# Iterative Development

## Pipeline Evaluations

One data pre-processing technique that was used during the development of this project is a Gaussian-blur. This is a function that image data is passed through as a smoothing technique, reducing noise in the image and adding a slight blur to remove fine detail. The standard deviation of the Gaussian function controls the amount of blurring, with a larger standard deviation blurring the image more and a smaller standard deviation blurring less.

Below is an example of an original, unaltered dataset image, compared to the same image after it has been blurred.

Original:

Gaussian Blur:

Another image pre-processing technique is Pixel Brightness Transformations (PBT). Transformations of the brightness of pixels are dependent on the properties of the pixels themselves. The output and resulting values of the pixels are reliant on the initial input values. There are two different types of pixel brightness transformation, one being position-dependent brightness correction and the other being grayscale transformation.

A grayscale transformation for road sign detection is not as suitable for this project as it would be for other applications such as a handwritten digit recognition system, where the colour of the object will most likely have no effect on offering additional data that is helpful when classifying. This is because road signs make strong, consistent use of colour throughout road signs, such as speed limits having a red circle around a white background with a black number, or many road crossings having a blue background with a white triangle containing a person crossing a zebra crossing in black. This consistent and contrasting use of colour is useful data that can be used when training and later classifying images, and therefore to apply a grayscale transformation would remove this useful data.

**Geometric Transformations**

Geometric Transformation works differently to others, it uses the positions of pixels and modifies them, but the colours remain the same.

There are two steps to geometric transformations, the first being spatial transformation of the physical rearrangement of pixels in the image. Secondly use grey level interpolation, this aligns grey levels to the transformed image.

**The different transformation methods**

1. Scaling: Scaling is just resizing of the image

$$\begin{aligned} x' &= x \cdot S_x \\ y' &= y \cdot S_y \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

2. Translation: Translation is the shifting of object's location

$$\begin{aligned} x' &= x + \Delta x \\ y' &= y + \Delta y \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

3. Rotation: Just rotating an object with theta degrees

$$\begin{aligned} x' &= x \cdot \cos\theta - y \cdot \sin\theta \\ y' &= x \cdot \sin\theta + y \cdot \cos\theta \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

4. Shearing: Shifting the pixels horizontally

$$\begin{aligned} x' &= x + y \cdot B_x \\ y' &= x \cdot B_y + y \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & B_x \\ B_y & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

5. Affine Transformation: Instead of defining the scale factors, it is common to merge into one matrix.

$$\begin{aligned} x' &= a_1 \cdot x + a_2 \cdot y + a_3 \\ y' &= b_1 \cdot x + b_2 \cdot y + b_3 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_3 \\ b_3 \end{bmatrix}$$

6. Perspective Transformation: Change the perspective of a given image. Here the points need to be provided on the image from which you want to gather information by changing the perspective.

### Image Segmentation and filtering

Image Segmentation is the procedure of distinguishing the background from the main object. This method allows enhancement and modification of various image properties. Significant information such as edges, curves and dots can also be obtained from these images. Kernel is a form of filter which can be operated on every pixel and its adjacent neighbors. Following are few of the known filtering techniques:

Low Pass Filtering (Smoothing): Smoothening of images is commonly done using a low pass filter. This is achieved by reducing the difference between pixel values by computing the mean of adjacent pixels.

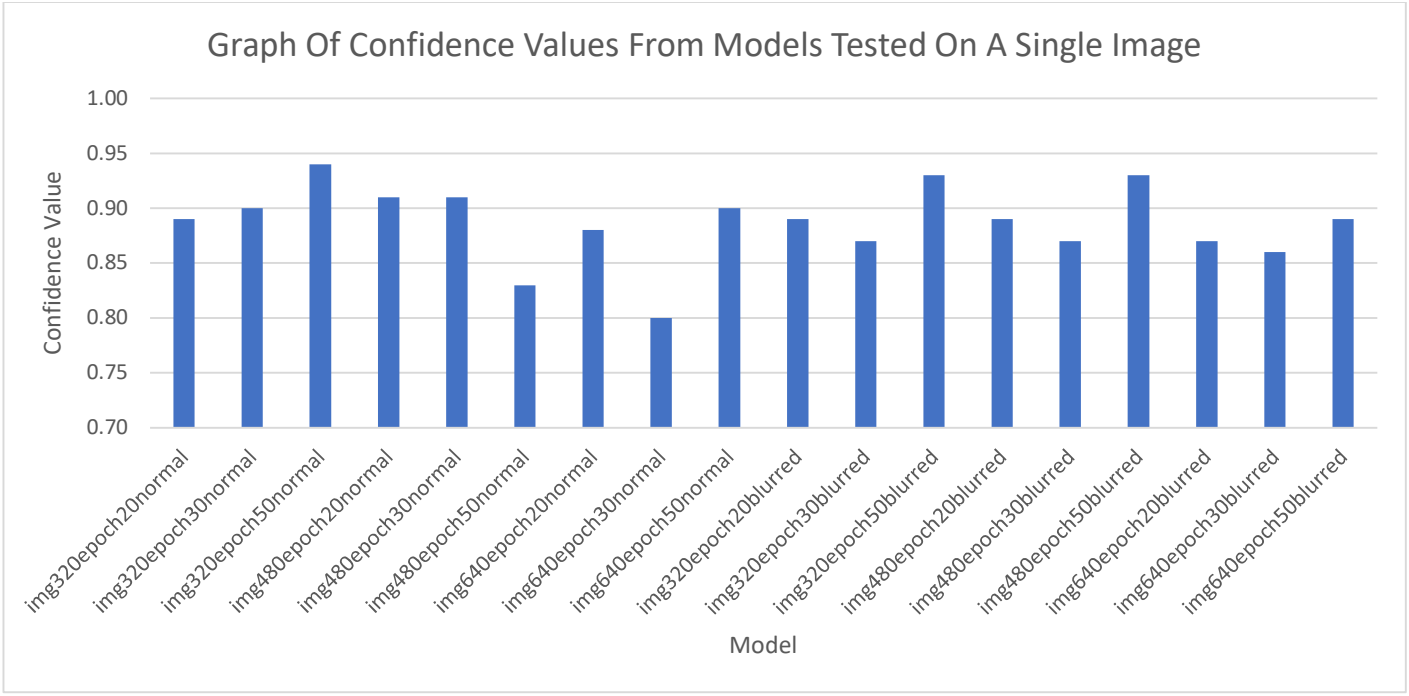
High pass filters (Edge detection, sharpening): Sharp images can be obtained using a high-pass filter. This filter focuses on the fine details of the image. The underlying principle is the same as that of a low-pass filter, however, a different convolutional kernel is used.

Directional filtering: The primary derivatives of an image can be calculated using an edge detector such as directional filter. These first derivatives are noticeable when a major change happens between nearby pixels. This filter can be leveraged for every direction within a provided space.

Laplacian filtering: The second derivatives of an image can be computed using an edge detector such as Laplacian filter. It provides information on whether the change in nearby picture element values is from an edge or constant progression. The kernels of this filter comprises of a cross pattern of negative values centered in the array. The corners have either 0 or non-negative values. The center value may be negative or positive.

## Model Evaluations

Models were tested on a single data image seen below. All models successfully detected a single traffic light with a suitable bounding box, the confidence values of which were recorded and displayed in a column graph (seen below).




As seen in the graph above, all models trained from a Gaussian Blur dataset remained above 85%, whereas two models (img480epoch30normal and img640epoch30normal) dropped below this threshold. Mean average detection rate for normal models was 88% whereas the blurred models were slightly higher at 89%.

Detection rate is not the only useful factor when comparing models though, as a good, well-performing model should also be capable of creating a suitable bounding box around the object. Some models are also more capable at this than others, with gaussian blur and image resolution demonstrating a significant impact on this.

Team Mace Windu



Above is an image detected using model with a training image size of 320 pixels at 20 epochs without a gaussian blur, compared to an image detected using a model with a training image size of 640 pixels at 50 epochs without a gaussian blur. While both managed to detect the object adequately, the latter model detected the object with a much more suitable bounding box, with the light emitting part of the traffic light more centralized and the top and bottom of the traffic light not cut off.

Team Mace Windu


Above is an image detected by a model similar to the previous latter model, using an image size of 640 with 50 epochs, but this time also using a Gaussian Blur. The bounding box is quite alike to the same model without the Gaussian Blur in that it keeps the light part more centralized and does not cut off the top or bottom, however, is also much wider on the left, “detecting” a part of the traffic light that is actually a tree/building in the background.


Above is another example of the same image tested on a different model. This model was created “from scratch” rather than taking a pre-existing model provided by yolov5 and fine tuning it, like all the others have been. This is because a “from scratch” model had severe issues with both low confidence (53% compared to >80% of all other models), false positives (see crossing button on right side of image) as well as very poor bounding boxes (cutting off parts of the traffic light while bounding nearby parts of the background).

Team Mace Windu

## Parameter Justifications

A batch size of 8 was used consistently as it is recommended to use the largest value of n for $2^n$ for the fastest possible training. This should give no penalty to model performance, and instead requires more GPU memory. During training for these models, the highest batch size possible for an image size of 640 was 8 before memory related errors occurred, this value was used consistently thereafter for all models including the smaller image size models where a higher batch size should theoretically be possible.

Image size was another parameter that was configured with different models, ranging from 320 pixels to 640 pixels. The 320-pixel models were both faster to train and can process images/frames at a faster speed, resulting in a faster fps (frames per second), increasing real time detection performance. However, these models also perform weaker in general, often making bounding boxes of images too big, too small or off center, not completely capturing the true outline of an object. This makes the higher resolution and slower, yet still "real time" enough models more compelling.

Number of epochs was also varied, with less epochs often being less confident, and higher epochs potentially performing poorer due to overfitting. Overfitting can be seen if a model is tested on a never-seen-before image and performs very poorly, however performs particularly well on an image that was present in the training dataset. Overfitting is often caused by training a model for too many epochs, without enough unique data to accommodate for the larger number of generations the model will learn from.

The model was trained on both normal data as well as data after undergoing a Gaussian Blur. This was to reduce noise and remove finer detail in images that the model may be focusing on too much that may not be present in real world data when the model is tested/implemented.

A pre-existing yolov5-provided weights file was used and fine-tuned rather than creating a weights file from scratch, as shown in the model evaluation this was much more effective compared to the poor performing from-scratch models, with differences of over 30-40% and false positives or negatives becoming much more common.

Overall, a batch size of 8 with an image size of 480 pixels are reasonable parameters for a good model, as it is trainable at a reasonable speed even on weaker systems, which is important when given the application of being implemented in a car as real time detection is important, and as such a high fps on a system capable of working from little power within a small space at an affordable price is important. An epoch count between 30-50 should be used, as 20 seems inaccurate with bounding boxes with somewhat lower confidence values, however 50 and above is where issues with overfitting data begin. A dataset that has undergone a Gaussian Blur should be used as this helps reduce noise in images and remove finer details. From testing of various models, this suggests that images are generally detected with a higher confidence value, with two models from the normal set of models having confidence scores of below 85% whereas all models tested on the same image with the same parameters but with a Gaussian Blur have no such issues.

## Reproducible Code

The code below is the code used from a bash script to train several models with various parameters. Models were trained at 3 different image resolutions at 3 different epoch values (number of generations), with half the models trained on the original dataset and the other half on the same dataset after undergoing a Gaussian Blur transformation.

```
python3 train.py --img 640 --batch 8 --epochs 20 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 640 --batch 8 --epochs 30 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 640 --batch 8 --epochs 50 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 640 --batch 8 --epochs 20 --data blurred.yaml --weights yolov5l.pt
python3 train.py --img 640 --batch 8 --epochs 30 --data blurred.yaml --weights yolov5l.pt
python3 train.py --img 640 --batch 8 --epochs 50 --data blurred.yaml --weights yolov5l.pt

python3 train.py --img 480 --batch 8 --epochs 20 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 480 --batch 8 --epochs 30 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 480 --batch 8 --epochs 50 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 480 --batch 8 --epochs 50 --data blurred.yaml --weights yolov5l.pt
python3 train.py --img 480 --batch 8 --epochs 20 --data blurred.yaml --weights yolov5l.pt
python3 train.py --img 480 --batch 8 --epochs 30 --data blurred.yaml --weights yolov5l.pt

python3 train.py --img 320 --batch 8 --epochs 20 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 320 --batch 8 --epochs 30 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 320 --batch 8 --epochs 50 --data normal.yaml --weights yolov5l.pt
python3 train.py --img 320 --batch 8 --epochs 50 --data blurred.yaml --weights yolov5l.pt
python3 train.py --img 320 --batch 8 --epochs 20 --data blurred.yaml --weights yolov5l.pt
python3 train.py --img 320 --batch 8 --epochs 30 --data blurred.yaml --weights yolov5l.pt
```