR 20-21 Training Session Files
- Training a custom object detector
- Measuring Model Accuracy
- Training YOLO V3 on Google Collab (Youtube)
- Training YOLO V4 on Google Collab (Youtube)
- Link for Darknet Repository (In case of errors)

Future Work

A functionality to translate the English output into Gujarati was decided. Google offers a free python library which does the work for us and can be used without limits. The library can be found HERE.

Another way is to use IndicTrans.

IndicTrans is a model trained on samanantar dataset.

Samanantar is the largest publicly available parallel corpora collection for Indic languages : Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Punjabi, Tamil, Telugu. The corpus has 49.6M sentence pairs between English to Indian Languages.

Two models are available which can translate from Indic to English and English to Indic. The model can perform offline translations for 11 languages: Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Punjabi, Tamil, Telugu.

This project can be found at : https://github.com/AI4Bharat/indicTrans

Contacts

Srividya Subramanian - srividya.ssa@gmail.com

Vineet Kekatpure - vineet.kekatpure@gmail.com

Gladina Raymond - rrcg2000@gmail.com