# PlaqSeg

## Instruction Manual

V0.1.0

Leo G Skingley

## Contents



# Introduction

Plaq seg is a mobile application that has been developed to provide automated plaque counting for phages on petri dishes. This application aims to speed up this common assay in the lab, and to standardize size and accuracy measurements. PlaqSeg is provided as a mobile application, desktop application and python package for use in HPC environments. Plaq seg is free, but donations to the Phage Collection Project are welcome.

PlaqSeg has two main methods for plaque counting: OpenCV pipelines, with a CNN filter, and a YOLO26 segmentation model. PlaqSeg can take any image format, for example .jpg, .png, .tiff etc, however we recommend lossless formats or bitmaps if possible.

# CV Pipeline Overview

The CV pipeline is only available on mobile devices, and uses a multi step pipeline to ensure accurate and reliable results. Images can be taken with sufficient resolution with any relatively modern device.

The first step of the pipeline is to crop the image down to just the petri dish – to accomplish this we use Hough Circle Detection. This removes any debris, or other noise that may interfere with subsequent steps. This is also how we calibrate for plaque size detection – if we assume that the user's petri dish is the standard 100mm, then we are able to extrapolate the number of pixels per mm, which enables us to estimate the size of plaques.

The next step of the pipeline is to apply CLAHE contrast enhancements, which can significantly improve quality of the image, even when taken in poor lighting conditions. The pipeline will dynamically adjust the method parameters depending on the average light conditions of your image.

Once this is complete, the pipeline will apply a binary mask to the enhanced image, to remove unnecessary data and leave just plaques. This process does leave behind some noise, which is where contour detection is able to find just the binary shapes that are an appropriate size, shape, circularity and intensity.

Finally, each detected plaque then has a rectangular bounding box drawn around it, and the image within this box is then extracted. Batched inference is then run on these extracted bitmaps, using our CNN classifier. This classifies the plaque as "plaque" or "not_plaque", and assigns a confidence to this guess. The model confidence cutoff can be adjusted by the user. This process filters out any text, debris or other noise that might cause erroneous results.

Finally, the filtered output is displayed to the user, for final validation/corrections. The user is able to add and remove erroneous detections, as well as view the final counts and average sizes etc. In the case of overlapping plaques, the system will take the average area of the non overlapping plaques, and then divide the area of the overlapping plaques by this value. This provides an estimation to how many plaques there may be. A visual summary of the pipeline can be found below:

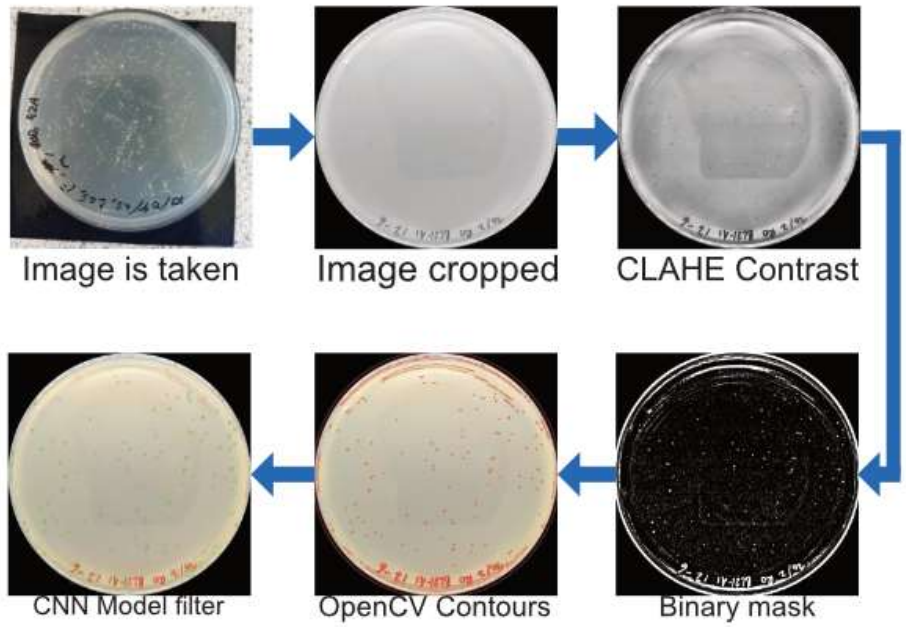


*Figure 1: CV Pipeline Summary*

## YOLO Pipeline Overview

The YOLO pipeline uses a YOLO26 segmentation model to locate plaques in your image. The model has been trained on over 150 ground truth annotated plate images, including challenging morphologies such as halo plaques, as well as blurry or imperfect images.

We used the latest YOLO26 segmentation models, and have trained both the nano and small variant of the model. This provides excellent performance at a rapid pace, suitable for end user mobile devices.

In order to reduce the memory consumption of the YOLO models, we use tiling; your image is split up into 1280x 1280px tiles, with a small overlap to ensure no plaques get lost. This allows the image to be processed at high speeds, without running out of memory on the constrained environment of mobile devices. The models are capable of running on CPU (TFLite), or GPU (ONNX), across almost any platform.

By using YOLO26, we are able to take advantage of the up to 43% speed increase compared to previous models. We are also able to use the new ProgLoss (Progressive Loss Balancing) & STAL training mechanisms, to recognise smaller clusters of plaques.

When you load an image into the YOLO pipeline, it is cropped with Hough Circle detection, and this image is then tiled. This is then fed into the inference service, which returns the detected plaque bounds. This is then displayed to the user, where again they are able to inspect and modify the model outputs.

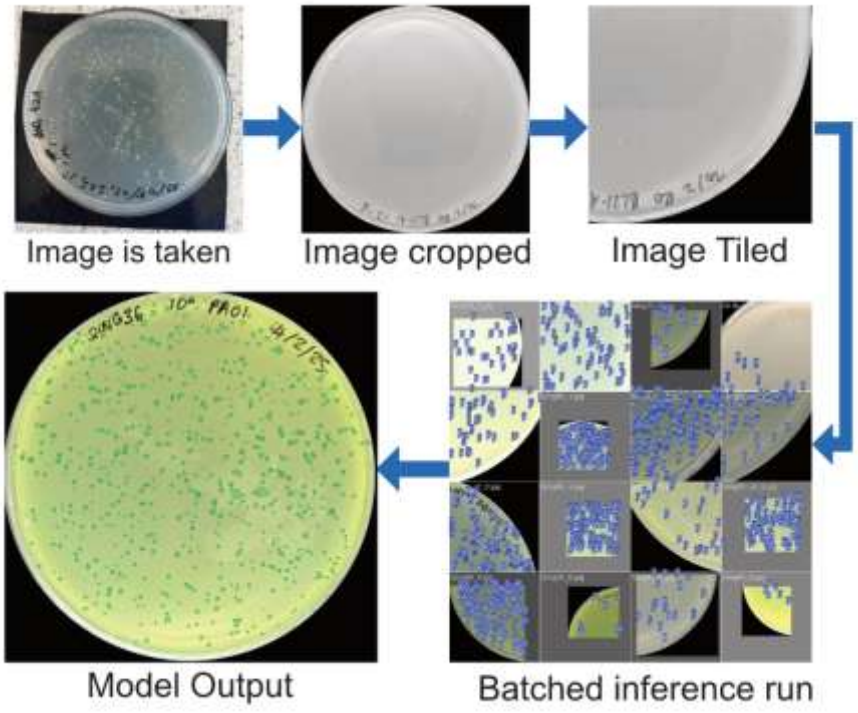


*Figure 2: YOLO Pipeline Summary*

# Best practice

## Growing plates

If possible, try to avoid your plaques becoming overgrown or overlapping, as this will reduce the accuracy of both automated and human counting.

## Taking Images

For best results, we recommend you use a modern device to take your image, and ideally using a lossless/raw format, such as .bmp or similar. We also strongly recommend you take your image lit brightly from above (avoiding glare on the dish lid), against a dark background. We have found that this helps to show plaques most clearly.

## Models

For almost every application, we would recommend using the YOLO model, as this tends to be faster, more reliable and deal with challenging image quality better. In some rare cases, you may get better results using the CV Pipeline, particularly if you are using the “droplet” feature on the mobile application.

## YOLO

### *Model Selection*

There are a number of YOLO models you can choose from; the small model will have better accuracy then the nano model, however if you have an older device, your hardware may struggle. You can also select which precision level you would like to use, float32 or float16. For more reliable results, we recommend choosing float32, a higher precision level.

### *NMS Level*

You are also able to adjust the model confidence level, and "NMS" level. NMS is Non-Maximum Suppression, which tells the model how much to supress overlapping plaques. If you notice a lot of duplicate detections, you may wish to lower this value. Similarly, if you notice that the model is missing a lot of plaques, or detecting noise, you may wish to raise/lower the model confidence accordingly.

## CV Pipeline

The CV Pipeline exposes substantially more parameters then the YOLO models, and as a result it may require further tuning in order to obtain the best results. In the app, there is a description of each parameter and its function. The app is pre-loaded with what we have tested to be the optimal settings. The different filter options are described below:

### *Min Area*

The minimum area required for a plaque to be detected. Decrease this if you notice the model missing small plaques, and increase it if you notice it is detecting noise or speckles.

### *Max Area*

The maximum area that a plaque can be. Decrease this if you notice the model detecting large artifacts, and increase it if the model is missing larger plaques.

### *Min Circularity*

How round a shape must be to be detected. If the pipeline is detecting irregular shapes, increase this. If it is missing oddly shaped or overlapping plaques, decrease this value.

### *ML Confidence*

The minimum confidence of the CNN model to accept a plaque. Increase this to filter out noise/erroneous detections, decrease this to accept more plaques. We recommend a value of ~30%

### *Advanced – Detection Sensitivity*

This sets the filter on how faint a plaque can be to be detected. Lower values increase the detection rate, but may increase false positives. If the model is missing plaques, you may wish to decrease this. If it is detecting noise, then you may wish to increase this value – however this is unlikely.

*Advanced – Plaque Separation*

Lower values separate closely-spaced plaques better, and may improve detection for smaller plaques. If your plate is overgrown, you may wish to decrease this value. Increasing this value will merge nearby plaques.

# HPC Documentation

In order to view guidance on using the HPC package, you can use `plaqseg -h` or `plaqseg --help`

```
usage: plaqseg [-h] --images PATH [PATH ...] [--model VARIANT] [--device DEVICE] [--workers N] [--crop]
               [--crop-scale FLOAT] [--crop-param1 INT] [--crop-param2 INT] [--crop-padding PX] [--tile-size PX]
               [--stride PX] [--conf FLOAT] [--nms FLOAT] [--output FILE] [--csv FILE] [--quiet]

Tiled YOLOv2.6-seg plaque segmentation — HPC & laptop ready.
Tiling is always enabled for handling high-resolution microscopy images.

options:
  -h, --help            show this help message and exit
  --images, -i PATH [PATH ...]
                        Image file(s), directory, or glob pattern(s).
  --model, -m VARIANT   Model variant to use. Choices: nano, small, nano-float32, small-float32, nano-float16, small-
                        float16. (default: small)
  --device, -d DEVICE   Compute device: 'cpu', 'cuda', or 'cuda:<N>'. TFLite variants support cpu only. (default: cpu)
  --workers, -w N       Number of parallel workers. CPU: separate processes. GPU: threads sharing one CUDA context.
                        (default: 1)

petri dish crop (optional pre-processing):
  --crop                Auto-detect and crop to the Petri dish boundary before tiling (Hough Circle Transform).
                        Recommended for plate images with significant background.
  --crop-scale FLOAT    Downscale factor for circle detection. (default: 0.25)
  --crop-param1 INT     HoughCircles param1 (Canny upper threshold). (default: 100)
  --crop-param2 INT     HoughCircles param2 (accumulator threshold). (default: 30)
  --crop-padding PX     Extra pixels beyond detected circle radius. (default: 0)

tiling (always enabled):
  --tile-size PX        Square tile size in pixels. (default: 1280)
  --stride PX           Tile stride in pixels — overlap = tile-size - stride. (default: 1024)

detection thresholds:
  --conf, -c FLOAT      Minimum detection confidence in [0, 1]. (default: 0.15)
  --nms FLOAT           NMS IoU threshold in [0, 1]. (default: 0.5)

output:
  --output, -o FILE     Path for JSON output file. If omitted, JSON is printed to stdout.
  --csv FILE            Path for CSV output (one row per detection).
  --quiet, -q           Suppress progress bar and log messages.

model variants:
  nano              YOLOv2.6n-seg PyTorch  (fastest, GPU/CPU)
  small             YOLOv2.6s-seg PyTorch  (more accurate, GPU/CPU)
  nano-float32      YOLOv2.6n-seg TFLite float32  (CPU only)
  small-float32     YOLOv2.6s-seg TFLite float32  (CPU only)
  nano-float16      YOLOv2.6n-seg TFLite float16  (CPU only)
  small-float16     YOLOv2.6s-seg TFLite float16  (CPU only)

examples:
  # Process a folder with 8 workers, JSON + CSV output
  plaqseg --images ./plates --model small --workers 8 \
          --output results.json --csv results.csv

  # GPU inference on a single image
  plaqseg --images plate.jpg --model small --device cuda --output out.json

  # Tune detection sensitivity
  plaqseg --images ./plates --model nano --conf 0.2 --nms 0.45 --output out.json
```

For full documentation, please visit the git repo: https://git.soton.ac.uk/phage-collection-project/PlaqSeg-HPC. As a general rule of thumb, we do not recommend changing the defaults unless you see issues with your results.

## Desktop Application

We have developed an efficient desktop application with a GUI for use on Apple, Windows and Linux devices. This application runs high quality ONNX & TFLITE models on your laptop or desktop device. This is generally more user friendly then the HPC Package, which may feel unfamiliar to non-technical users.

With the desktop application, you are able to select which model you'd like to use, as well as adjust the number of concurrent workers for your device. With a backend written in rust, the application is able to batch process hundreds of uncropped images much faster than the mobile application. It also supports exporting your results as either json or csv, speeding up large batch processing.

If you have a more capable PC, we would recommend increasing the number of workers to speed up processing. If your device has a GPU or is CUDA enabled, then the software should detect this and run accordingly.

For further documentation, please visit the project repository:
https://github.com/Carbon16/PlaqSegDesktop

## Benchmarks

## Code

Source code is available in git repos. PRs/Issue reports are welcome!
HPC Repo: https://git.soton.ac.uk/phage-collection-project/PlaqSeg-HPC

Mobile Repo: https://github.com/Carbon16/PlaqSeg-App

Desktop Repo: https://github.com/Carbon16/PlaqSegDesktop

## Support

For technical support with these application, you can contact us at xxxxxxxxxxxx@soton.ac.uk, where we will be happy to help you setup, run and tune your images. If you would like to contribute images to our dataset for training, these are most welcome and can again be sent to us at the above email address.