



Automatic Defect Inspection Using the NVIDIA End-to-End Deep Learning Platform

WP-09268-001 | June 2019

White Paper



Document Change History

WP-09268-001

Version	Date	Authors	Description of Change
01	2018-10-30	Peter Pyun, Andrew Liu, and Robert Sohigian	Initial release
02	2019-03-12	Peter Pyun, Charles Cheung, Andrew Liu, and Robert Sohigian	Updated inference experiments on T4 GPUs / TRT5 engine.
03	2019-06-17	Peter Pyun, Robert Sohigian	Updated inference experiments run on Jetson AGX Xavier and Nano platforms.

Abstract

Quality requirements for manufacturers are becoming higher to meet customer demands. Manual inspection is traditionally required to guarantee product quality. But this requires significant cost, production bottlenecks, lowered productivity, and reduced efficiency. Automating defect inspection with artificial intelligence (AI) is beginning to revolutionize manufacturing. Deep learning (DL), especially convolutional neural networks (CNN), has proven to be very effective for image detection and classification, and is now being adopted to solve industrial inspection tasks. One popular and effective neural network architecture—originally proposed for biomedical image segmentation—is the U-Net architecture, which is composed of encoders, decoders and skip-connection layers.

Industrial defect detection and classification shares many of the same data challenges as medical image segmentation, including the scarcity of labeled data and highly asymmetric datasets. U-Net has proven to be successful at generalizing performance with regularization techniques. In this white paper, U-Net was utilized to build an end-to-end generic defect inspection model on a public dataset, using the NVIDIA® DL platform for end-to-end training and inference. A recall rate of 96.38% and a 99.25% precision rate with a 0.11% false-alarm rate were achieved. Although traditional computer vision methods might be able to achieve similar results, they typically require intensive human and capital involvement. The U-Net approach avoids labor-intensive and potentially fragile feature engineering and instead allows data-driven models to automatically learn robust feature representations to generate state of the art detection and segmentation results.

By using [NVIDIA Tesla® V100 GPUs](#) with the [NVIDIA TensorRT™](#) (TRT) 4 engine integrated into a TensorFlow (TF) container, inference throughput increased by a factor of 2.1. With an optimized TRT container using the [NVIDIA GPU Cloud](#) (NGC), inference throughput was further improved by a factor of 8.6 compared to native TF. [NVIDIA T4 GPUs](#) and TensorRT 5 engine were used for energy efficient and small form factor inference deployment. Compared CPU-based TF inferencing, inference throughput was increased by a factor of 23.5 based on an optimized TensorRT container from NGC. For low-power and small form factor edge-inferencing, the [NVIDIA Jetson™ Nano platform](#) throughput is 18 fps, and the [NVIDIA Jetson AGX Xavier platform](#) is 228.1 fps (12.7 times faster than the Jetson Nano platform)

Contents

Chapter 1. Automatic Defect Inspection for Industrial Applications.....	1
Chapter 2. Using U-Net to Solve Defect Inspection Challenges.....	4
Chapter 3. Defect Inspection from Inception to Production	7
3.1 Deep Learning End-to-End Workflow.....	7
3.2 Setting the Target Metric and Project Scope	8
3.3 Labeling Defect Images.....	9
3.4 Data Preparation.....	9
3.5 Model Development	11
3.6 Output Tradeoffs Between Precision and Recall	13
3.7 Model Deployment.....	18
Chapter 4. Summary and Future Work.....	21

Chapter 1. Automatic Defect Inspection for Industrial Applications

With increased global competition, manufacturers are seeking to develop smart factory strategies that utilize advanced Information Technology (IT) and Operational Technology (OT) to facilitate and automate their manufacturing processes. To achieve this goal, manufacturing systems are often required to automatically see and understand the world. In this white paper, we focus on the problem of manufacturing automation based on optical inspection using DL.

Optical quality inspection remains one of the common methods to ensure quality control for high-precision manufacturing. However, this step remains a bottleneck to full automation and integration. The quality checks may include a series of manual operations, including visual confirmation to make sure components are the right color, shape, texture, and position, which are very challenging due to wide product variations. Quality inspectors must constantly adapt to different quality requirements for different products, which often leads to inaccuracy and a lack of consistent quality. With ever-increasing production volumes, quality inspectors often suffer from eye fatigue and other health issues caused by repetitive product inspection over long hours, allowing more defective parts to pass. Human inspection is constrained by increasing cost, making it a challenging solution to scale.

A common type of industrial defects are local anomalies on homogeneous surfaces. Proposed approaches to automate the detection and classification of these anomalies can be divided into four categories:

1. Structural based on defect morphology¹
2. Statistical texture measure-based^{2,3}
3. Hand-crafted transform filter-based^{4,5}
4. Machine learning model-based⁶

Before DL, most of these traditional approaches were designed with hand-crafted features, making them application dependent and not able to generalize or scale-out to new applications. These traditional approaches also typically suffered from poor flexibility and often required expensive and time-consuming manual feature engineering by domain experts. In contrast to traditional methods, DL performs automated feature extraction using a data driven method that does not rely on hand-crafted features. A nearly overwhelming corpus of CNN-based defect classification models have been proposed in recent years^{7,8,9}. Industrial defects are often

¹ " Detecting Defects in Fabric with Laser-Based Morphological Image Processing"
<http://journals.sagepub.com/doi/10.1177/004051750007000902>. [Accessed 2019-06-17]

² T. Vujasinovic, J. Pribic, K. Kanjer, Milosevic NT, Z. Tomasevic, Z. Milovanovic, D. Nikolic-Vukosavljevic, M. Radulovic, Gray-level co-occurrence matrix texture analysis of breast tumor images in prognosis of distant metastasis risk, *Microscopy Microanal.*, vol. 21, no. 3, pp. 646–654, 2015.

³ T. Mäenpää and M. Pietikäinen, Texture analysis with local binary patterns, *Handbook Pattern Recognit. Comput. Vis.*, vol.3, pp. 197–216, May 2005.

⁴ F. Malik and B. Baharudin, the statistical quantized histogram texture features analysis for image retrieval based on median and Laplacian filters in the DCT domain, *Int. Arab J. Inf. Technol.*, vol. 10, no. 6, pp. 616–624, 2012

⁵ H. Ji, X. Yang, H. Ling, and Y. Xu, Wavelet domain multifractal analysis for static and dynamic texture classification, *IEEE Trans. Image Process.*, vol. 22, no. 1, pp. 286–299, Jan. 2013

⁶ " 3D surface inspection using coupled HMMs"
<https://ieeexplore.ieee.org/document/1334508/>. [Accessed 2019-06-17]

⁷ R. Ren, T. Hung, and K. C. Tan, A generic deep-learning-based approach for automated surface inspection, *IEEE Trans. Cybern.*, vol. 99, no. 2, pp. 1–12, 2017.

⁸ D. Weimer, B. Scholz-Reiter, and M. Shpitalni, Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection, *CIRP Ann.-Manuf. Technol.*, vol. 65, no. 1, pp. 417–420, 2016.

⁹ J.-K. Park, B.-K. Kwon, J.-H. Park, and D.-J. Kang, Machine learning-based imaging system for surface defect inspection, *Int. J. Precision Eng. Manuf.-Green Technol.*, vol. 3, no. 3, pp. 303–310, 2016.

relatively small, so it is necessary to localize the defect area in addition to detecting and classifying them.

Defect inspection for industrial applications has unique characteristics and challenges compared to other computer vision problems for consumer applications:

- ▶ **Lack of labeled, annotated, and curated data.** Curating quality datasets requires manpower, time, and budget. Before executing a data campaign, feasibility testing with a limited labeled dataset is beneficial. Datasets with very few labeled defects and with abundant instances of normal class are heavily unbalanced and require augmentation. Proper data augmentation that preserves the statistical properties of the true physical behavior is a key technique to complement defect classes and to balance datasets.
- ▶ **Defects of interests are structures with low contrast.** Industrial inspection applications from different equipment often produces different sized images with different contrast. Often defects are with low contrast. An example of this is the [DAGM 2007 dataset](#).
- ▶ **Multi-scale defect sizes.** Trained DL models are typically scale invariant, meaning they need not be retrained across image sizes to be effective for multiple input sizes. CNN based DL can segment arbitrary input size. The segmentation algorithm should be end-to-end, segmenting defects in one shot execution of pipeline.

Detection algorithms should be able to segment defects from images with a variety of background, focus, rotation, scale, occlusion, and lighting. They also need to detect and locate defects with the required precision and meet production goal of latency and throughput. DL has shown performance improvement over hand-crafted machine vision algorithm in computer vision since 2012. In this white paper, DL is applied to detect defects in an industrial dataset reliably and efficiently. Specifically, the U-Net DL architecture has been applied to segment 2D defects as discussed in the next chapter.

Chapter 2. Using U-Net to Solve Defect Inspection Challenges

Deep neural networks (DNNs) have attained impressive breakthroughs in various domains, such as image classification, object detection, and semantic segmentation.¹⁰ Successful applications span across business verticals, including consumer internet services, healthcare, autonomous vehicles manufacturing, and many others.

In the last few years, one of the most successful state-of-the-art DL methods for single forward pass segmentation has been based on the fully convolutional neural network (FCN)¹¹. The main idea of this approach is to use CNN as a powerful feature extractor by replacing the fully connected layers with convolutional layers to output spatial feature maps. Those maps are further upsampled to produce dense pixel-wise output. This method allows training CNN to segment images of arbitrary sizes. Moreover, this approach achieved an improvement in segmentation accuracy over common methods on standard datasets like [PASCAL VOC](#).

This method has been further extended and improved for a variety of use cases, including a popularly cited network architecture called the U-Net neural network architecture¹². In this white paper, we apply U-Net as a DL model for 2D industrial defect inspection. U-Net is a CNN architecture that can detect, classify, localize, and segment defects at the same time. The basic architecture is an encoder-decoder pair with

¹⁰ "Deep Learning for Computer Vision Tasks: A review." 2018-04-2018, <https://arxiv.org/abs/1804.03928>. [Accessed 2019-06-17]

¹¹ J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In CVPR, 2014.

¹² O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In MICCAI, pages 234–241. Springer, 2015.

skip connections to combine low-level feature maps with higher-level ones (see Figure 1, which is from the published U-net paper, for an overview of the architecture). An encoder contracts the whole image for analysis and then a decoder successively expands an encoded bottleneck layer output to produce a full-resolution segmentation result. Many feature channels in the upsampling part allows context information to propagate to higher resolution layers.

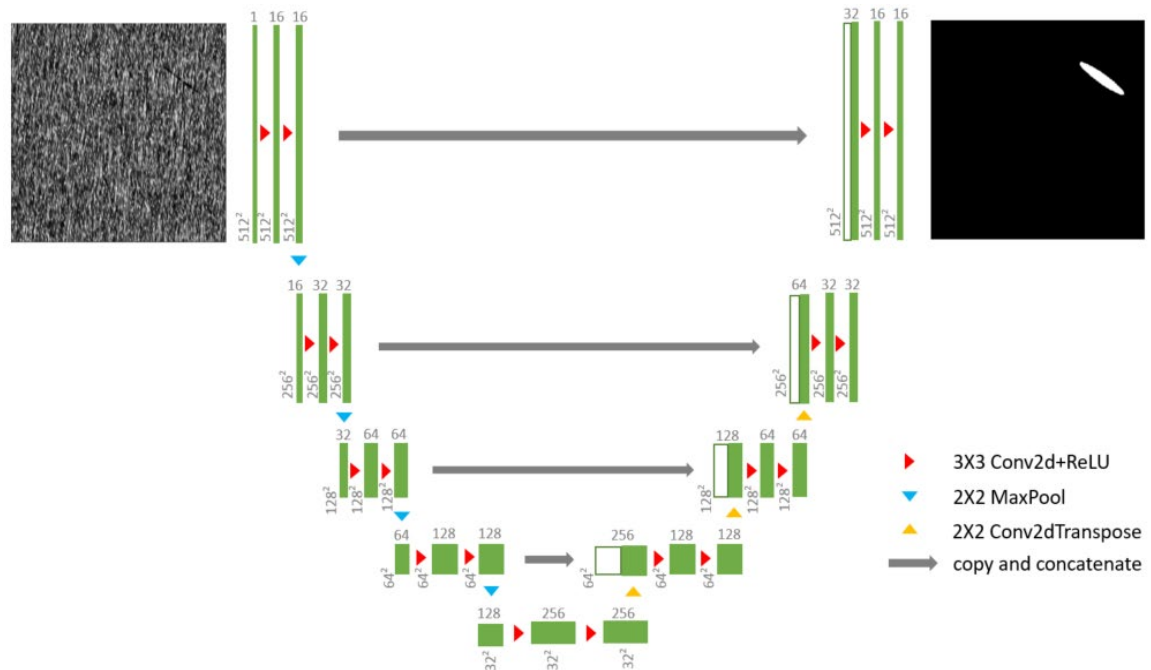


Figure 1. U-Net architecture

Modified U-Net is a suitable model for defect segmentation of DAGM dataset. When there is a shortage of labeled data and fast performance is needed, U-net is a great choice. On different scenarios of industrial inspection, there are other DL networks worth mentioning. If there are golden samples available paired with defect samples, pairing two-channel input and feeding into the network or even Siamese network¹³, can increase discriminative power of the model. For example, this is the case of an AOI machine example from PCB assembly manufacturing. If there is no shortage of labeled data and a bounding box as well as a segmentation mask are needed, Mask R-CNN¹⁴ can be used. With decoupled mask and class prediction, inference speed of 5 fps on NVIDIA Tesla M40 GPU was reported for Mask R-CNN. If only bounding box is needed, many well-known object detection methods can be applied. Abundant object detection methods were proposed in the field of autonomous driving. For example, Single-Shot

¹³ G. Koch, R. Zemel, R. Salakhutdinov Siamese Neural Networks for One-shot Image Recognition. In ICML 2015

¹⁴ K. He, G. Gkioxari, P. Dollar, R. Girshick Mask R-CNN in ICCV 2017

Detector (SSD)¹⁵ and YOLO v3¹⁶ are publicly available. Many derivatives of these methods were published and demonstrated for real-time super-human performance of object detection.

To address a strong data imbalance in the dataset (such as far more nominal than defective data), random elastic image deformation is used as a data augmentation technique to enable the end-to-end training of the network. These results show that U-Net learns to predict segmentation accurately with good generalization by using a semi-supervised end-to-end training approach. A great deal of low-level information is shared by skip-connection to a symmetrical layer in encoder-decoder architecture. Shortcutting low-level information produces high-quality results.

U-Net was initially proposed for medical image segmentation and won the international segmentation and tracking competition in 2015¹⁷. Since then, U-Net has also been successfully applied for a variety of applications outside of medical imaging, including source separation (singing voice)¹⁸, 3D dense volumetric segmentation¹⁹ from sparse annotation, and image-to-image translation²⁰.

The defect inspection and localization models were trained and evaluated using the public dataset originally introduced for the DAGM 2007 Competition of “Weakly Supervised Learning for Industrial Optical Inspection”.

¹⁵ W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. E. Reed. SSD: single shot multibox detector. CoRR, abs/1512.02325, 2015

¹⁶ J. Redmon and A. Farhadi. Yolov3: An incremental improvement. CoRR, abs/1804.02767, 2018.

¹⁷ "U-Net: Convolutional Networks for Biomedical Image Segmentation"
<https://arxiv.org/abs/1505.04597>. [Accessed 2019-06-16]

¹⁸ Jansson, A., Humphrey, E., Montecchio, N., Bittner, R., Kumar, A. & [Weyde, T.](#) (2017). Singing voice separation with deep U-Net convolutional networks. Paper presented at the 18th International Society for Music Information Retrieval Conference, 23-27 Oct 2017, Suzhou, China.

¹⁹ O. Cicek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. arXiv preprint arXiv:1606.06650, 2016

²⁰ P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In CVPR, 2017

Chapter 3. Defect Inspection from Inception to Production

3.1 Deep Learning End-to-End Workflow

An overview of the developer workflow of a DL project, from pre-training to training to inference, is shown in Figure 2. All end-to-end processes are built on top of NGC optimized docker images for fast iterations. Pre-training involves model development, debugging, testing, and validation. During pre-training, a data scientist pulls the optimized images from the NGC docker registry and directly works on top of pulled images.

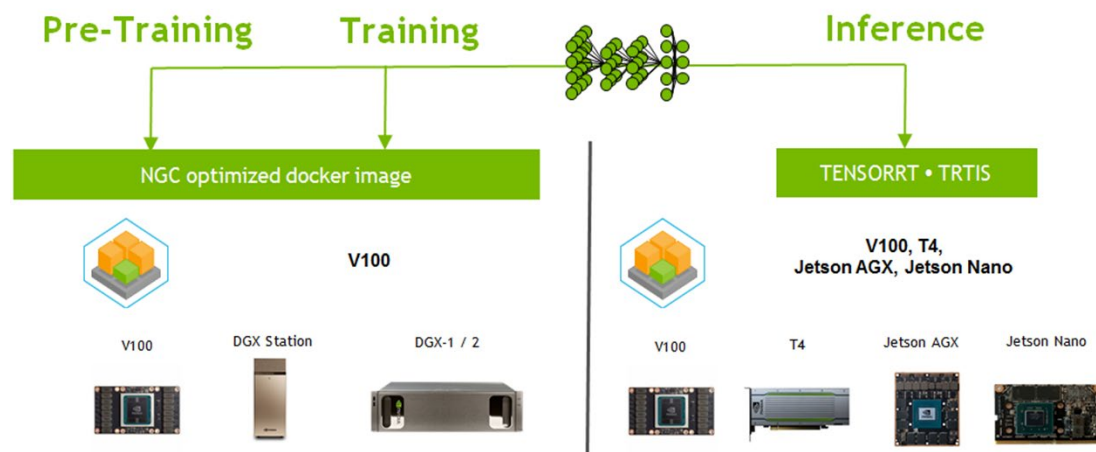


Figure 2. Defect inspection from inception to production

As a best practice, before starting large-scale training with a powerful machines such as [NVIDIA DGX™ servers](#), the engineer may run a few training epochs to test and validate the software stack and system configuration. If the training behaves as expected after a few epochs, the engineer can then shift the process to a full-scale training run to fully train

the neural network model. At this point the model is ready for large-scale training and can be moved to a powerful training system easily and conveniently with [GPU-enabled docker containers](#). After the model is trained and achieves desired target metrics, the model can be optimized for performance, latency, and efficiency with TRT. Easy deployment of TRT is also available from NGC TRT docker containers. Furthermore, [TensorRT inference server](#) (TRT-IS) enables flexible deployment of the inference model.

TRT is integrated into TensorFlow 1.7+ branches (see [this](#) NVIDIA Developer Blog for details). With a simple API call to TRT, powerful mixed precision Tensor Core inferencing on Tesla V100 GPUs can be applied without leaving TF. In this white paper, we used [TensorFlow:18.08-py3](#) NGC container, which was built based on TensorFlow 1.9.0 and TensorRT 4.0.1. Advanced users can use native TensorRT inferencing with [tensorrt 18.08-py3](#) container. We will share inferencing experimental results based on both TensorFlow-TensorRT (TF-TRT) and TRT in section 3.6. Using T4 GPUs powered by [NVIDIA Turing Tensor Cores](#), we further experimented inference with NGC container [TensorFlow:19.01-py3](#) and [TensorRT:19.01-py3](#), with results updated in section 3.6. These containers were built with TensorFlow 1.12.0 and [TensorRT 5.0.2](#), respectively. Once test feasibility is verified on a datacenter GPU like V100 or T4 with TF-TRT for fast prototyping, then Jetson Nano or Jetson AGX Xavier platform can be used for low-power edge-inferencing.

3.2 Setting the Target Metric and Project Scope

The first step is to decide performance evaluation metrics for the defect inspection project. This metric can be [precision and recall rates](#), or the harmonic mean of the two, and [F1-measure](#). The single-number evaluation metric is vital for a successful DL project since the follow-up experiments would be developed upon this metric²¹. In a typical defect inspection scenario, recall rate would usually be required to achieve 100%, which means any defect escape is not allowed. What we are trying to achieve with DL is to increase the precision rate, or in other words, to reduce false alarm rate while keeping 100% recall of all defects. However, achieving both high recall and precision a hard problem as shown in section 3.6 and the tradeoff is needed per application.

There may be more than one defect class in a typical defect inspection scenario, and we would suggest focusing on top N defect classes first. N is 5 or 10 primary defects depends on your scenario and application. Instead of setting up a huge scope to tackle all defect classes, an alternative is to select N defects and start to collect and label images.

²¹ A. Ng, Machine Learning Yearning <http://www.mlyearning.org/>

3.3 Labeling Defect Images

In our application, labeled defect images are composed of binary outputs, where zeros represent defect areas shown in white, and ones represent defect-free areas shown in black in Figure 3.

In most cases that you need labeling, there are many open source labeling tools in GitHub, such as [LabelMe](#). They can be integrated and implemented with an in-house data preparation pipeline. It is an important interactive step to build a smooth data preparation pipeline with correct labeling. In our experience, the first label output from labelers might not be correct. It is typical for labeling to go through iterative review process. The challenge is that not all defect patterns are easy to recognize by people without necessary training. In practice we have found that this labeling process can be one of the most productive ways to leverage your most valuable domain experts.

In a related case that occurred at NVIDIA, three labelers reviewed a single defective image to draw bounding boxes. To speed-up review process, reference and defective images were combined to create a side-by-side view. After the first iteration of labeling, between 5-10% mismatched labels were identified. The mismatched images were then isolated, and the labels were again reviewed. This process was repeated until a 0% mismatch between labelers was reached. Finally, the average of the three labelers dimensions was calculated and used for modeling.

Since the outputs of a DL segmentation model are probabilities of all image pixels, we can fully utilize these values as a threshold to achieve project target metrics. We will illustrate this topic in Section 3.4.

3.4 Data Preparation

Segmentation is a supervised learning process that maps input images to an output segmentation map. DL trains this nonlinear complex mapping function. The following image shows the input image and its corresponding output images of six DAGM defect classes. Each defect class contains 150 defect images and 1000 defect-free images. The defect inside an image was bounded with an ellipse. The parameters of this ellipse were recorded in a separate text file, including semi-major axis, semi-minor axis, rotating angle, and x, y position of the center of the ellipsoid.

OpenCV was used generate the respective label image with these parameters as shown in Figure 3.

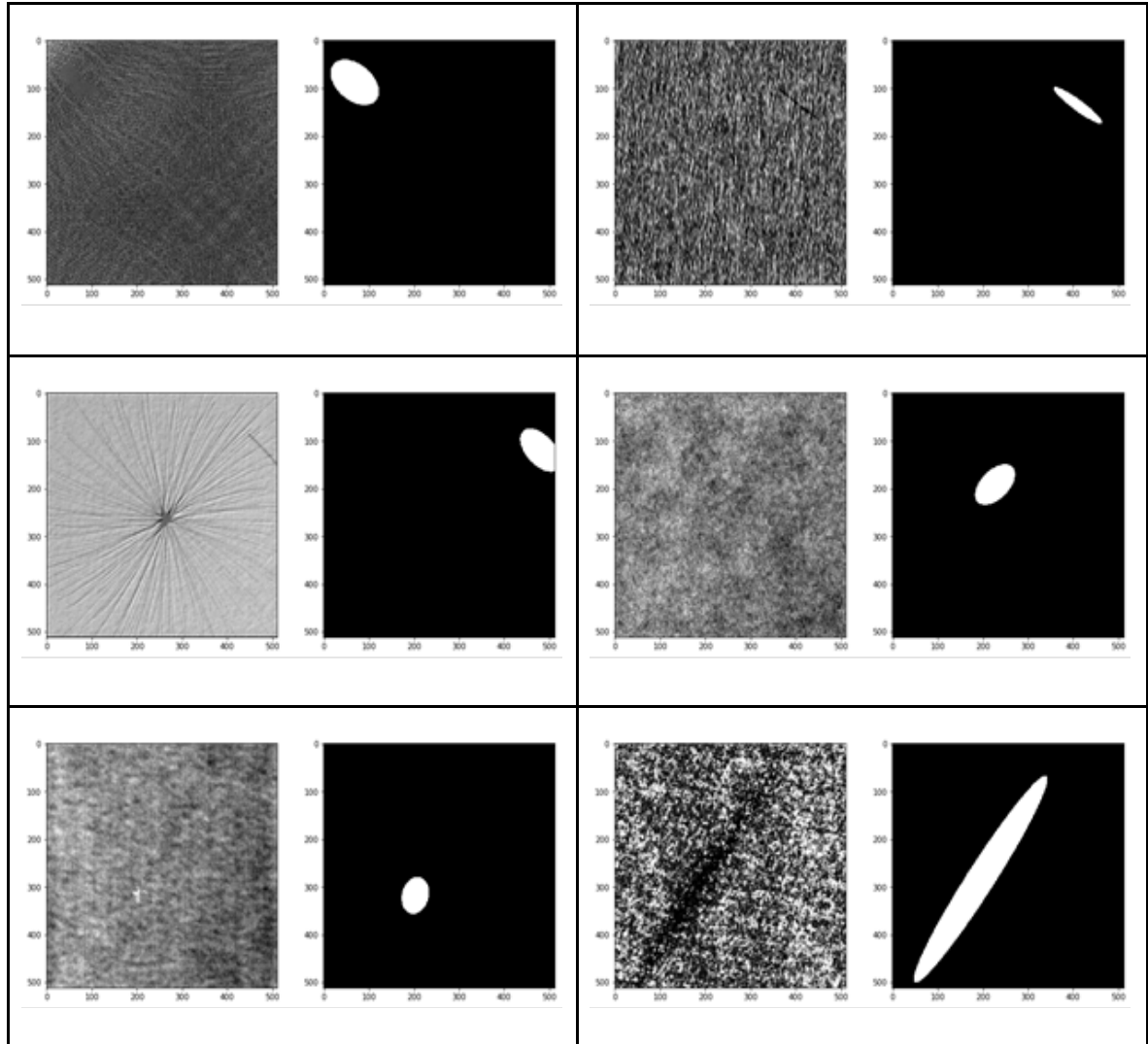


Figure 3. DAGM defect images and its corresponding labeled output

Two different settings were prepared using the same dataset (Table 1). Training Set 1 is used to train the model to detect each defect class, totaling six defects. Therefore, the defect images of each class will be prepared separately. 100 defect images are used for each defect class to train the U-Net model, the other 50 defect images and 1000 defect-free images will be used to test the model, and provide average IOUs²², precision and recall rates. Training Set 2 is used to demonstrate the generalization capability of U-Net. All defect classes in Training Set 2 are unified and treated as a single class. We will build a single U-Net model that is able to segment all defect classes. This kind of process is

²² "Intersection over Union (IoU) for object detection - PyImageSearch." 2016-11-07, <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. [Accessed 2019-03-11]

common in defect inspection. In this white paper, we will explain the DL application process for Training Set 2.

Setting	Training Set Size	Validation Set Size	Testing Set Size Defect / Defect-free
1	100	30	20 / 1000
2	630	138	132 / 6000

Table 1. Dataset preparation

3.5 Model Development

Once the dataset is ready with defect images and its corresponding labeled images, we can start to build the mapping function with U-Net deep learning segmentation model. Many best practices on how to set up appropriate experiments to achieve target metrics can be found in Andrew Ng's [Machine Learning Yearning book](#). To make modeling work easier, we use the same architecture as U-Net, and experiment with the number of kernels to make the model fit into our dataset. U-Net is a flexible DL model. The first step is to fit the dataset with U-Net model, i.e., eight kernel numbers in the first layer, then doubling the kernel number in the following hierarchy of layers, as shown in Figure 1. Then we observe the loss from the training learning curve and decide whether to increase further the model complexity. This step is to make sure that we have a model with enough capacity to recognize all defects. Once you have confidence that the model is complex enough to fit the dataset, then add regularization techniques, such as drop out layers, L1 or L2 regularization, or even try to augment the dataset. The goal is to make sure the loss on validation learning curve drops smoothly along with the loss on training. This prevents overfitting the model.

In this white paper, U-Net model was built with eight kernel number in the first layer for both training sets. The other parameter that was ignored was padding. Since U-Net modeling is an encoding-decoding process, it first encodes the input image with several convolution and max pooling layers, then decodes to the output layer with the same number of convolution layers and max pooling layers. It is a symmetric mapping function with the same layer numbers in both encoding and decoding sections. Having the same padding setting makes things much easier, since each feature map convolved by kernels remain the same size. The dropout layer is the only layer that would decrease the shape of feature maps. The input image size should be a power of two (as per Figure 1). Every image was resized to 512x512 from the original DAGM dataset of size 500x500.

The model was trained with binary cross entropy and Adam optimizer²³ with a learning rate starting with 1e-3. As shown in Figure 4, learning curves for both training and validation loss decline smoothly over 50 epochs. This means the learned U-Net model can learn to fit to the training and validation dataset well.

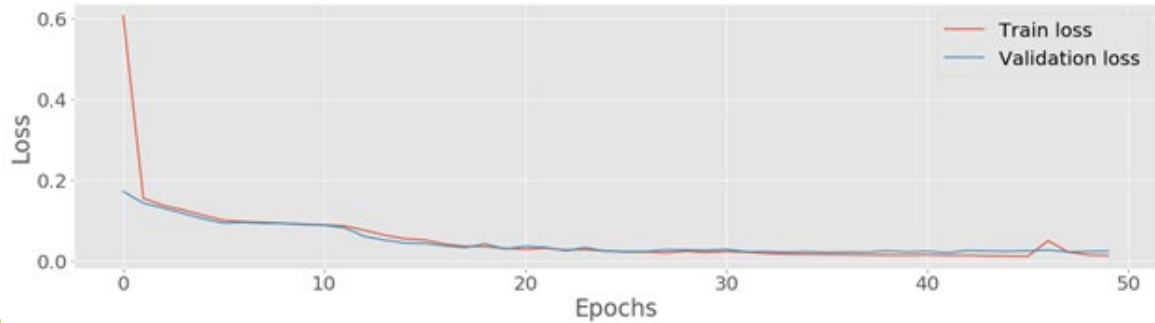


Figure 4. Learning curve

After training for 50 epochs, Figure 5 shows the prediction result of U-net compared to Ground truth.

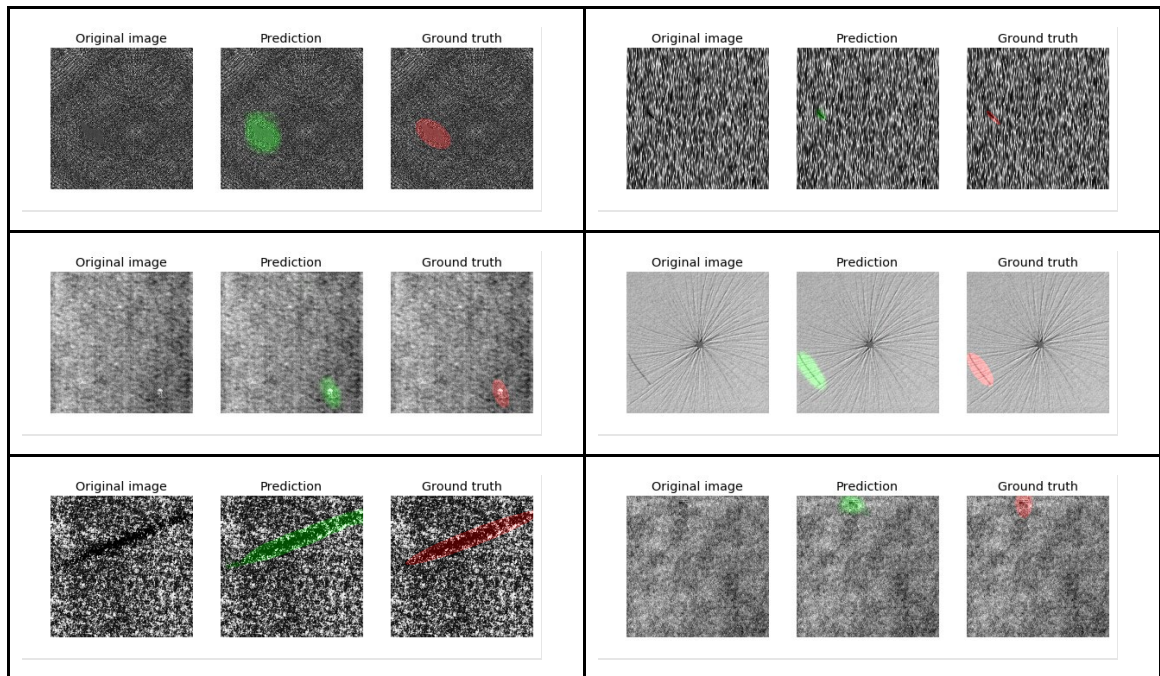


Figure 5. U-Net prediction results

²³ "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning" 2017-07-03, <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>. [Accessed 2019-03-11]

3.6 Output Tradeoffs Between Precision and Recall

The output of a classification model is a vector of probabilities, where each vector represents class probability of each class of the query image predicted by the model. The final predicted class is assigned with maximum likelihood criteria. In a segmentation model, like U-Net, the output is also a set of probabilities. These probabilities are well organized in the original query image format.

Table 2 shows an example output from a segmentation model. The original output from the segmentation model is shown in left column in a NumPy array format. In the right column is the visualized output that would usually be seen when printed out by matplotlib. These digits are probabilities of all pixels of 512x512. The threshold was set to 0.5 by default, i.e., every pixel was shown in white if the number is above 0.5, otherwise in black. This finally leads to a visualization to show us the defect area in the image.

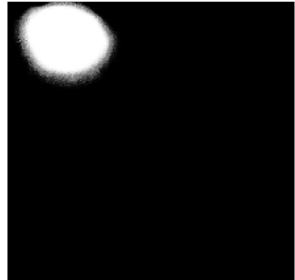
NumPy Output	matplotlib Output
<pre>array([[6.17885776e-03, 3.82234044e-02, 9.50025606e-06, ..., 4.50918742e-05, 3.49759248e-05, 3.65408661e-04], [6.45390755e-05, 4.58258086e-07, 2.12041887e-05, ..., 3.15845439e-09, 1.69029056e-06, 1.10975248e-04], [2.03725667e-05, 4.74613626e-06, 6.89793808e-07, ..., 6.43013749e-08, 1.97115969e-06, 2.85665534e-04], ..., [2.50566706e-10, 4.80150497e-08, 2.86757146e-10, ..., 9.31098111e-06, 2.05957076e-05, 4.73519601e-03], [1.80557666e-10, 1.41850676e-09, 1.18475485e-09, ..., 2.04379503e-05, 3.10234725e-03, 4.20572087e-02], [1.74140851e-07, 1.64427387e-08, 2.98866799e-11, ..., 3.39166650e-06, 1.28269540e-02, 2.99611967e-02]], dtype=float32)</pre>	

Table 2. Output from U-Net model

The following abbreviations are used throughout the rest of this section:

- ▶ TN: true negative
- ▶ FN: false negative
- ▶ TP: true positive
- ▶ FP: false positive

Figure 6 is a simple illustration of binary anomaly detection, visually showing the impacts of threshold change on precision and recall. For illustration only, defects and non-defects distributions are Gaussian distributions, and defects are fewer than non-defects. In real scenarios, each distribution would be more complicated, most likely with multiple peaks, either from Gaussian distribution (from Central Limit Theorem) or non-Gaussian distribution. In addition, defects populations are typically much smaller than non-defects in industrial applications.

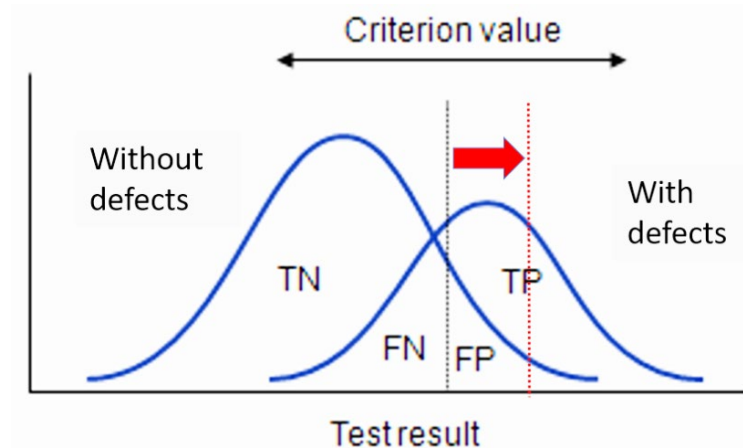


Figure 6. Illustration of simple binary anomaly detection

In the experimental results in Table 3, defects are about seven times fewer. In real production case, defects are much smaller, typically in the tens to hundreds of defect parts per million (DPPM) range. The red arrow in Figure 6 shows an increase in the probability threshold, raising the criteria for predicting a defect. As Figure 6 illustrates, false positives decrease as the threshold increases, thereby increasing precision. However, at the same time, false negatives get larger, reducing recall. This illustrates the fundamental trade-off between precision and recall. Determining the right threshold to bias towards precision or recall is entirely application dependent. This requires sweep experiments of precision and recall on the threshold of probability as shown in Table 3. If reducing false positives (increasing precision) is more important, you need to increase the threshold on probability while balancing precision-recall tradeoff. Achieving high precision and high recall means your classifier has high discriminative power and the areas under the [receiver operating characteristic](#) (ROC) curve is a measure of how well classifier works.

Figure 7 is another graphical illustration of a precision-recall diagram. You can think of this diagram as projecting data distributions in Figure 6 into x-y plane from z-axis. The area inside the circle is what ML algorithm detects as a defect. 7(a) and 7(b) show the impact of precision and recall depending on the threshold of probability gets higher. If the threshold on probability becomes higher, defect detection becomes harder, making the circle smaller, and shifting the decision circle towards defects areas. In summary, false negatives increase and false positives decrease, making recall lower and precision higher.

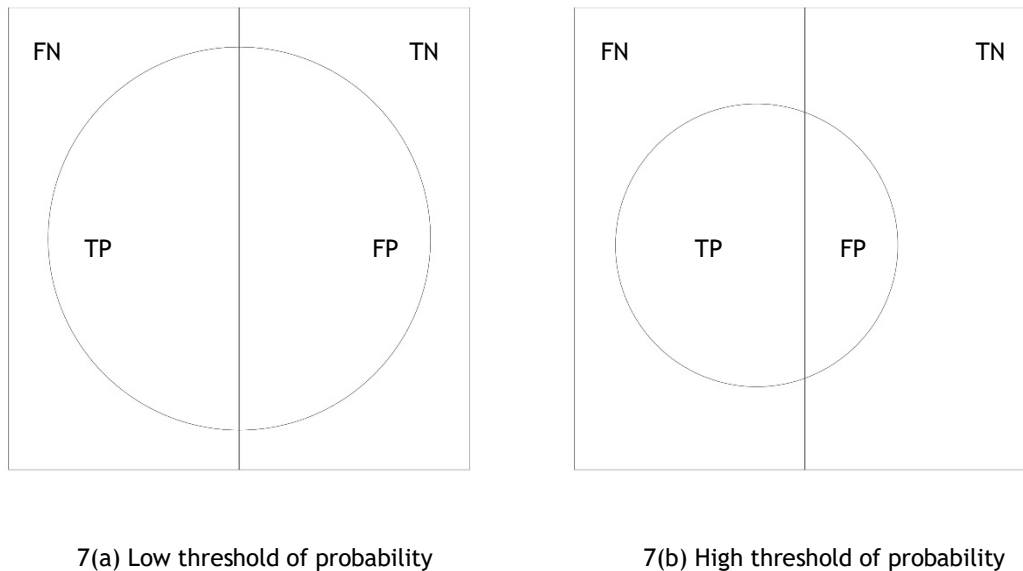


Figure 7. Precision-recall diagram on different thresholds of probability

In real scenarios, data distribution as well as selecting the decision boundary from anomaly detector based on DL is much more complicated to visualize. The journal Nature's 2015 paper titled "Deep Learning" by LeCun, Bengio, Hinton²⁴ has a great discussion and illustration on how learning data representations using DL transforms a high-dimensional dataset into a suitable internal representation and automatically discover the best representation from which to define boundaries needed for detection or classification. These learned internal representations allow decision boundaries to be defined that have been shown to drastically reduce the tradeoff between increasing sensitivity to true positives while significantly reducing the rate of false positives. For industrial applications where the cost of false positives can be extremely high, this can be a source of tremendous value.

²⁴ Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning, 436, Vol 521, Nature, 2015.

Table 3 shows precision-recall tradeoff from the sweep experiments varying the threshold of probability. Total image data were 1039 (defects: 138, non-defects: 901).

Experiment	Threshold								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
TP	137	135	135	135	135	135	135	133	131
TN	885	893	899	899	899	899	899	900	901
FP	16	8	2	2	2	2	2	1	0
FN	1	3	3	3	3	3	3	5	7
FP rate	0.0178	0.0089	0.0023	0.0023	0.0023	0.0023	0.0023	0.0011	0.0000
Precision	0.8954	0.9441	0.9854	0.9854	0.9854	0.9854	0.9854	0.9925	1.0000
Recall	0.9928	0.9783	0.9783	0.9783	0.9783	0.9783	0.9783	0.9638	0.9493

Table 3. Precision-recall experiments

With a threshold of 0.8, reasonable 99.25% precision, 96.38% recall, and 0.11% false alarm rate were achieved. Table 4 is another way to view the same result.

		Actual	
		Defect	Defect-free
Predicted	Defect	99.25%	0.75%
	Defect-free	0.55%	99.45%

Table 4. Confusion matrix

In a production case at NVIDIA, a non-DL based AOI machine in PCBA manufacturing produces high false positives with low precision. Thus, increasing precision by DL automation is critical.

The U-Net model learns to accurately detect defects from a test dataset. It can generate pixel-wise probabilities that can be used to make final decisions such as whether something has a defect or is defect-free. This two-step process is common and can be applied to many use cases, including medical imaging, video surveillance, and autonomous machines. The first step uses U-Net to extract information from the input, then the second step makes the final decision according to information from the previous step.

Some examples from different verticals are shown in Table 5.

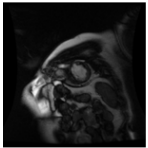
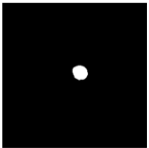

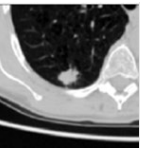
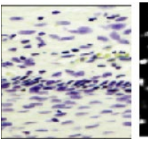
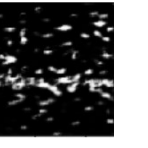



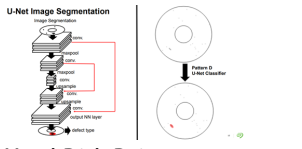
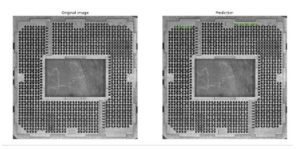
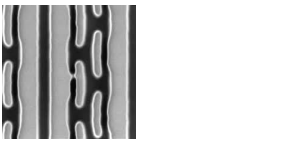
Healthcare			
	  MRI	  CT Scan	  Microscopic Image
Output	Left Ventricle	Nodule	Nuclei
Interpretation	Heart disease	Lung Cancer	Drug Discovery
Smart Cities/Autonomous Machines			
	 Camera Image	 Camera Image	 Camera Image
Output	Human	Road	Road
Interpretation	Intrusion alert	Path for car to drive	Path for drone to fly
Industrial			
	 Hard Disk Drive	 Injection Molding Parts	 Semiconductor Wafer
Output	Defect area	Defect area	Defect (optical hot spot)
Interpretation	Defect or defect free	Defect or defect free	Defect or defect free

Table 5. Segmentation models in different verticals

3.7 Model Deployment

The [NVIDIA Deep Learning SDK](#), including TRT will be helpful before deployment in the field and [TensorRT inference server \(TRT-IS\)](#) enables flexible deployment of the inference model. TRT is a high-performance DL inference optimizer and runtime that delivers low latency and high-throughput for DL inference applications. Based on NVIDIA Docker, the TRT container encapsulates all the libraries, executables, and drivers you need to develop a TRT-based inference application. After downloading an NGC Docker image, you are in a GPU-optimized local development environment for your inference solution with all software and required dependencies installed. This is a starting point for your own container-based edge, datacenter, or cloud deployment.

Applications software engineers or advanced users, who are comfortable with the additional steps required to take a DL model into an environment which might not have TensorFlow framework, are encouraged to use native TRT for maximum performance.

We used [tensorrt 18.08-py3](#) NGC container for our experiments, based on [TRT 4.0.1](#). Data scientists or users for rapid prototyping should run optimized accelerated inference without leaving TF framework. We used TF-TRT inferencing based on [TensorFlow 18.08-py3](#) NGC container. This TF-TRT container integrates with TRT 4.0.1.

Inferencing experimental results based on Tesla V100 GPUs and TensorRT 4 were shown in Table 6. On mixed precision FP16 inferencing, we can see there are performance boost of 2.1 times with TF-TRT and 8.6 times with TRT compared to native TensorFlow. The tradeoff for TF-TRT versus TRT is that TF-TRT is easy to use and integrates with TensorFlow workflows for fast prototyping. Once your idea is verified to work with TF-TRT, TRT can be used for maximum performance.

Precision		Inference Method		
		TF	TF-TRT	TRT
FP32	Images/sec	141.8	236.1	1079.8
	Performance Increase	1	1.7	7.6
FP16	Images/sec	N/A	297.4	1219.7
	Performance Increase	1	2.1	8.6
NOTE: FP32 TRT, FP32/FP16 TF-TRT for batch=128. FP16 TRT for batch=64. FP16 results are all through Tensor Core mixed precision inferencing at Tesla V100 GPU for the input image size of 512x512. Tesla V100 GPUs on a NVIDIA DGX Station™ workstation were used				

Table 6. TF, TF-TRT, and TRT performance comparisons – Tesla V100 GPUs and TensorRT 4 engine

Inferencing experimental results based on T4 GPUs and TensorRT 5 are shown in Table 7. Compared to CPU-based inference performance, there is a performance boost of 11.9 times with TF-TRT and 23.5 times with INT8 precision of TRT5 by [NVIDIA Turing Tensor Cores](#). The T4 GPU is packaged with an energy-efficient 70-watt, small PCIe form factor, optimized for scale-out servers and purpose built to deliver state-of-the-art AI.

Precision		Inference Method			
		CPU-TF	GPU-TF	TF-TRT5	TRT5
FP32	Images/sec	38.6	230.4	320.0	438.8
	Performance increase	1	5.8	8.1	11.1
FP16	Images/sec	N/A	N/A	334.0	501.0
	Performance increase	N/A	N/A	8.4	12.6
INT8	Images/sec	N/A	N/A	459.0	909.0
	Performance Increase	N/A	N/A	11.9	23.5
NOTE: All experiments were with batch=64. INT8 and FP16 results are all through Tensor Core mixed precision inferencing at T4 GPU for the input image size of 512x512. The T4 GPU was used with Intel CPU Skylake Gold 6148 v3 @ 2.40 GHz, dual socket 20-core					

Table 7. TF, TF-TRT, and TRT performance comparisons – T4 GPUs

Using the same code, Jetson Nano platform throughput is 18 fps and Jetson AGX Xavier platform throughput is 228.1 fps — a performance boost of 12.7 times that of the Jetson Nano platform. These results, shown in Table 8, were obtained using native TRT.

Field	Embedded Device		
	Jetson Nano	Jetson AGX Xavier	
Input Precision	FP16	FP16	INT8
Throughput(fps)	18.0	131.9	228.1
Speedup vs. Nano	1.0	7.3	12.7
NOTE: All experiments were with batch=1 and with maximum performance mode at Jetson.			

Table 8. Edge device performance — Jetson Nano and Xavier AGX platforms

With native TRT, a DL framework like Tensorflow does not need to be installed on edge devices. This is important for them since compute power and disk storage are limited.

Chapter 4. Summary and Future Work

As shown in this white paper, the NVIDIA Deep Learning platform can be successfully applied to detection and segment defects in an end-to-end fashion for fast development of automatic industrial inspection. Particularly, the U-Net architecture and the DAGM 2007 dataset was used for this demonstration. U-Net was able to train with good generalization using a small labeled dataset. U-Net produced one-shot high-resolution segmentation results overlaid on top of the input image. For fast deployment of DL training and inferencing, NGC was used for Volta GPU optimized TensorFlow and TensorRT docker containers. We achieved 96.38% recall rate and 99.25% precision rate with 0.11% false alarm rate. Best-performing results on these two metrics are key requirements for industrial inspection.

On Tesla V100 GPUs and TRT4 engine, from a TensorFlow container with a TensorRT engine integrated, inference throughput increased by a factor of 2.1. With an optimized TensorRT container from NGC, inference throughput was further improved by a factor of 8.6. T4 GPUs and TRT5 engine were used for energy efficient and small form factor inference deployment. Compared CPU-based TF inferencing, inference throughput was increased by a factor of 23.5 based on an optimized TensorRT container from NGC. For low-power and small form factor edge-inferencing, Jetson Nano platform throughput is 18 fps and Jetson AGX Xavier platform throughput 228.1 fps.

Future work includes using Siamese network for one-shot learning. This is an interesting idea and worth exploration given the scarcity of dataset industrial inspection. For AOI defect detection on PCBA manufacturing, Siamese network will be explored since they have matching golden samples as well as defects.

Notice

The information provided in this specification is believed to be accurate and reliable as of the date provided. However, NVIDIA Corporation (“NVIDIA”) does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This publication supersedes and replaces all other specifications for the product that may have been previously supplied.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and other changes to this specification, at any time and/or to discontinue any product or service without notice. Customer should obtain the latest relevant specification before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer. NVIDIA hereby expressly objects to applying any customer general terms and conditions about the purchase of the NVIDIA product referenced in this specification.

NVIDIA products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on these specifications will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this specification. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this specification, or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this specification. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA. Reproduction of information in this specification is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, GeForce, and SLI are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2019 NVIDIA Corporation. All rights reserved.