# Probe Detection Using Deep Learning

Michael Morris
Email: m.morris95@protonmail.com
Phone: +41 76 226 94 74

## 1 Summary

The goal is to create a deep learning system to correctly identify and locate a bounding box around an ultrasonic thickness measurement probe attached to a drone. We are provided with 308 labelled images for training, validation, and testing. Each label contains an x-coordinate, y-coordinate, width, and height of the bounding box. There are no images that do not contain the probe.

The task is open-ended, and it would be possible to invest considerable time and resources to find the best solution. However, I chose to limit myself to two half-days of work, with a simple run of hyperparameter tuning in between.

## 2 Approach

There are several approaches that could work for the system. A traditional image processing approach based on thresholding is unlikely to work, as the images often contain other artefacts and are greyscale. If, for example, the images were in colour and the probe was always the same colour, a deep learning approach may be unnecessary. Given the limited data, a pre-trained system is likely to work best.

The most common approaches are to fine-tune a pre-existing model like YOLO or SSD, or to use an existing architecture (such as VGG, MobileNet, etc.) as a feature extractor with a custom prediction step for the output. In this work, I used VGG16 and MobileNetV2 as feature extractors for transfer learning, with an additional prediction head.

**Justification** Ideally, both approaches would be explored in detail. YOLO is the existing state-of-the-art, so it is likely to perform well. However, it is designed for multi-object detection in complex images, whereas this dataset contains simple, black-and-white images with a single object. Furthermore, YOLO benefits from larger datasets and requires extensive computational resources.

Pre-trained CNNs are well suited to this task. By using transfer learning, we can leverage models trained on large datasets, which significantly reduces the need for training data, computational resources, and development time. I have prior experience with transfer learning, so I am confident that I can quickly develop a reliable system without the overhead of developing a more complex model like YOLO.

I did not consider building an architecture from scratch, as pre-trained CNNs already offer a proven foundation for image modelling tasks. Given the dataset and the project timeline, I believe that transfer learning offers the best balance of efficiency and effectiveness for this problem. However, with more time, I would compare the system I developed with YOLO and SSD.

## 3 Methods

**Data Processing** The dataset consists of 308 black-and-white images, each of size 600x400 pixels. To adapt them for a pre-trained CNN, I first resized all images to 224x224 pixels and then duplicated the channels to convert them to RGB images. Finally, the pixel values were normalised to the range $\in (0, 1)$.

Each target contains four values representing bounding box coordinates, which were normalised as follows: the $x$ position and width were scaled by $x/(640/224)$, and the $y$ position and height by $y/(400/224)$.

The inputs were converted to `tf.float32`, and the dataset was shuffled before being split into training (70%), validation (15%), and test (15%) sets. To ensure the integrity of the data, I visualised the results by plotting the images with their corresponding bounding boxes.

I did not use any data augmentation techniques due to time constraints. However, given more time, I would use rotation, translation, flipping, cropping, brightness/contrast adjustment as well as adding noise and Gaussian blur, this augmentation would create a larger training set. The development of this would be time-consuming as the bounding boxes would have to be adjusted correspondingly, which is not the case for classification tasks.

**Model Design** I decided to try a pre-trained CNN with its output layer removed. In place of the output layer, I added a small feed-forward neural network with ReLU activations. The output of the feed-forward network contains 4 units for the positions of the bounding box. This is essentially a regression task, where the outputs are the normalised positions of the bounding box, so no activation function is required. However, I tried both a sigmoid activation function and no activation function as part of my hyperparameter tuning. As this is a regression problem, the network is trained using mean squared error (MSE).

I decided to run a grid search over hyperparameters based on validation MSE. I trained each model for a maximum of 500 epochs using an Adam optimiser, and I used early stopping based on validation loss to avoid overfitting. I iterated over the hyperparameters described in Table 1.

| **Hyperparameter** | **Values Tested** |
|---|---|
| Base Model Type | `mobilenetv2`, `vgg16` |
| Dense Units List | $[128, 64, 32]$, $[32, 16]$ |
| Dropout Percent | 0.2, 0 |
| Output Activation | `linear`, `sigmoid` |
| Learning Rate | 0.001, 0.0001 |
| Batch Size | 32, 16 |

Table 1: Hyperparameter values tested.

**Adjustments for Classification Model** I did not have time to implement a classification model, as there is no data where no probe is present. This could be generated by taking images with a probe and extracting an area outside of the bounding box.

To modify the network for classification, I would add an additional unit to the output which would use a sigmoid activation function. This unit would be trained with a binary cross-entropy loss. The loss function would be as follows:

```
Input: outputs (model predictions),
bounding_box_positions (ground truth positions),
probe (binary indicator: 1 if probe is in the photo, 0 otherwise),
$\lambda$ (weighting factor for the binary cross-entropy loss)
Output: loss (calculated loss value)
1 if probe is True then
2     loss = MSE(outputs[..., :-1], bounding_box_positions)
3         + $\lambda$ * BCE(outputs[..., -1:], probe)
4 end
5 else
6     loss = $\lambda$ * BCE(outputs[..., -1:], probe)
7 end
```

.

## 4 Results

We compare the best `vgg16` and `mobilenetv2` models as chosen by the hyperparameter tuning. These are shown in Table 2.

| **Hyperparameter** | **mobilenetv2** | **vgg16** |
|---|---|---|
| **dense_units_list** | [128, 64, 32] | [128, 64, 32] |
| **dropout_percent** | 0 | 0.2 |
| **output_activation** | sigmoid | sigmoid |
| **learning_rate** | 0.0001 | 0.0001 |
| **batch_size** | 32 | 16 |
| **ID** | 14 | 39 |
| **val_loss** | 0.005426 | 0.0067 |

Table 2: Hyperparameters and Validation Loss for Different Models

We compare the models on the test set using Intersection over Union (IoU) and mean average precision (MAP). For both metrics, a higher score is better; for MAP, we use a threshold of 0.5. The results are enumerated in Table 3.

| **Model** | **IOU** | **MAP** |
|---|---|---|
| MobileNetV2 | 0.43 | 0.50 |
| VGG16 | 0.49 | 0.54 |

Table 3: Test set performance comparison between MobileNetV2 and VGG16 models.

Results from the test set are shown in Figures 1 and 2:

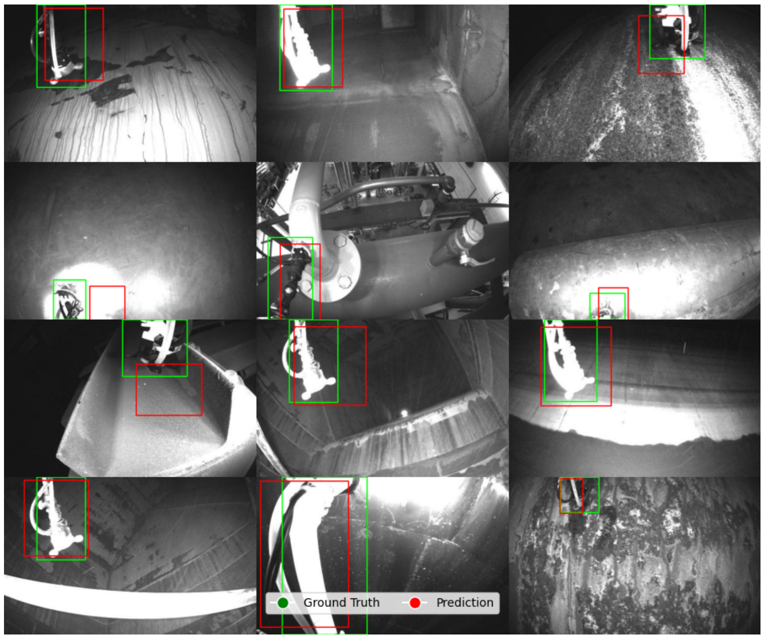

Figure 1: VGG16 predictions (images chosen randomly from test set)

## 5 Discussion

**Performance Analysis** Both models performed more poorly than expected, with IoU scores under 0.5. This can be attributed to a number of factors. Firstly, the hyperparameter tuning was quite basic. It would have been better to use an adaptive and efficient system such as Bayesian optimisation to find the optimal solution and avoid wasting time testing solutions that do not work well. It would also have been better to try more than just two architectures. However, the biggest challenge was the lack of data; this could be remedied by using data augmentation. Due to time constraints, however, it was not feasible to implement and validate data augmentation at this time.

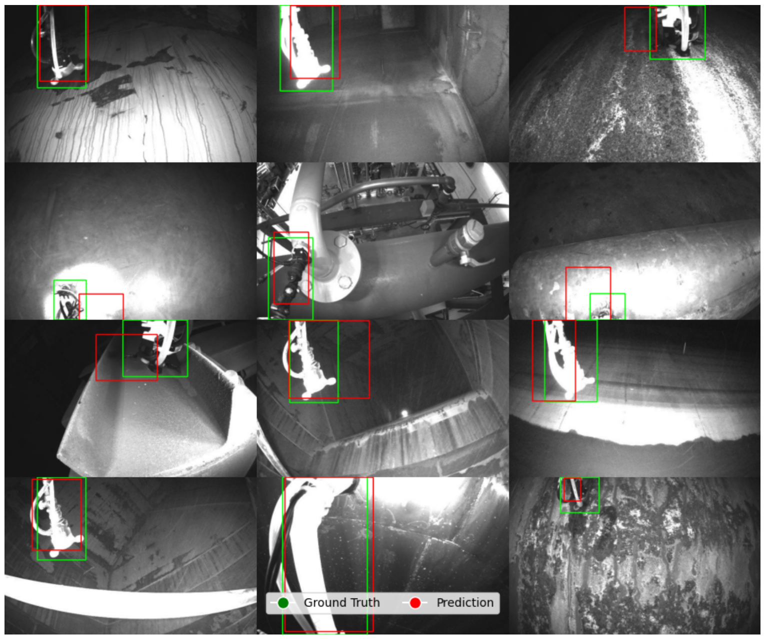

Figure 2: MobileNetV2 predictions (images chosen randomly from test set)

The VGG16 model performed better both in terms of IoU and MAP. This is unsurprising, given the complexity of VGG16 compared with MobileNetV2. Mobilenetv2 performed the best in terms of validation score, this was surprising given that the next 4 were VGG16 models, suggesting that this was an outlier. Using a larger validation set would give more robust tuning results. However, given the limited availability of data, I made the choice to use as large a training set as possible.

From Figures 1 and 2, we can see that the models tend to perform well on the same images. Where the contrast between the probe and the background is high, performance improves. When the contrast is lower or the probe is less well defined, performance deteriorates. This is expected behaviour and could be improved by further training on augmented data where the contrast is reduced, noise is added, or a Gaussian blur is applied.

**Run Time** Running on a CPU (Intel 14700K), MobileNetV2 was able to run at approximately 18 FPS, while VGG16 runs at approximately 13 FPS. Both of these would be able to run much faster on a GPU; however, performance could also be accelerated by using lower precision parameters, e.g., `float16`. Using a system like YOLO, which is optimised for inference, may run faster, but at the cost of training complexity.

**Improvements** The most obvious improvement is to use data augmentation to increase the size of the dataset. This could greatly improve the robustness and accuracy of the system by providing much more diverse training examples. In doing this, it would also be possible to create a dataset of images that do not contain a probe, and thus train the model for classification as well. To prevent class imbalance, it would be important to ensure that roughly half of the images contain a probe and half do not.

Depending on the accuracy of the model after implementing data augmentation, it may be necessary to include an uncertainty estimation method. This could be done relatively simply using a Bayesian neural network approach and simply reusing the base model as a feature extractor.

Finally, it would be good to try more diverse approaches. YOLO, SSD, and more varied transfer learning approaches may provide better results. It is important to try things in a considered way to avoid spending unnecessary time developing and manually tuning models. Efficient hyperparameter tuning schemes like Bayesian optimisation provide an avenue to do this.