Defect Detection Based on Synthetic Dataset

Author: Yaseen Moolla

Date: 9 February 2022

# **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.

# Part 1: 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.

## 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).

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.

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.

[Train graph]

Vertical lines, possibly due to overtraining or using incorrect loss function.

Table 2 shows the precision and recall. A low precision indicates a large number of false positives. A visual analysis shows that some regions have false area, such as vertical lines in Figure 3. This may be due to low because… Improve by…

Recall low because… improve by…

Dents vs scratches

Reference files

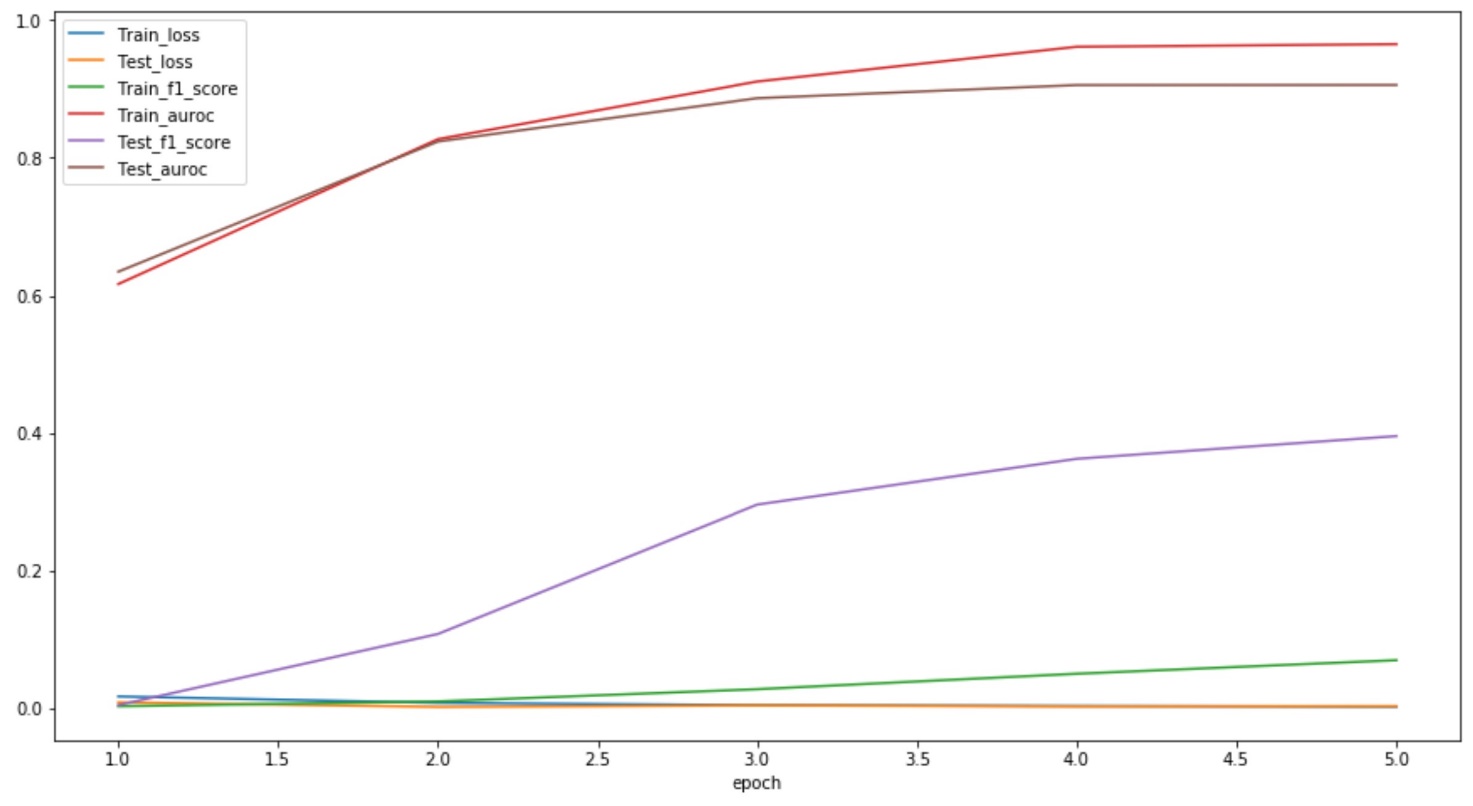


Figure 1: Performance for training defect detection segmentation on gear images over 5 epochs

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 |

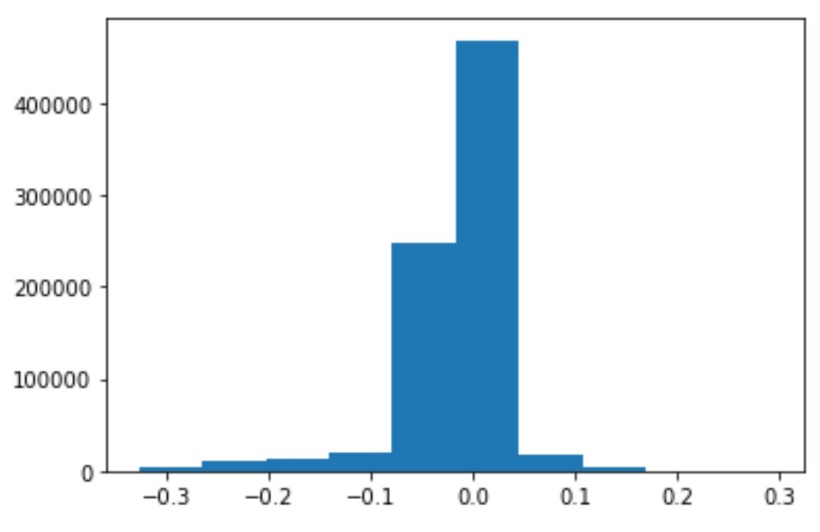


Figure 2:

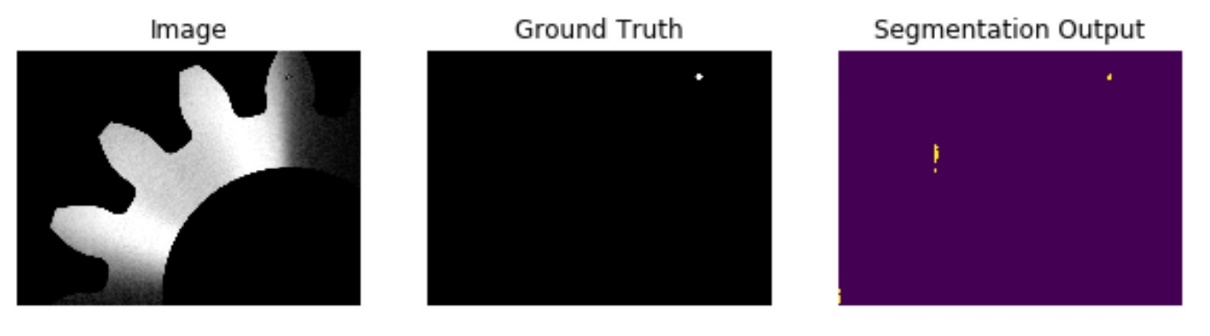


Figure 3:

## Assessment of Real Gear Images

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

Images needed to be resized to match the size of synthetic images for effective application of the model.

Further images and masks are available in the folder Processed Real Gear Images.

[TODO: add in folder 3 and 4.

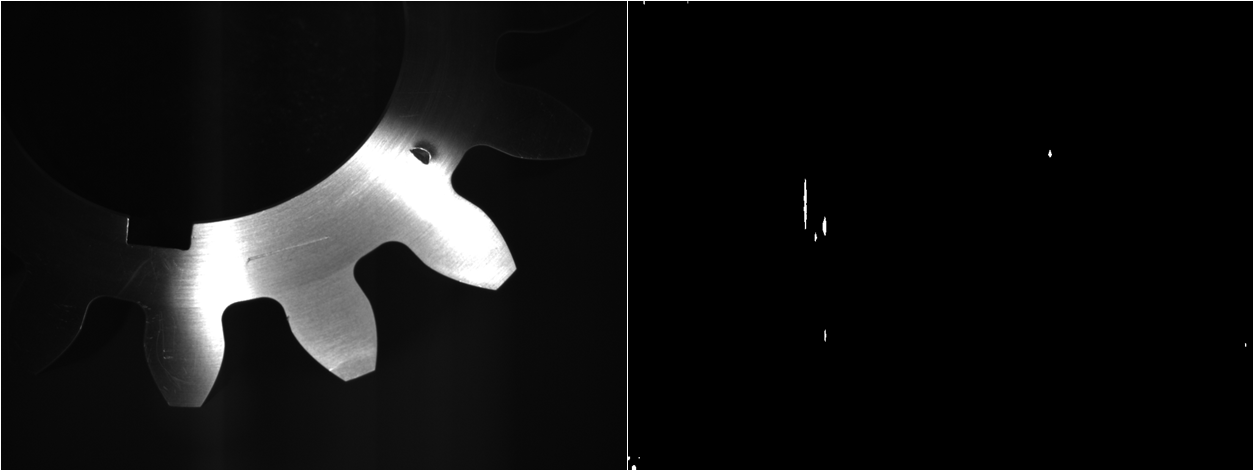


Figure 4: A sample of defect detection on real images. The real image is on the left and the detected defects are on the right.

## Generation of Synthetic Gear Images

Used Paint 3D. Limitations on ability to generate various surfaces; adjust the angle, intensity and wavelength of light; and reflections off surfaces.

Further images and masks are available in the folder GeneratedSyntheticGears.



Figure 5:



Figure 6:

## Defect Detection for Springs

Performance is inconsistent. Ideal Thresholds for segmentation vary greatly between different image types, since a spring comprises of different parts of varying shapes and sizes.. Sub-classification of spring images and training a different model for each sub-class may result in better performance.

Lower F1 and AUROC than gears. Possibly due to wider range of angles and surface shapes from which images were captured. It may be more effective to train separate segmentation models for each surface type and train a classifier (or use classic image processing) to determine which segmentation model to use.

Chart, line chart

Description automatically generated

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

[Table]

A picture containing shape

Description automatically generated

Figure 8:

A screenshot of a computer

Description automatically generated with low confidence

Figure 9:

Graphical user interface

Description automatically generated with low confidence

Figure 10:

# Classification

Transfer learning on VGG [add citation]. VGG is a popular image classification model based on Convolutional Neural Networks. Train: validate: test was applied on an 80:10:10 ratio. The test results resulted in 100% classification accuracy.

Used transfer learning on the VGG-16 (Very Deep Convolutional Networks for Large-Scale Image Recognition) convolutional neural network. It’s a commonly used baseline. Small and fast, compared to ResNet and newer classification models, which may be overkill.

Skipped image augmentation for simplicity. Although, augmentation may make classification more robust and accurate, especially on real images or different synthetic images.

Training validation and testing were performed on an approximately 8:1:1 ratio.

loss: 7.6346 - acc: 0.7111 - val\_loss: 1.6257e-11 - val\_acc: 1.0000

Training accuracy: 0.71%

Validation accuracy: 100%

Test accuracy: 100%

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.



Figure 11: Current dataflow for classification and defect detection



Figure 12: Proposed future design for classification and defect detection

# Suggestions for future improvements

Different loss function.

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.

Try other segmentation models, eg. …?

GAN for image generation.

Synthetic images with more realistic surface generation, reflections and lighting conditions, including variations of the angle of light.

Future work: Better lighting when capturing images. E.g. Different coloured lights at different angles cast different shadows, which may assist with highlighting faults. Alternatively, 3D volumetric images would assist in detecting irregularities in surfaces, but hardware may be expensive for high pixel density over a small area.

# 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]. |