Defect Detection Based on Synthetic Dataset

Author: Yaseen Moolla

Date: 9 February 2022

1 Introduction

The goal of this exercise is to use machine learning (ML) to perform segmentation on a dataset of synthetically generated gear and spring images. An ensemble of neural networks was designed. In the first step, a classification neural network model was used to determine if the image was of a gear or spring. In the second step, two separate segmentation models were use for gear and spring images respectively, in order to detect regions of defects in the input images. To design the classification model, transfer learning was applied to the VGG16 convolutional neural network (CNN). For segmentation, transfer learning was applied on the DeepLabv3 neural network, which is an existing deep neural network model which has been designed for segmentation. The results are discussed, and suggestions for improvements are made.

2 Defect Detection in Synthetic Images

Two separate models were trained for springs and gears, because the shapes of the source objects and the manner in which light reflects off them creates very different images. The DeepLabv3 deep neural network with a ResNet101 backbone is a pretrained model which was chosen as a starting point. It is a popular ML model that is used to identify and segment regions in images. It uses atrous convolution for multiscale segmentation, and is effective with input images of any size. [1]

Transfer learning was applied to the last layer, in order to train the model to recognize the defects in gear and spring images. The DeepLabv3FineTune package [2] was used to apply the transfer learning. Transfer learning allows for fast and effective training when there is a limited dataset.

2.1 Data Cleaning

Using the code in the folder `DatasetCleaning` it was found that one image was broken (orig_4_25.png) and one image was truncated (orig_3_20.png) in the spring dataset. These images were removed from the dataset before further processing continued.

2.2 Defect Detection for Gears

An 8:2:1 training:validation:test split was used. This was achieved by leaving 1 folder aside for testing and applying a random 80:20 split on the rest of the 10 synthetic image folders. 5 epochs were used for training, with a total training time of 13 hours on a CPU. (No Cuda-enabled GPU was available for faster and more numerous model generation.) The loss metric used was the mean squared error (MSE). The DeepLabv3FineTune package [2] was used.

Figure 1 shows the loss, F1 score and Area Under the ROC (AUROC) over the 5 epochs. Table 1 shows the final values for training and validation, as well as testing. The loss function reduces as expected and the ROC increases and levels off near a maximum. The accuracy with the test set is 99.72%. However, these metrics are misleading. The F1 score, precision and recall remain low, as shown in Table 2. While MSE is the default, it may not be the ideal loss function for this scenario. There is an imbalance in the area covered by defects and the unaffected area, which creates a bias towards the unaffected areas of the image.

For segmentation of the defect regions, a histogram is generated for the output image when a gear image is passed as input into the segmentation model, as shown in Figure 2. The ideal binarization threshold is close to the upper bound. Through a visual analysis of several images, using the code in the Jupyter Notebook `AnalysisGears.ipynb`, a threshold value was chosen. This threshold was then applied to all images in the test set to generate masks for each image, using the code in `gearsPredictSegmentationTestSet.py`. These images are stored in the folder `GearsTestSetOutput`. A sample is shown in Figure 3. Precision, recall and F1 score were then calculated in comparison to the ground truth images, using the code in `gearsTestMetrics.py`.

Table 2 shows the precision and recall for the test set. A low precision indicates a large number of false positives. A visual analysis shows that some regions have false segmentation areas, such as vertical lines in Figure 3. This may be due to overtraining, using the incorrect loss function or imperfections in finding the ideal threshold. A low recall indicates a high number of false negatives. This may be due to two factors. First, while regions with dents are detected, the segmentation doesn't not always cover the full region of the dent. Second, scratches are usually not detected.

To address these issues and to improve the precision and recall, a different metric for the loss function should be used. Potential alternatives include either Cross Entropy or a custom function based on the F1 score that focuses specifically on measuring the detected areas rather than a measurement of the pixels of the entire image.

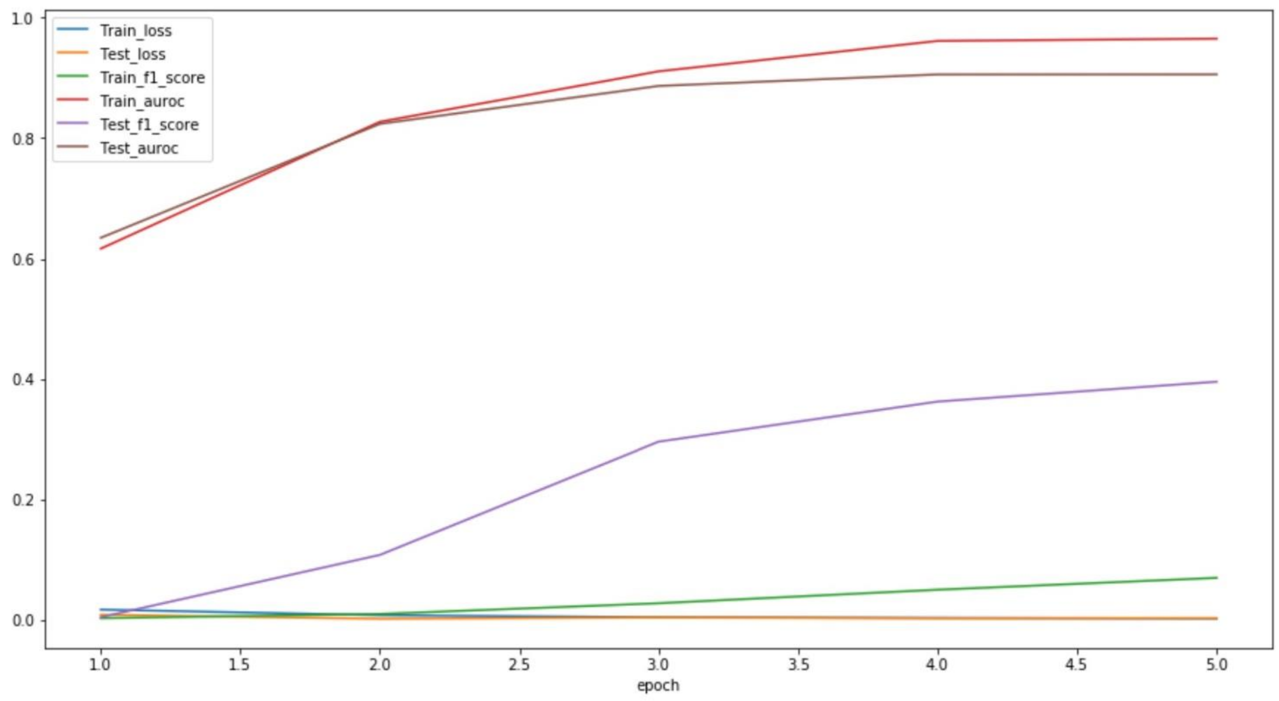

Figure 1: Performance for training defect detection segmentation on gear images over 5 epochs

Table 1: Performance Measurements for Gear Detection

	Loss	**F1 score**	**AUROC**
Train	0.00226	0.070	0.965
Validation	0.00299	0.395	0.906
Test		0.031	0.564

Table 2: Performance measurements for synthetic gear images at pixel level, where 0 is black pixels and 1 is white pixels

	Precision	Recall	F1 Score
0	1.00	1.00	1.00
1	0.02	0.13	0.03

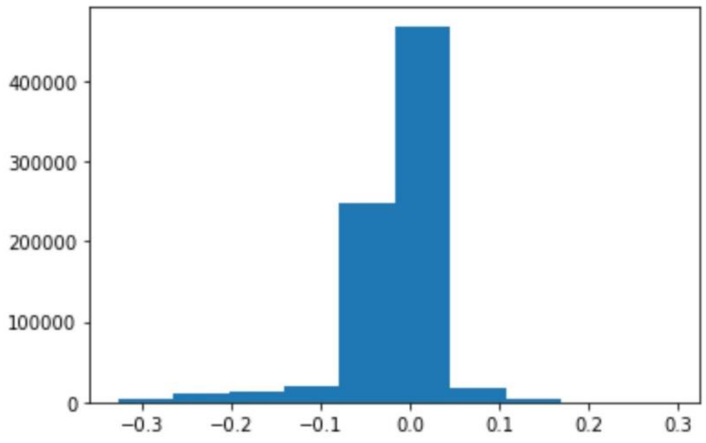

Figure 2: The histogram for the output image when a gear image is passed as input into the segmentation model. The ideal binarization threshold is close to the upper bound.

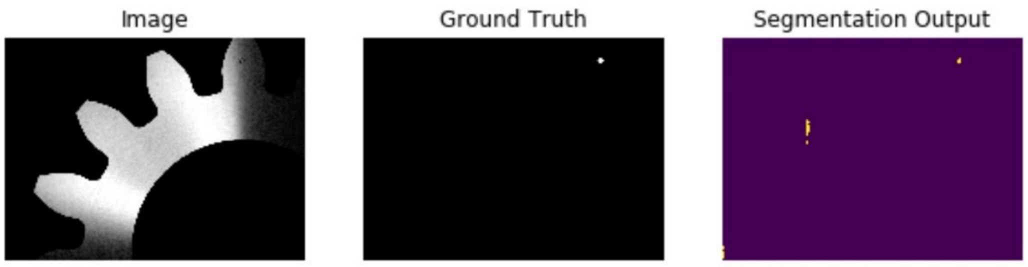

Figure 3: From left to right, sample of an image, its ground truth mask and its predicted mask

2.3 Assessment of Real Gear Images

To process the real images, images first need to be resized to match the size of synthetic images for effective application of the model, i.e. dimensions of 1024 x 768. As with the test set, an analysis of the images was performed to find the ideal threshold using a Jupyter Notebook, `AnalysisGearsReal.ipynb`, and masks were then generated using the code `gearsPredictSegmentationReal.py`. The output is in the folder `Processed Real Gear Images`.

Figure 4 show a sample of defect detection on real images. The location of the dent in the gear has been detected, although the full area of the dent was not detected. False regions have also been detected. Scratches were not detected. Detection was less accurate with some images. Methods to improve detection have been discussed in the previous section.

[TODO: add in folder 3 and 4.]

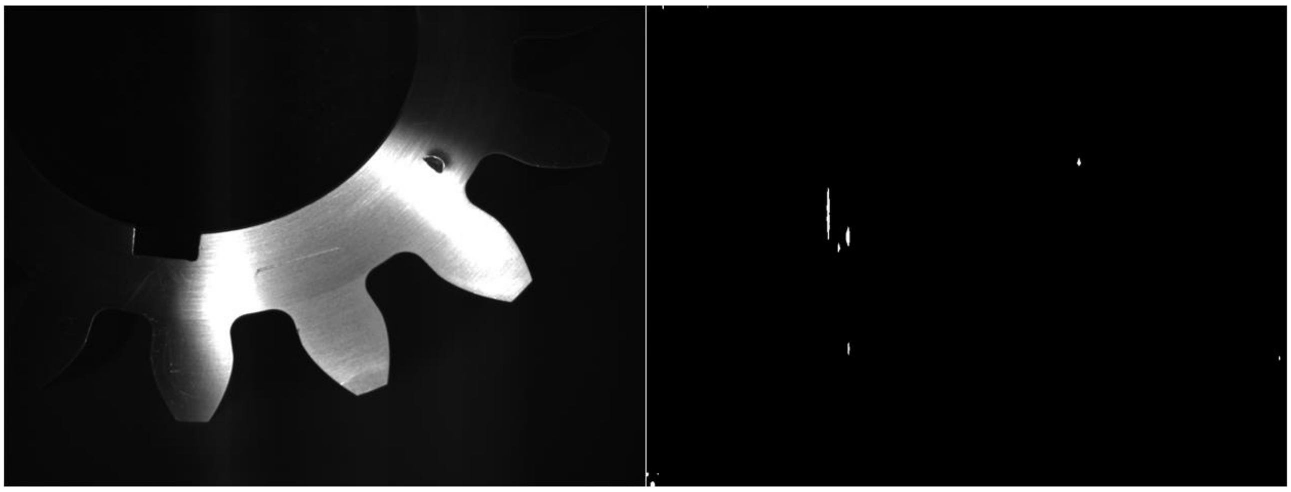

Figure 4: **A sample of defect detection on real images. The real image is on the left and the detected defects are on the right.**

2.4 Generation of Synthetic Gear Images

Further synthetic gear images were generated using Microsoft Paint 3D, which is a default application in Windows 10. While this application can be used to create 2D renderings from 3D objects, its functionality is limited. These limitations include a lack of ability to generate a variety of surfaces; a limited range of adjustments for the the angle, intensity and wavelength of light; and an inability to simulate reflections off metallic surfaces.

To process the generated images, images first need to be resized to match the size of synthetic images for effective application of the model, i.e. dimensions of 1024 x 768. As with the test set, an analysis of the images was performed to find the ideal threshold using a Jupyter Notebook, `AnalysisGearsGenerated.ipynb`, and masks were then generated using the code `gearsPredictSegmentationGenerated.py`. The generated images and the output of the segmentation is in the folder `GeneratedSyntheticGears`.

Figure 5 and Figure 6 show a samples of defect detection on selected generated images. The location of the four out of fives dents were detected. The scratch on the generated gear was not detected. Additionally, several false defect regions were detected. Detection was less accurate with some images. Differences in lighting did not seem to affect performance. A wider range of lighting conditions should be tested in the future. Methods to improve detection have been discussed in the previous section.

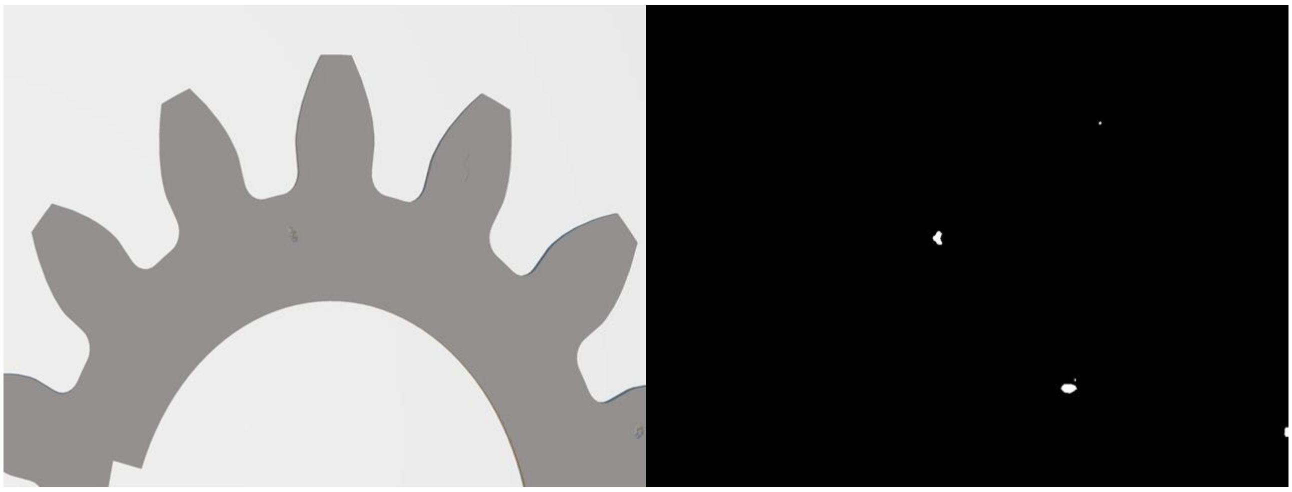

Figure 5: **A sample generated gear and its detected defect regions.**

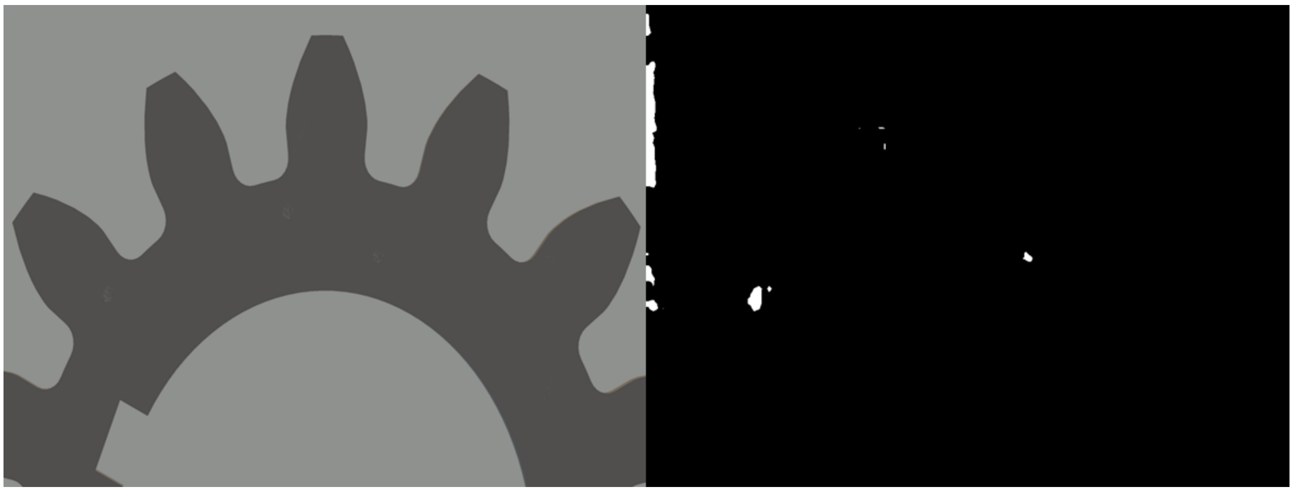

Figure 6: A sample generated gear with lower lighting conditions and its detected defect regions.

2.5 Defect Detection for Springs

Training, validation and histogram analysis were performed for the Spring dataset in the same manner as the Gear dataset. The results of training and validation are shown in Figure 7 and Table 3. The F1 score and AUROC were found to be very low. Histogram analysis, binary thresholding and mask generation were performed on selected images using the file `AnalysisSprings.ipynb`. Performance was found to be inconsistent. Since a spring comprises of different parts of varying shapes and sizes, the ideal binarization threshold for segmentation varies greatly between different image types. This shown in Figure 8, Figure 9 and Figure 10, where each represents a different part of a spring. In Figure 8, a dent is detected, but several false defects are also detected. In Figure 9 and Figure 10, many false regions are detected but no true defect regions were detected. As a possible method for better performance, sub-classification of spring images and training a different model for each sub-class may result in better performance. Additionally, an automated classifier for each sub-class would need to be created. This is discussed further in the classification section.

Due to time constraints and the low performance of the defect detection model for springs, analysis with real and generated images was omitted. Focus was placed on creating a classifier instead, as discussed in the next section.

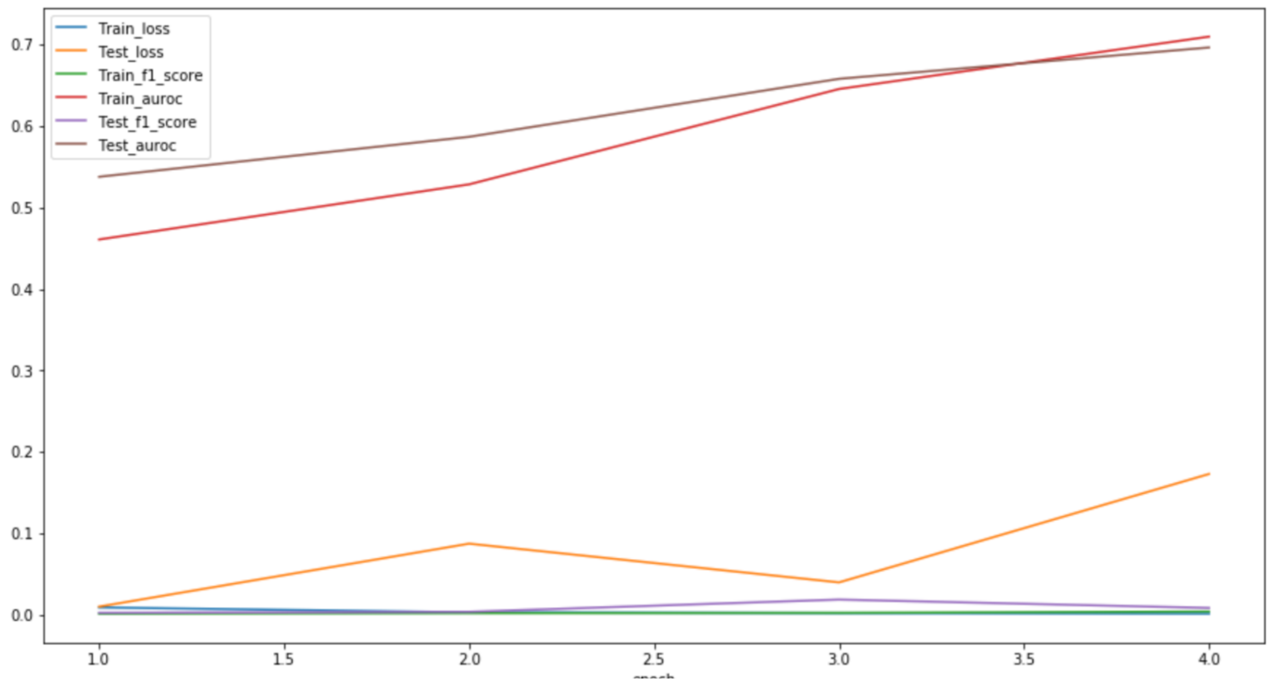

Figure 7: Performance for training defect detection segmentation on spring images over 4 epochs

Table 3: Performance Measurements for Spring Defect Detection

	Loss	F1 score	AUROC
Train	0.00139	0.00397	0.710
Validation	0.173	0.00838	0.696

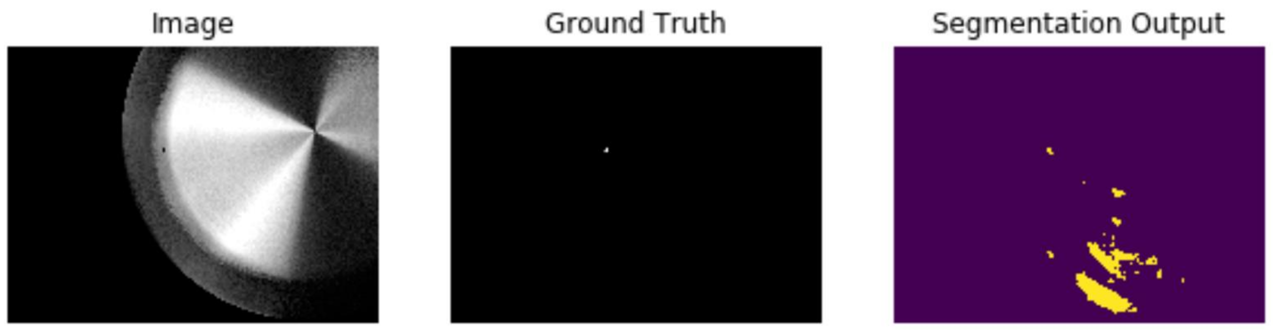

Figure 8: Defect detection on the flat end of a spring

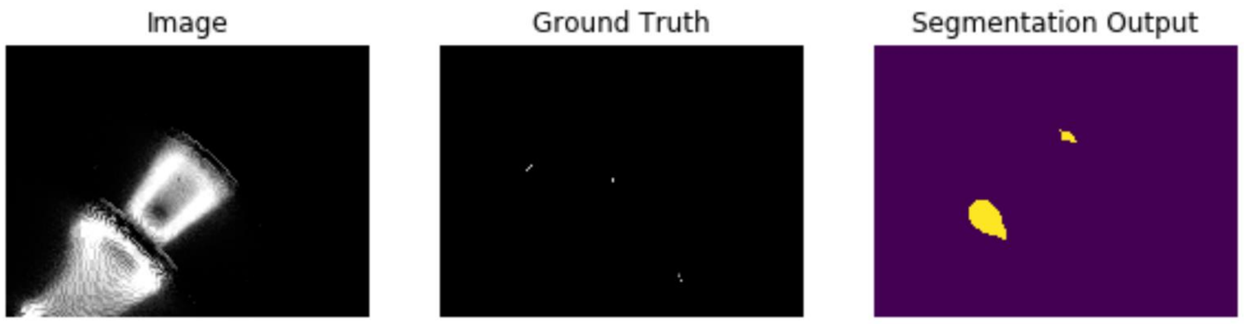

Figure 9: Defect detection from the side view of the flat end of a spring

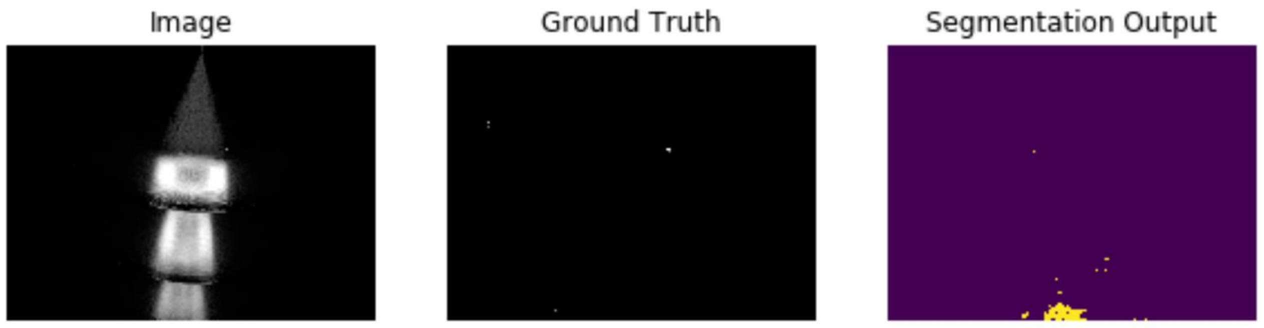

Figure 10: Defect detection from the side view of the pointed end of a spring

3 Classification

For classification between gear and spring images, transfer learning was applied on the VGG-16 (Very Deep Convolutional Networks for Large-Scale Image Recognition) convolutional neural network. VGG is a popular image classification model, which is commonly used as a baseline. It is small and fast, compared to newer image classification models.

Train: validate: test was applied on an 80:10:10 ratio over 1 epoch, using the code in `CreateGearSpringClassifier.py`. The resulting classifier is found in folder `ClassificationDataSet`, along with the training, validation and testing datasets. Analysis of the test set was performed using the code in `ClassifyGearSpringBatch.py`. The test produced 100% classification accuracy. The results of classification are shown in Table 4. This model can now be used to create a pipeline to classify an image as either gear or spring and call the relevant segmentation model. The current pipeline is shown in Figure 11. A suggested future pipeline, which may improve the defect detection in spring images through an ensemble of classifiers, is shown in Figure 12.

The application to run the classification and defect detection, as described in Figure 11 is found in `GearSpringPrediction.py.`

Table 4: Performance Measurements for classification of gear and spring images

	Loss	**Accuracy**
Train	7.63	0.711
Validation	1.63e-11	1.00
Test		1.00

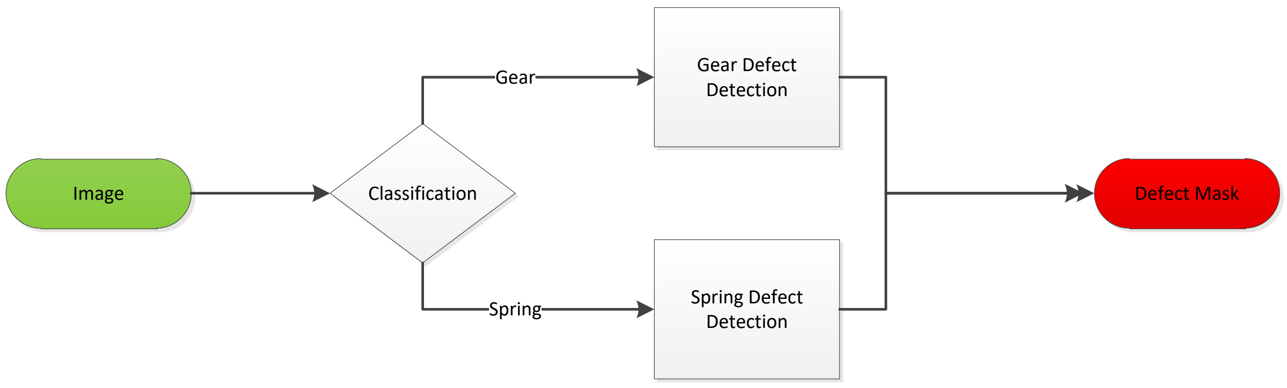

Figure 11: Current dataflow for classification and defect detection

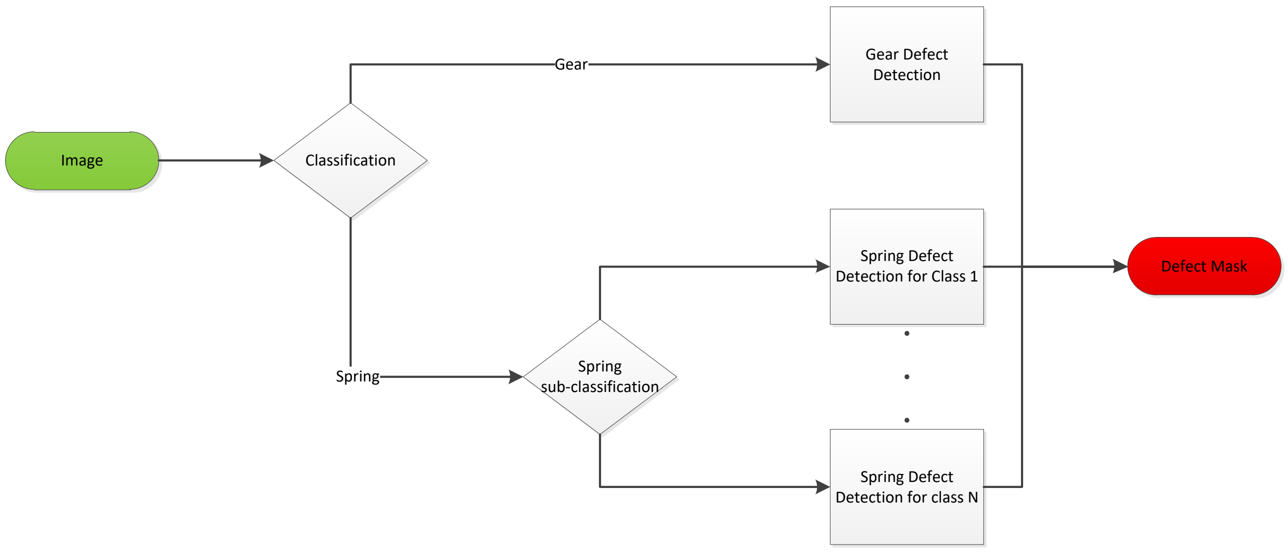
Image
Classification
Gear
Gear Defect Detection
Spring
Spring sub-classification
Spring Defect Detection for Class 1
Spring Defect Detection for class N
Defect Mask

Figure 12: Proposed future design for classification and defect detection

4 Suggestions for future improvements

As discussed in previous sections, a different loss function, such as cross entropy or a custom-made loss function based on the F1 score, may improve defect detection.

Practically, it may be effective to save the model after each epoch. This will allow for testing of earlier model weights in the event of over training.

For the generation of synthetic images, a generative adversarial network (GAN) may be used.

Training should be performed on a wider range of synthetic images with more realistic surface generation, reflections and lighting conditions, including variations of the angle of light. This will allow for a more flexible model that can deal with a wider range of variations in real images.

Improving the illumination when capturing images may reduce the noise from reflective surfaces. Further, different coloured lights at different angles cast different shadows, which may assist with highlighting faults.

3D volumetric images would assist in detecting irregularities in surfaces. This could be achieved through infrared cameras, similar in design to the Kinect, or through the use of optical coherence tomography. However, hardware may be expensive for high pixel density over a small area.

5 Conclusion

A defect detection model was created for gear images and applied to both synthetic and real images. A defect detection model was also created for spring images. Performance for springs was worse. Suggestions were provided to improve the defect detection technique which was applied.

A classification program was written to classify images as either gears or springs, and then apply the appropriate defect detection model.

6 References

[1] L.-C. Chen, G. Papandreou, F. Schroff and H. Adam, “Rethinking atrous convolution for semantic image segmentation,” *arXiv preprint arXiv:1706.05587,* 2017.

[2] M. S. Minhas, “Transfer Learning for Semantic Segmentation using PyTorch DeepLab v3,” GitHub.com/msminhas93, 12 September 2019. [Online]. Available: https://github.com/msminhas93/DeepLabv3FineTuning. [Accessed 2 February 2022].