Software Engineering Practice User Interface

# API Design

The API uses an os.system() call from the parent user interface script. This uses the operating system’s terminal-style interface (Windows command prompt/Linux terminal etc.) to execute a command that is passed as an argument. This command is a string built within the parent UI program with parameters being passed in consisting of the path of the source image, the path of the weights file, the confidence threshold and the size of the image. All four of these parameters are configurable from the user interface, with all but the source image path being pre-populated. This means that if the child script were to change completely, it could still be fully interacted with should the API remain the same.

All variables concatenated into the execution command are validated. This consists of checking that the paths given for the source image and weights file actually exist, as well as the confidence threshold being a float above 0 and the image size being an integer above 0. If any of these validations fail, the command is not executed. This should significantly help prevent any code injection attacks, however since the program is executed as the current user they should not be able to access anything they could not access via terminal anyway.

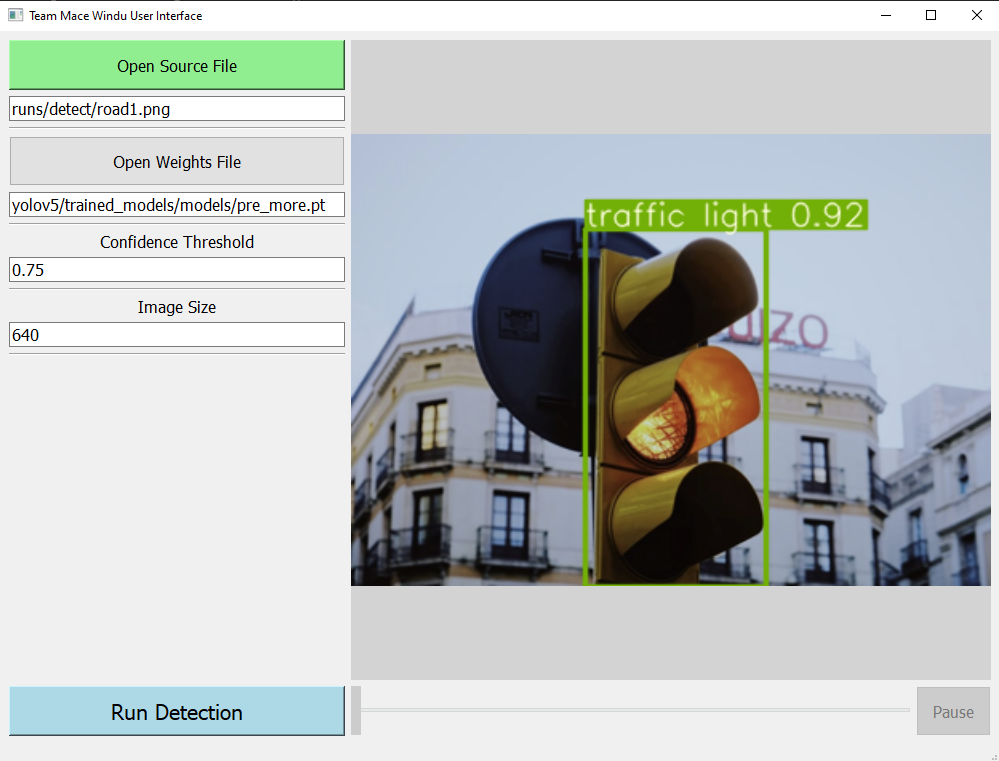
The testing script could be imported directly into the parent UI script, however this would require implementation from the testing script, and would be much more difficult to maintain a working API should the testing script require significant change, such as using a different algorithm altogether.

The use of a separate weights file also means that several different models can be compared quickly, without the need to make changes to any scripts. This also means that models can be more easily distributed as long as each user has a working UI framework.

# UI Design

The user interface was designed with QT Designer, a popular user interface designer compatible with several languages including C++, JavaScript and Python. It uses Qt Widgets to create interfaces for use with the Qt API and Qt GUI application development framework. It allows designs to be very easily created with a graphical designer, that can be looked at immediately from within the designer app. The interface can then be compiled for use via the PyQT package, creating a .py file that can easily be imported and used within a program. This also means that the UI can change after the program is created, and all functionalities will remain as long as no widgets are deleted or renamed.

Python’s own tkinter was also considered, however given how much more difficult it is to change how the interface looks after the fact, as well as requiring that all objects be created programmatically, PyQT and Qt designer were chosen instead for their ease of use.



Included above is a screenshot of the final version of the user interface. It can be seen that an image of a traffic light has been selected as the source image, and a detection completed that has resulted in a 92% certainty that the object in the image is a traffic light.

A large green button in the top left was used to draw attention as it is the first thing a user must do when using the app. This will open an operating system specific “browse” window to allow the user to graphically select an image or video file rather than manually type the path, however typing the path is also an option.

The weights file is the default grey as it is pre-populated and should not need to be manually specified if the file structure is left unchanged. Confidence threshold and image size are also automatically pre-populated for ease of use, and the run detection button is a different colour with bold text to draw attention also. The video scrubbing bar and the pause button are automatically disabled if the selected content is not a video, and re-enabled if a video file is selected. Setting the video progress bar is done within the dedicated video thread as it needs access to both the progress bar’s setValue() function as well as the current frame count stored within the thread. The progress bar is done by setting the max value to the number of frames in the video, and then having the current value as the current frame. This is better as if it were to be normalised to between 0 and 100, it could display the progress ok however when trying to move a very small increment in a very large video, you could only move (#frames / 100) / fps, seconds. So, for example in a 5-minute video at 24fps (7200 frames total), the smallest increment possible would be (7200 / 100) / 24, or 3 full seconds. This worsens with a longer video such as a 1 hour video at 24 fps has a smallest increment of 36 seconds, simply unusable for fine scrubbing.

# Solution Integration

Clicking the Run Detection option opens an operating system specific terminal window showing the progress of the detection for the specified source content. For images, this window will be open very briefly, however it may remain open much longer for larger content such as long, high resolution videos. A progress indicator is displayed in the terminal, in the form of showing the current frame out of the total number of frames for a given video, as well as the class name of any objects that appear in that frame. When the detection is complete, the processed content is automatically displayed in the media viewing window, with the scrub bar and pause button enabled or disabled for videos and images, respectively.

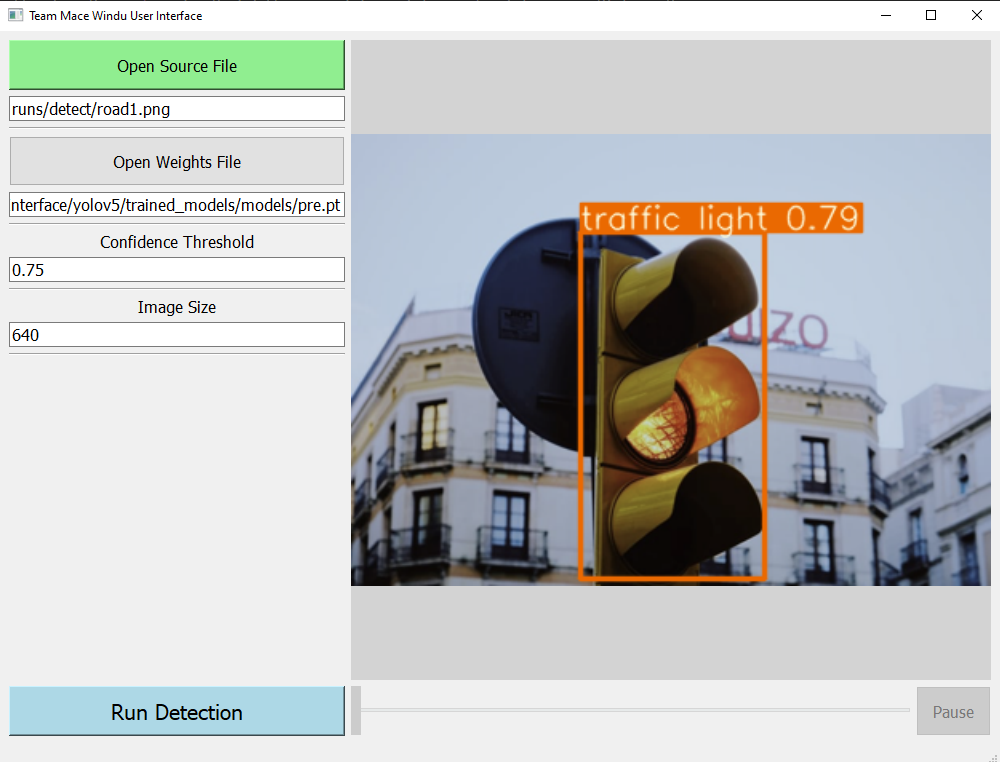
Any trained weight file can easily be used with the solution and so comparing accuracy on the same image with different models is very easy. The command calling the detection itself runs from within the main thread while the video player runs in its own thread, as if it were done within the main thread the entire UI would freeze until each frame of the video had been processed, even though it would not actually be seen as the UI would not be updated.

As for the training section of the solution, all parameters for training can be configured such as image size, batch size, epoch count, the dataset used and the weights file. Optionally, no weights file can be used, as well as the option of fine tuning an existing weights file or continuing training from an already existing weights file. Data is automatically logged and graphed on wandb.ai where it can be easily seen and compared before manually testing. The graphs here are also helpful to identify which models may be overfitting data.

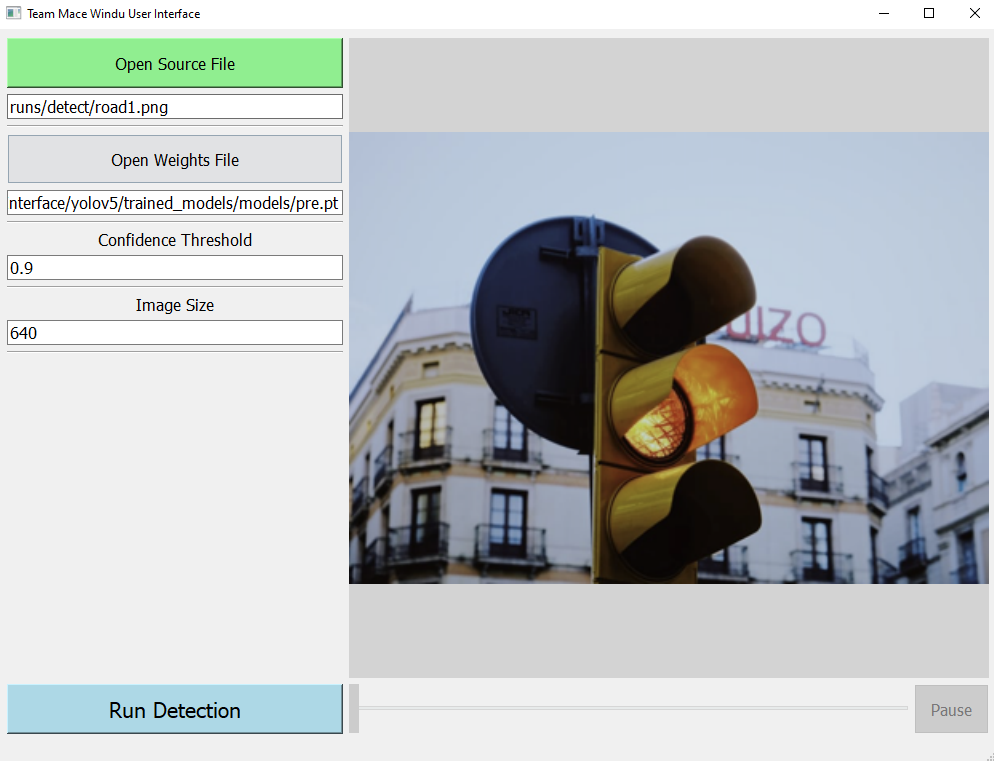
# User Testing

## 

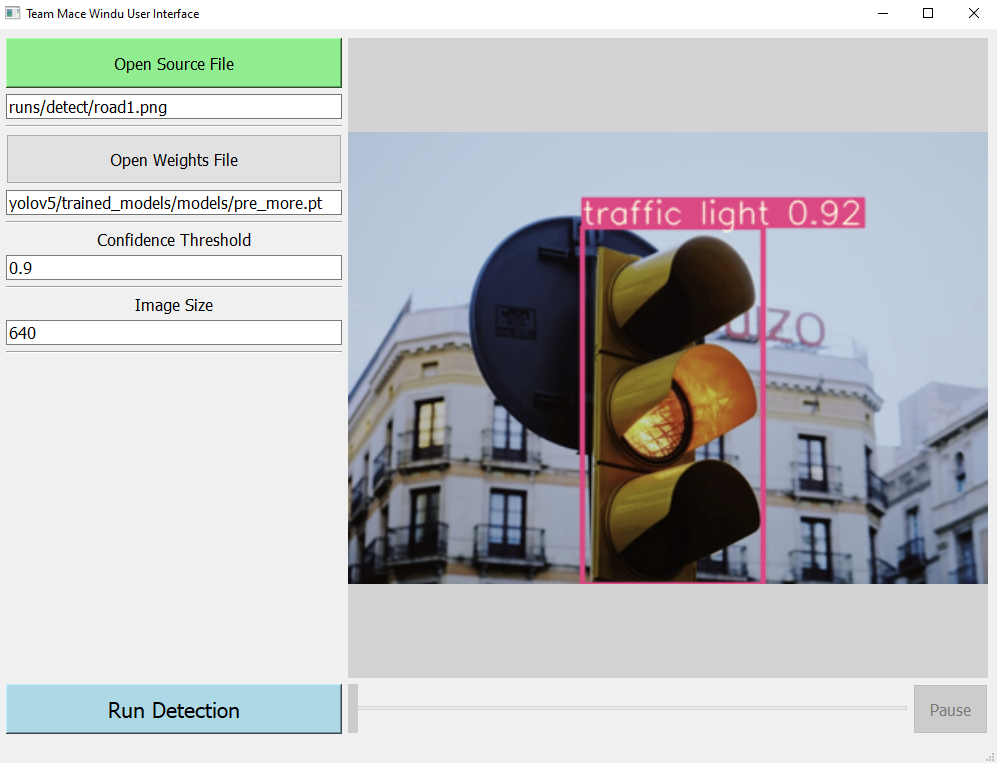
Seen Here is a successful detection of a traffic light using a premade then fine tuned model, an image of a traffic light with detection confidence of 92%, a confidence threshold of 75% and an image size of 640. This is the ideal final scenario and allows the user to clearly see any features in the image that the solution has detected using a large but readable font size for the class label and confidence and a good border width for the feature bounding box.



Above is a similar outcome, however it uses a different model (pre.pt vs pre\_more.pt (pre\_more.pt is the same fine-tuned model after continued training)). It can be seen that the confidence output from the detection program is not as high, indicating that the model is much more uncertain about what it is actually being displayed in this image.

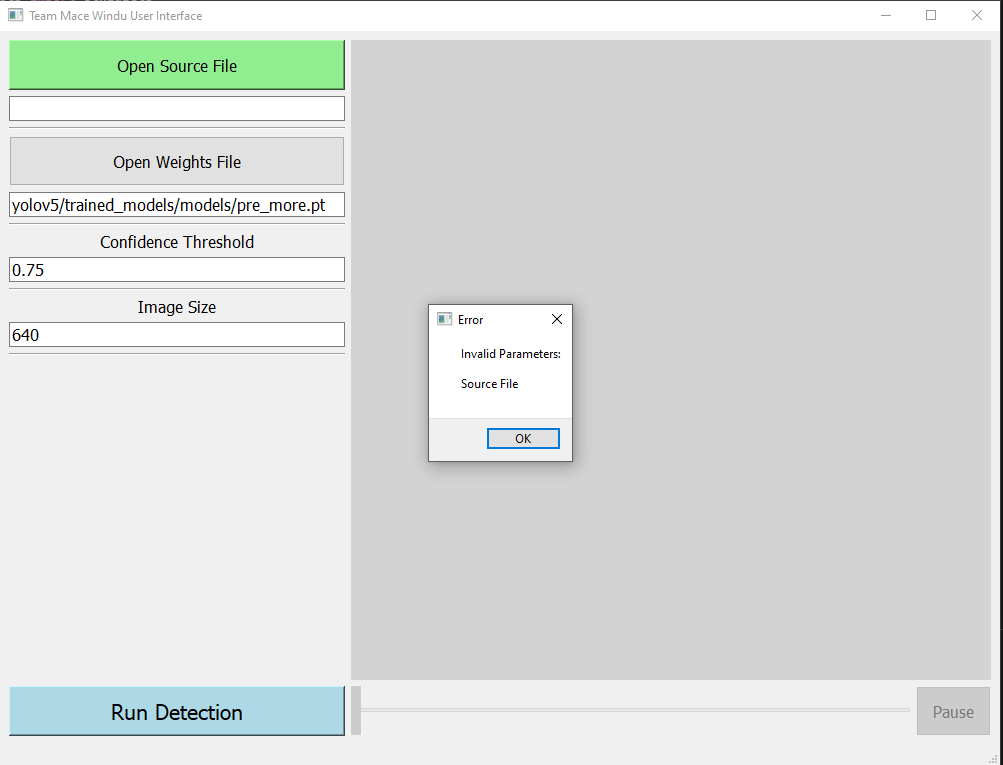


Keeping the same, weaker model by increasing the confidence threshold means that the weaker model no longer classifies this as a successful detection, whereas for the stronger model the traffic light is still detected, as seen below.

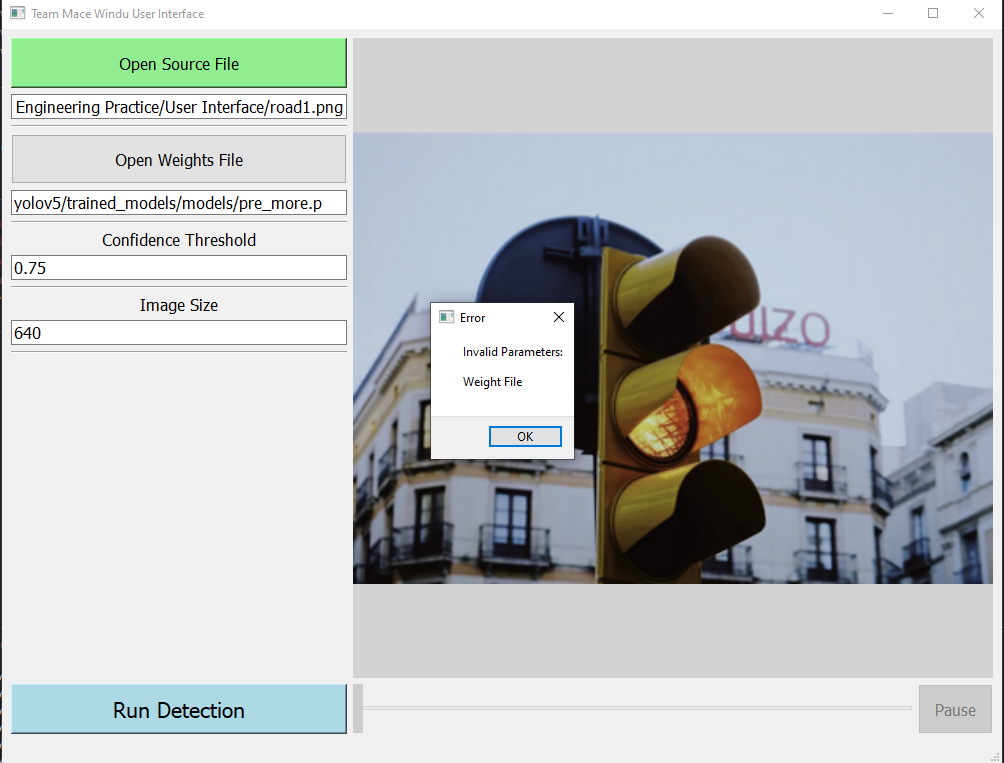


Seen in the next few images are all examples of user input parameters failing the validation process used when creating the command to call the detection program.

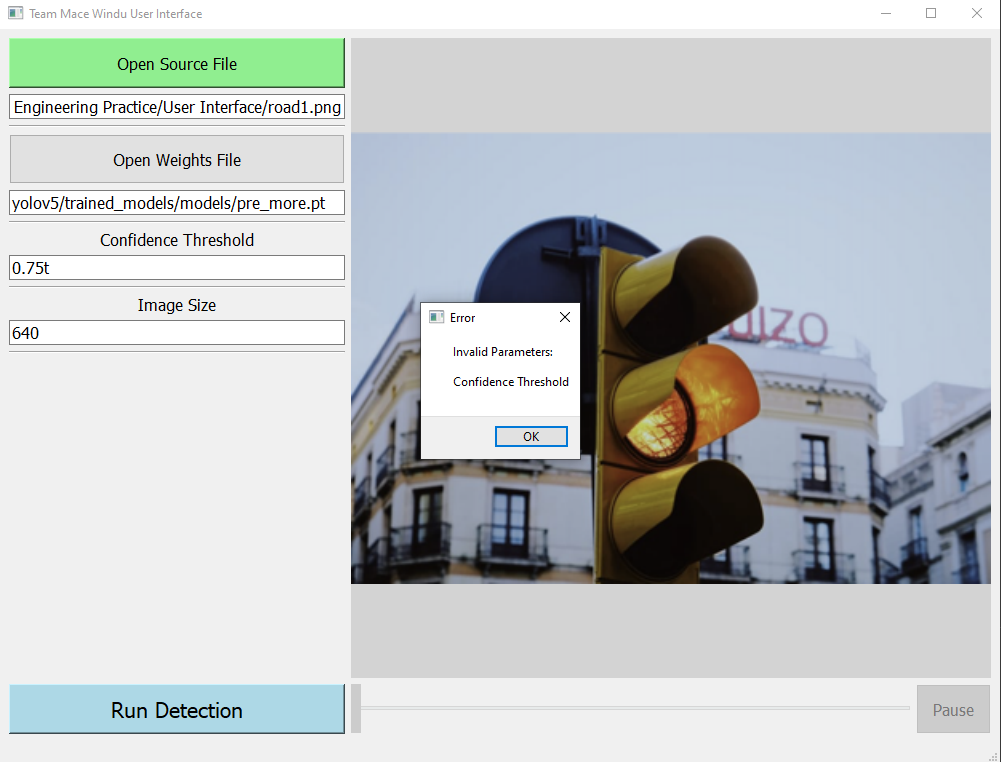
Invalid source file:



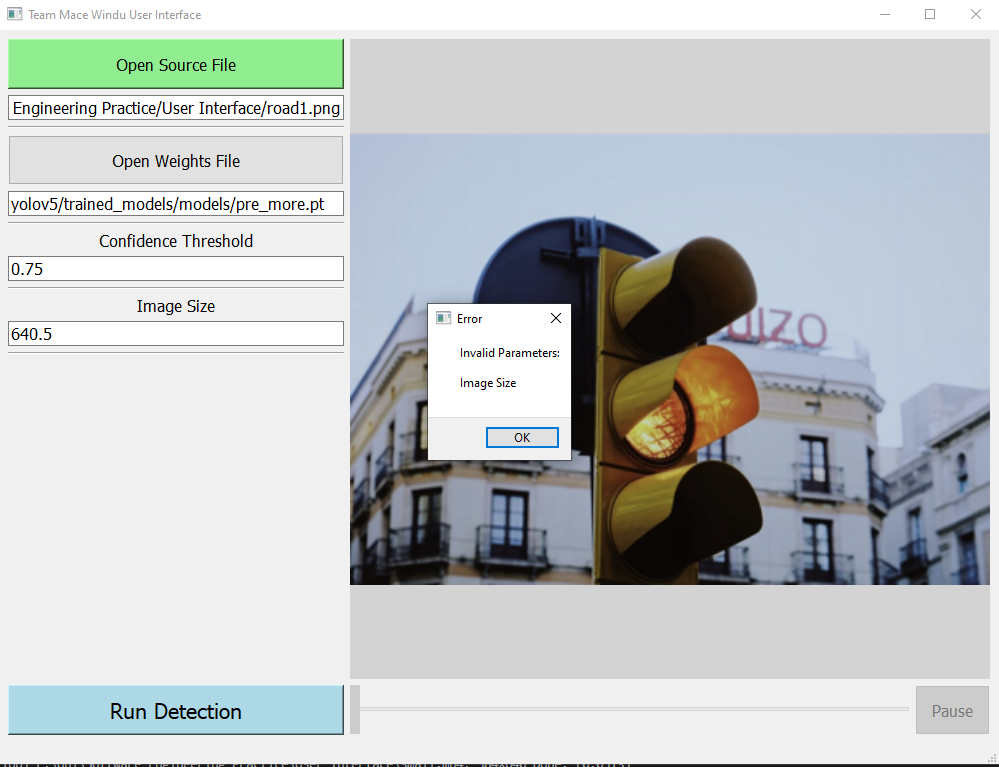
Invalid weight file:



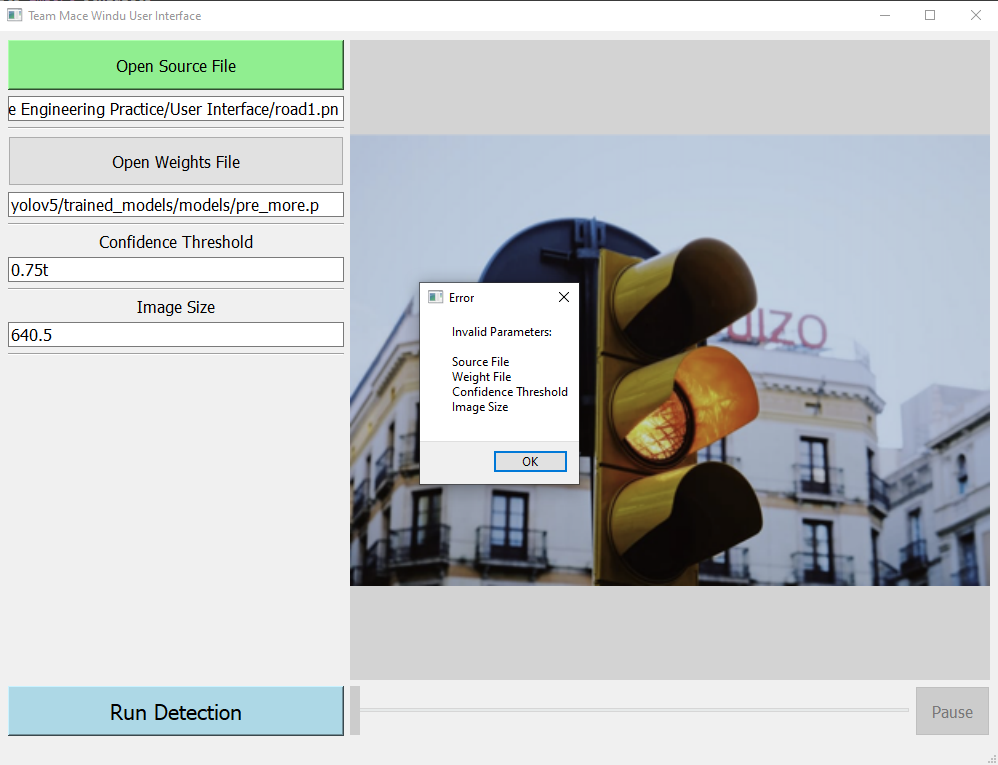
Invalid confidence threshold:



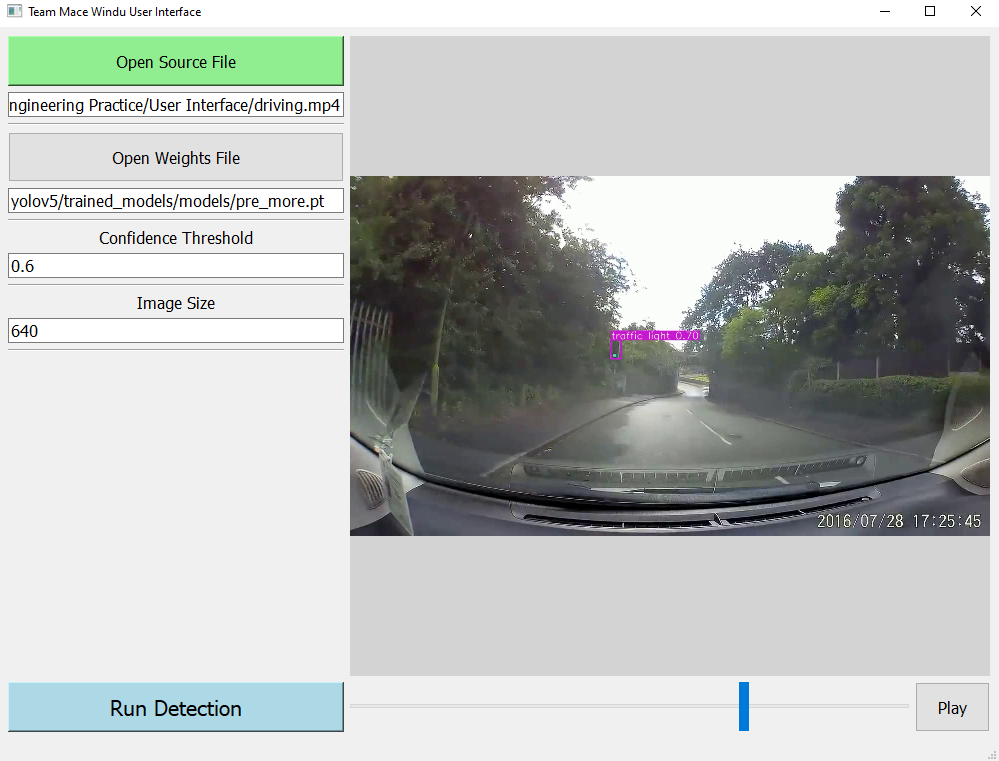
Invalid image size:



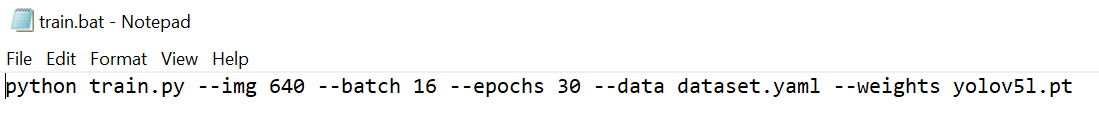
Invalid everything:



Displaying all invalid parameters in a single message box is both much easier and cleaner looking compared to displaying an error message box for each invalid parameter, which would very quickly “clog-up” the screen.



Above is a snapshot from a paused processed video of a dashcam from a car detecting a traffic light ahead through the windscreen, in a relatively complex area with wet conditions and a large number of trees.



Here is a screenshot from the train.bat file, an executable file used to begin training with some pre-set parameters. It can be seen that an image size of 640 pixels has been selected, a batch size of 16, 30 epochs/generations of training, a dataset has been specified and a pre-existing weights file is to be used. The weights file name is yolov5l.pt: the large variant of the pre-existing yolov5 weights file intended for use by fine tuning them to a dataset, often resulting in better accuracy with less overfitting.