WHITE PAPER

Energy and Utilities

Pipeline Inspection/Assessment, Proactive Maintenance and Repair

Pipe Sleuth with Optimized Inference on Intel® Processors

INTEL®
AI BUILDERS
MEMBER

Executive Summary

Pipe Sleuth is an automated anomaly detection solution designed to eliminate the need for manual review and coding of underground sewer pipeline video scans. It uses advanced image processing and Deep Learning to identify, grade and score overall pipe segments using NASSCO PACP standard. This paper describes an approach used in Pipe Sleuth for improving model accuracy with deep learning. The approach described uses Faster R-CNN[1,2] a CNN-based object detection algorithm, and a ResNet-101[3]-based feature extraction architecture. Training images were created from real sewer pipe assessment videos. The TensorFlow[4] based deep learning model was optimized for inference using Intel's OpenVINO™[5] toolkit. In this solution, different types of anomalies are detected in pipeline assessment videos and inference performance is examined with and without OpenVINO optimization.

Introduction

Regular inspection of underground water/sewage pipelines is required in the water utility industry. Inspection helps prioritize periodic maintenance tasks to avoid leakage, breakage, and blockage, which may result in property damages or safety hazards. Inspection is a manual process and is cumbersome and time-consuming.

NASSCO's PACP[6] (Pipeline Assessment Certification Program) is the North American Standard for pipeline defect identification and assessment, providing standardization and consistency to the methods by which pipeline conditions are identified, evaluated and managed. This standard categorizes the anomalies into different types.[7]

A camera-mounted rover is maneuvered by an operator into the underground pipelines to scan and record videos. The operator notes the observations and generates an inspection log and summary report, which highlights any problems discovered and their locations. Video recordings and summary reports are subsequently reviewed by a quality control department for accuracy. As these pipes are lengthy and numerous, and the number of video logs being generated is high, the process of manually reviewing these data is time-consuming and error-prone. This has motivated utility companies to look for ways to make the pipeline inspection process more efficient. In response, Wipro has introduced an approach to automate pipeline inspection from the recorded videos using techniques in computer vision and machine learning.

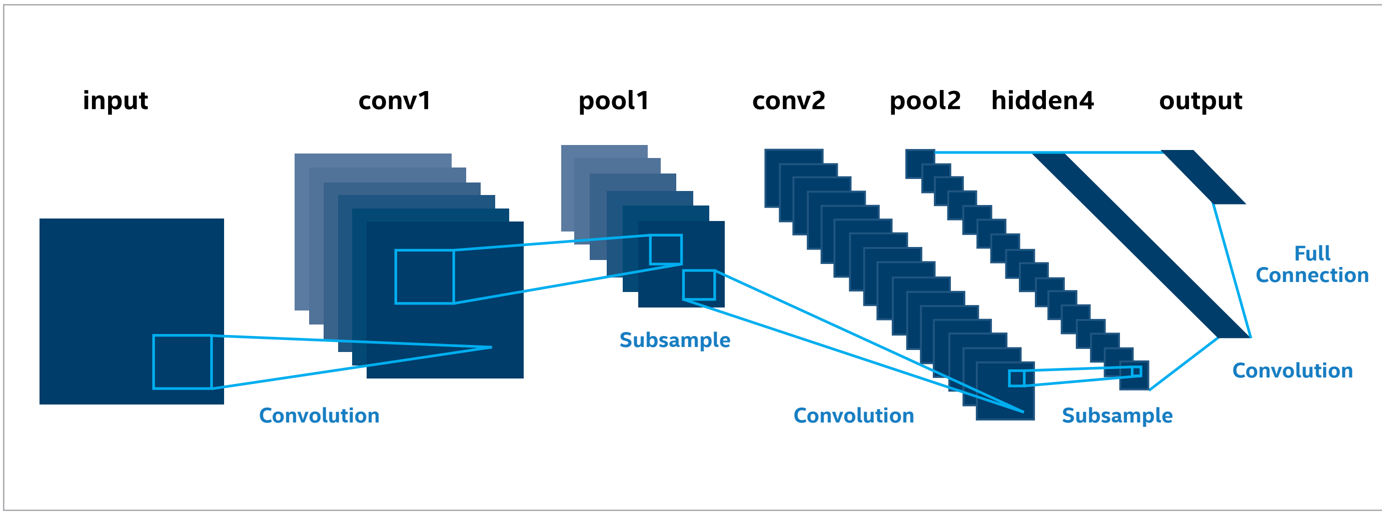

Figure 1. Typical Convolutional Neural Network layers[10]

Anomaly Detection via Deep Learning

Deep learning[8] is a class of machine learning algorithms that uses deeply-nested neural network topologies for feature extraction, transformation, and data classification. The field of computer vision has seen a great deal of success leveraging these methods, particularly the convolutional neural network (CNN[9]) for object detection and classification CNNs are widely used for detecting objects in images and known for their generalizability, particularly in the case of CNN topologies with many layers, as they are able to learn about low-level features (e.g., lines, edges, angles), and abstract them into high-level features (e.g., curved surfaces, textures, etc.) deep within the network. Figure 1 shows the typical CNN layers. In this paper, each anomaly type is treated as an object to be detected and classified.

As an image is passed as input to the network, it is processed and read by various convolutional layers and pooling layers and finally output as an object's label at the last layer of the network. For each input image, the output is one or more bounding boxes and a class label for each bounding box. This technique is used to detect various objects in a given image. CNN divides the image into multiple regions and then classifies each region into various class labels. Therefore, many regions in various shapes and sizes are needed to predict accurately, resulting in a very high computation time. To overcome this bottleneck and reduce the number of regions, Region-based Convolution Neural Network (R-CNN[10]) can be used.

Selection of Object Detection Algorithm and Network Topology

R-CNN, Fast R-CNN and Faster R-CNN are the different algorithms in the R-CNN family. R-CNN extracts different regions from the input image using a "selective search"[11] method and then checks for the object presence in those regions/boxes (Figure 2). First, it extracts these regions, and for each region, CNN is used to extract the specific features. These features are used to detect objects. This process requires high computation time, as each region is passed to the CNN separately.

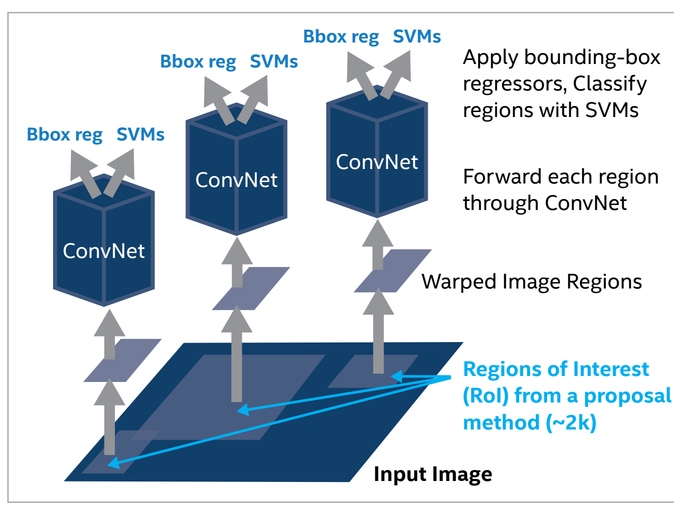

Figure 2. R-CNN[10]

Fast R-CNN[12], instead of running CNN for each region, runs the CNN once per image and then generates the convolutional feature maps (Figure 3). Selective search is used on these maps, and regions of proposal are extracted. A "selective search" algorithm identifies the different regions of proposal by identifying the patterns, which forms an object, such as varying scales, colors, textures, and shapes. As the selective search method involves many image-processing techniques and is slow, the computation time for this Fast R- CNN is still high.

Faster R-CNN[1,2] replaces the selective search method with Region Proposal Network (RPN[13]), which makes the algorithm much faster than R-CNN and Fast R-CNN (Figure 4). Faster R-CNN has two networks: one is a region proposal network (RPN) for generating region proposals, the second one which uses these proposals to detect objects. First the feature maps from the input image are extracted using the convolutional network, and those feature maps are passed to the region proposal network (RPN).

The RPN uses a sliding window on those feature maps, and, for each window, it generates k anchor boxes of different sizes and shapes (Figure 5). For each anchor box, RPN finds the probability of object presence and bounding box coordinates for adjusting the anchors to better fit the object. It returns the object proposals along with their objectness score. Then a ROI pooling layer[14] is applied to resize all proposals to the same size. These proposals are passed to the fully connected layer and then to the softmax layer[15] to classify the object, then through a linear regression layer for the bounding boxes around the object.

Wipro selected Faster R-CNN with ResNet-101[16] topology as the feature extractor to detect different types of anomalies in the extracted video frame from the inspection video footage and to draw a bounding box around them. The input image frame is run through the ResNet101 network to get the feature map. It then passes through the region proposal network and produces the output of bounding boxes for the anomalies and associated class labels in the output layer.

ResNet101,[17] Residual Neural Network, an efficiently trained network with 101 layers, won 1st place in the ILSVRC 2015[18] classification competition with top-5 error rate of 3.57%. It also won 1st place in the ILSVRC and COCO 2015[18] competition in ImageNet Detection, ImageNet localization, coco detection and coco segmentation. A relative improvement of 28% was observed by replacing the VGG-16[19] layers in Faster R-CNN with ResNet-101. When network depth increases, accuracy gets saturated and degrades rapidly. ResNet topology solves this degradation problem in deeper networks.

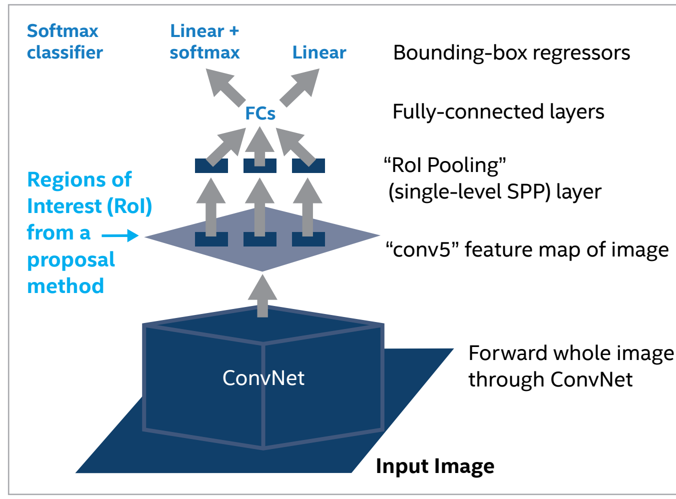

Figure 3. Fast R-CNN[10]

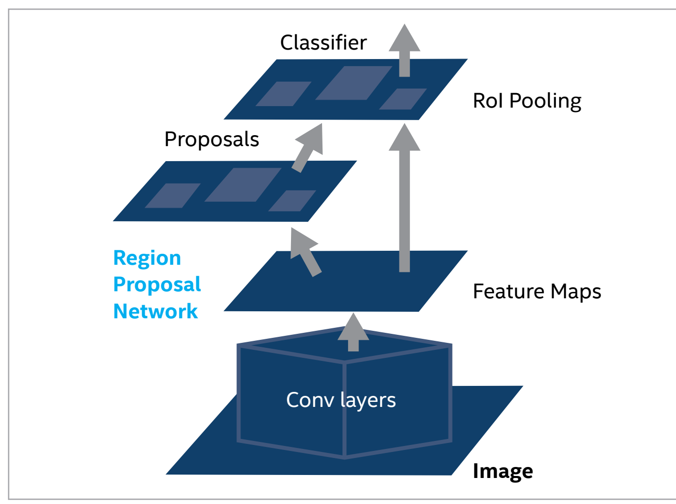

Figure 4. Faster R-CNN[10]

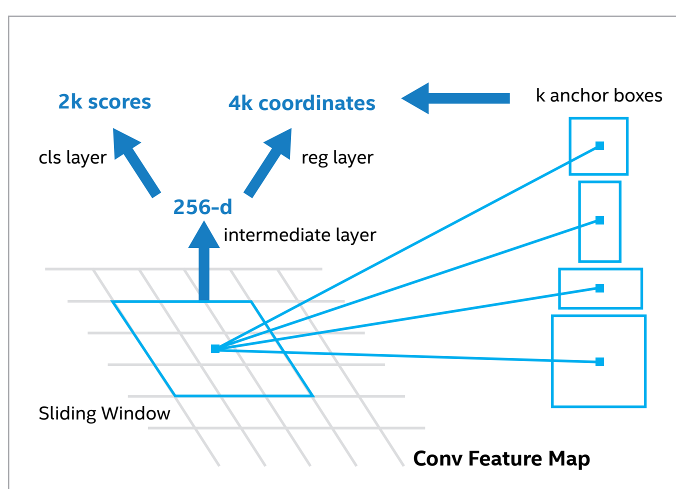

Figure 5. Region proposal network workflow[10]

Solution Approach

This solution approach follows six phases (Figure 6):

1. Get images with potential anomalies.
2. Annotate and label the dataset.
3. Convert dataset to the format needed by the TensorFlow framework.
4. Train the model with Faster R-CNN ResNet-101 architecture.
5. Compute evaluation metrics on the validation data set.
6. Optimize the model using OpenVINO-Model optimizer[20] for inference performance.

Dataset Acquisition

A good dataset is very important in order to train a model for high-quality. To collect such a dataset, assessment videos of sewer pipes with all targeted anomalies must be collected. Frames are extracted at a constant skip rate, for example 3 to 5 frames per second, and saved as image files. A sequence of computer vision algorithms was used to filter out frames that are not likely to contain anomalies.

Annotations and Labelling

Nearly 800 videos were used to extract 17,796 images, constituting all sub-categories of Cracks, such as Crack Longitudinal (CL), Crack Circumferential (CC), Crack Hinge (CH), Crack Spiral (CS) and Crack Multiple (CM). The images are 720x480 RGB image or 640x480 RGB image resolution. Table 1 lists the number of objects annotated of all sub-categories from 17,796 images. The solution is further enhanced with additional anomaly classes. For more information, refer to the Addendum below.

Name of Anomaly	Symbol	Number of Annotations
Crack Longitudinal	CL	14865
Crack Circumferential	CC	4041
Crack Hinge	CH	942
Crack Spiral	CS	342
Crack Multiple	CM	3846
Total		**24036**

Table 1. Annotations of Anomalies in Dataset

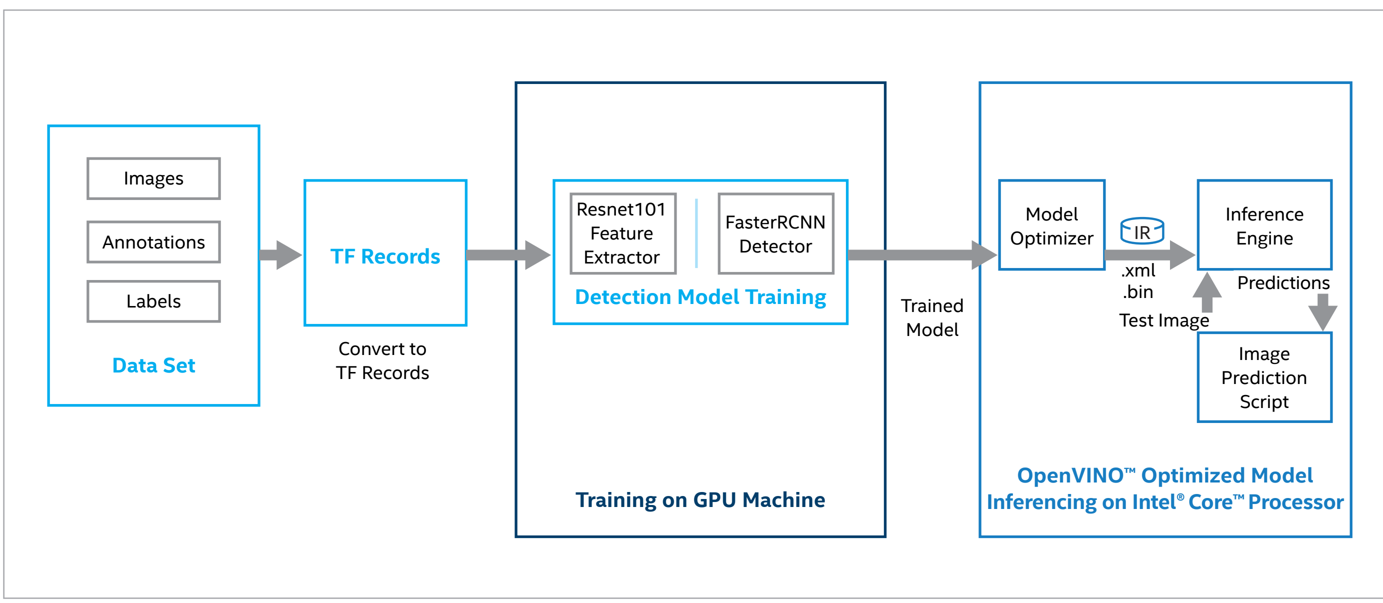

Figure 6. Workflow of Pipe Sleuth process

All images in the dataset were annotated with all subcategories by drawing bounding boxes around each anomaly found and labeled accordingly. The bounding box is defined as the minimal rectangle enclosing the whole anomaly, as shown in Figure 7. Multiple bounding boxes may be used for anomalies of larger sizes to cover most of the anomaly features, rather than covering the entire image to accommodate the anomaly object in a single bounding box. Annotation information for each image was stored in separate XML files.

Generating TF Records

In TensorFlow training framework, the data is read from its native binary format called, TFRecord. First the dataset is converted TFRecord binary file format, which consolidates all images and their respective labels into a single file. The TFRecord format serializes the data and annotations, and helps to reduce the time needed for data import and preprocessing by allowing TensorFlow to read the data linearly. This is especially helpful when the datasets that are too large to be held in memory and, only the data needed at the time of processing (e.g., a batch) is loaded from disk.

Training the Model

The total dataset was partitioned into 80% for training and 20% for testing. Within 80% training dataset, it was further partitioned into 70% for training and 30% for evaluation.

Transfer learning[21] approach was used for model training which uses pre-trained model developed for a similar or related task is used as the starting point for training. Weights were initialized from a Faster R-CNN ResNet101 model pre-trained on the COCO dataset[22]—a large-scale object detection, segmentation, and captioning dataset, followed by re-training all layers with the pipeline anomaly image dataset. As the dataset is completely different from the COCO dataset no layer parameters were frozen and all the layers were re-trained. Some model parameters, like the number of anomaly classes (which in this case is 5), image size (width and height), the number of steps for which training must be iterated, and learning rate were modified for training on the pipeline anomaly image dataset.

The initial learning rate was set to the default value of 0.000300000014249. Training was conducted up to 200,000 steps using a training dataset of 17,796 images. Various training parameters such as loss, accuracy, and mAP (mean Average Precision) were monitored during training. This was very important to avoid over fitting of the model.

After training, the model was converted into a frozen graph using TensorFlow's freeze_graph function, which was then used for inference.

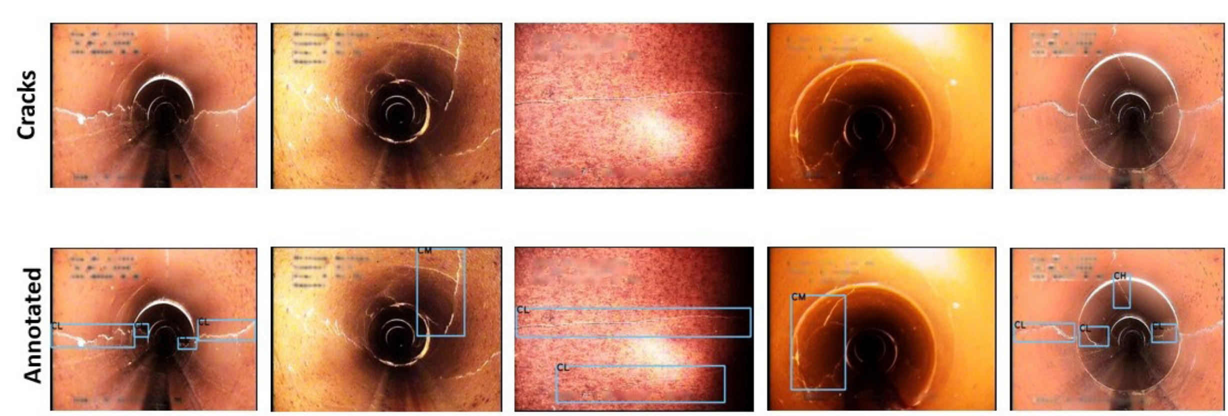

Figure 7. Sample Cracks and Annotated Images

Validating the Model

Wipro evaluated the model using the hold-out validation data set with respect to several performance metrics.

Model detection recall is calculated based on the below formula. Usually an object detection solution is evaluated based on its mean average precision value, but here, the focus is on calculating the percentage of the actual anomalies/cracks found against the total anomalies/cracks, and not the precision of the anomaly/crack detected. To calculate the detection recall, the Confusion Matrix is formed based on the predictions, which gives values of TP, TN, FP, and FN.

A True Positive (TP) is an outcome where the model correctly predicts the positive class. Similarly, a True Negative (TN) is an outcome where the model correctly predicts the negative class. A False Positive (FP) is an outcome where the model incorrectly predicts the positive class. A False Negative (FN) is an outcome where the model incorrectly predicts the negative class. Then Recall is calculated by the following formula. The model achieved ~84% of detection recall.

	Actual = Yes	Actual = No
Predicted = Yes	TP	FP
Predicted = No	FN	TN

$$\text{Recall} = \frac{TP}{TP + FN}$$

Optimization Using OpenVINO

Inference Optimization

Upon re-training of the model, it was optimized using Model Optimizer and used for inferencing on Intel® CPU architecture. Inference performance optimization[23] for the CPU was done using the Intel® Distribution of OpenVINO™ toolkit (version R4.420).

The Intel Distribution of OpenVINO toolkit is a comprehensive toolkit that can be used to develop and deploy vision-oriented solutions on Intel® platforms. Vision-oriented means the solutions use images or videos to perform specific tasks. It enables CNN-based deep learning inference on the edge to support heterogeneous execution across Intel® Vision Products using a common API for the Intel® processors, Intel® Integrated Graphics, Intel® Movidius™ Neural Compute Stick (NCS), Intel® Neural Compute Stick 2, Intel® Vision Accelerator Design with Intel® Movidius™ VPUs and Intel® FPGAs. It accelerates time-to-market through an easy-to-use library of computer vision (CV) functions and pre-optimized kernels.

Following is a summary of the steps for optimizing and deploying a trained model (Figure 8):

1. Save the trained model as a frozen graph, which can be used as input to the Model Optimizer.
2. Configure the Model Optimizer[24] and convert the frozen graph to IR format, an optimized Intermediate Representation of the trained model.
3. Test the model in the Intermediate Representation format with the test images using the Inference Engine in the target environment via provided Inference Engine validation application or sample applications.
4. Integrate the Inference Engine into the application to deploy the model in the target environment.

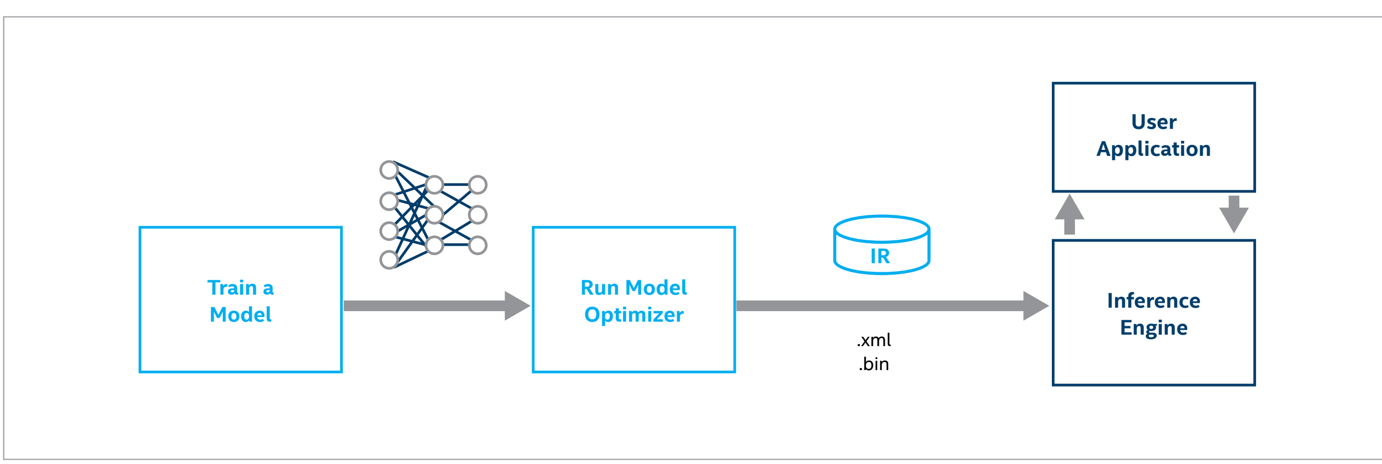

Figure 8. OpenVINO™ toolkit workflow for optimizing and deploying a trained deep learning model

The Model Optimizer is a cross-platform command line tool that facilitates the transition between the training and deployment environment. It performs static model analysis and simplifies the network topology by removing any operations not needed for inference, or by combining adjacent operations where possible. The result of this is that the optimized model is smaller and more efficient, decreasing inference time. It produces an Intermediate Representation (IR) of the network, contained in two files:

".xml": Describes the network topology.

".bin": Contains the weights and biases binary data.

The Inference Engine loads the IR stages it for inference and offers a unified API across supported Intel platforms. It enables deployment your network model trained with any of supported deep learning frameworks: Caffe, TensorFlow, MXNet, Kaldi, or ONNX.

For this use case, the Model Optimizer is used with the default optimization techniques[24] enabled, like Linear Operation Fusing, ResNet Optimization, etc., to produce the IR files. Then it was evaluated with new images for performance measurement. The average inference time taken for one image with non-optimized models and with OpenVINO-optimized models on various Intel processor-based platforms are shown in Figure 9. As you can see, inference time was improved by using the OpenVINO-optimized model. Wipro observed a reduction of up to 80% in inference time for Intel® Xeon® processors by using OpenVINO toolkit with no significant loss in model precision or accuracy.[26]

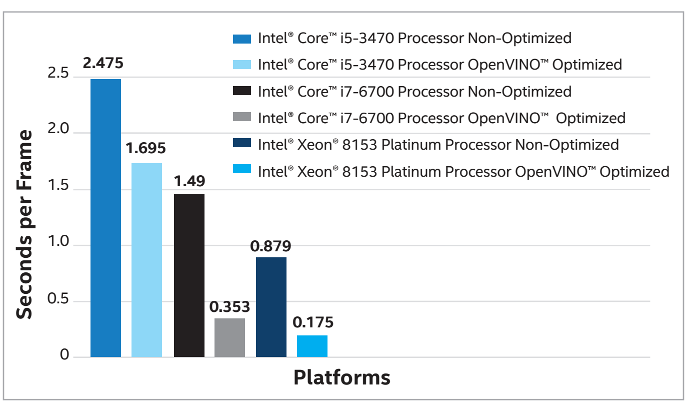

Figure 9. Performance metrics for OpenVINO™ optimized vs. Non-optimized inference on Intel® processors

Addendum

The solution and DL model are further enhanced and trained to include 8 more anomaly classes. It now supports detection of 13 different anomaly types.

Dataset Distribution

Table 2 below gives distribution of all annotated objects used in training and validation dataset for all anomaly types. These objects were created from a set of 26,600 images extracted from pipe inspection videos. Each of these images were annotated by drawing bounding boxes around the anomalies found (one or many) and creating corresponding label. The number of annotations is obviously more than the number of images as one image may contain more than one anomaly. Please refer to Figure 10.

Name of Anomaly	Symbol	Number of Annotations
Crack Longitudinal	CL	15407
Crack Circumferential	CC	4179
Crack Hinge	CH	994
Crack Spiral	CS	353
Crack Multiple	CM	4039
Deposit Attached Encrustation	DAE	1899
Deposit Attached Grease	DAGS	1899
Deposit Attached Others	DAZ	1934
Deposit Settled Others	DSZ	1225
Root Ball	RB	564
Root Medium	RM	1415
Root Fine	RF	1802
Collapse	X	361
Total		**36071**

Table 2. Annotations of Anomalies in Dataset

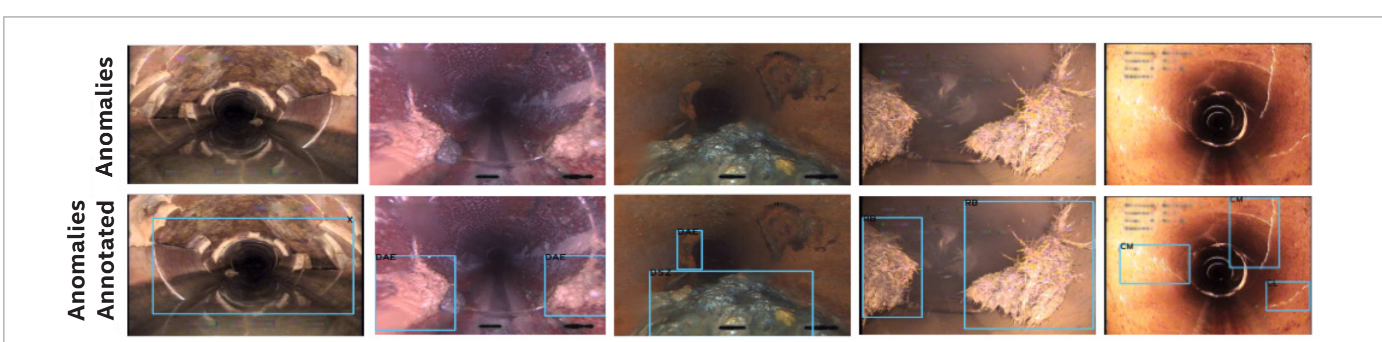

Figure 10. Sample Anomalies and Annotated Images

Model Training

The dataset was randomly partitioned (80% for training and 20% for testing) to train the model and TF records were generated for it (see "Generating TF Records" section). A pre-trained weights of COCO2015 dataset was used as a starting point. The initial learning rate was set to the default value of 0.000300000014249 and the momentum optimizer was used to train. Training was conducted up to 300,000 steps using a training dataset (80% of 26,600 images i.e., 21,280 images). Various training parameters such as loss, accuracy, and mAP (mean Average Precision) were monitored during training. This was very important to avoid over fitting of the model.

Model Evaluation

The model was tested using the hold-out test dataset (20% of 26,600 i.e., 5,320 images). Model detection recall is calculated based on the formula shown in the "Validating the Model" section. The model achieved a Recall score of 80.11% on the test dataset.

Model Inference

The trained model was converted to a TensorFlow frozen graph. It was then optimized using OpenVINO (see "Inference Optimization" section) for inference, and the inference performance results were captured on various Intel® platforms (Figure 11).

Conclusion

The objective of this work was to enable CNN-based deep learning inferencing on edge devices with Intel CPUs, for pipeline condition assessment. From the performance results in Figure 11, it is clear that automated pipe condition assessment from pipeline inspection videos can be performed on various Intel processor-based platforms with acceptable performance.

Based on this experiment, optimizing the deep learning model using OpenVINO for Intel architecture-specific hardware results in real-time performance for this use case (i.e., about two to three frames per second for inferencing/evaluation).

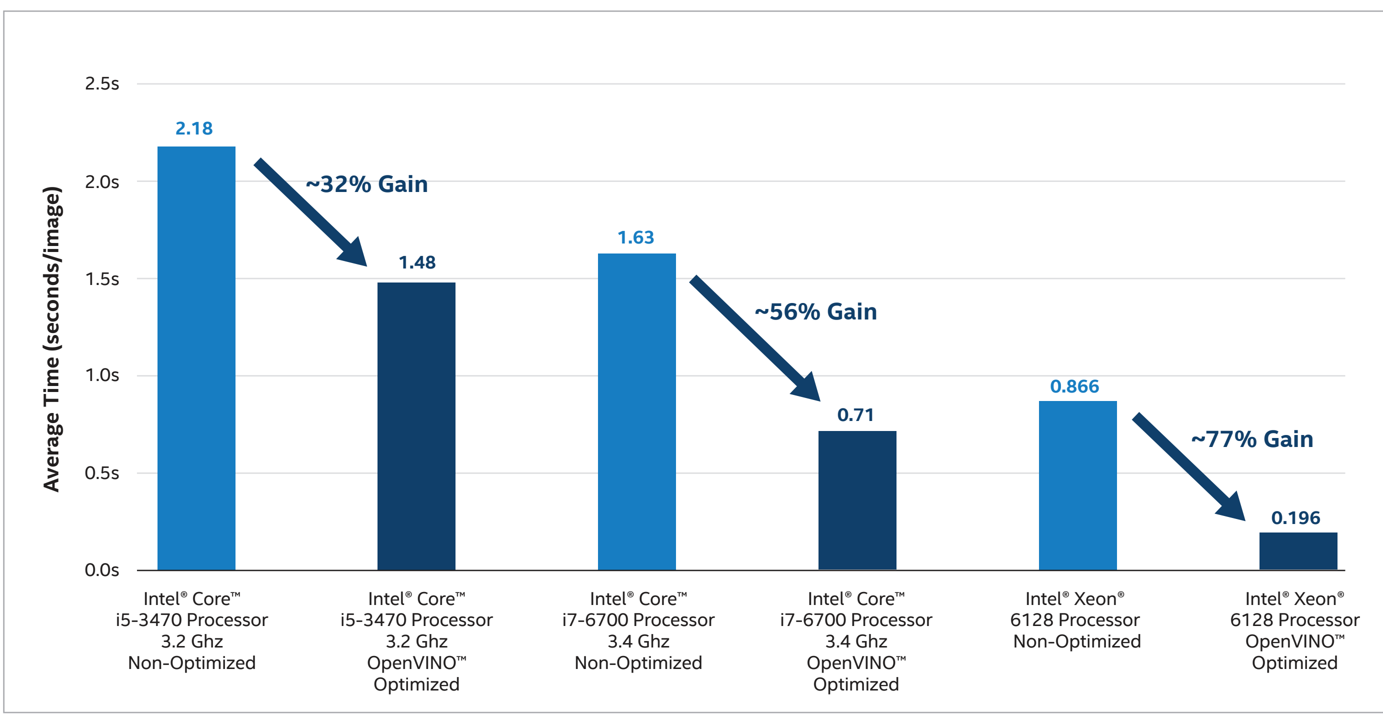

Figure 11. Performance metrics for OpenVINO™ optimized vs. Non-optimized inference on Intel® processors

Authors

Kuljeet Singh has worked extensively in the IoT, mobility and embedded systems domains for 17 years. Presently a Solutions Architect (IoT & AI) with Wipro's Industrial and Engineering Services division, he is working on the development and deployment of AI/ML based solutions for IoT domain. He is also working with the technology teams of a leading semiconductor customer on the next generation of IoT solutions. For more information, contact him at kuljeet.singh@wipro.com.

Sundar Varadarajan is an industry expert in the field of analytics, machine learning and AI, having ideated, architected and implemented innovative AI solutions across various industry verticals. Currently, Sundar is a Consulting Partner at Wipro on AI/ML, and plays an advisory role on various leading edge AI and Machine Learning solutions. Sundar can be reached at sundar.varadarajan@wipro.com.

Subin Guruvayurappan is a Software Engineer at Wipro with 3.5 years of experience in various domains like machine learning, deep learning, computer vision, DevOps and MEAN stack development. He is also interested in AI/ML and IoT. For more information, contact him at subin.guruvayurappan@wipro.com.

Anubhav Anand is a Software Engineer at Wipro who is involved in the development of applications in Predictive Analytics, Computer Vision and NLP. He is interested in building the AI-powered SaaS products. For more information, contact him at anubhav.anand1@wipro.com.

Saravanan Solaiyappan has worked extensively in the Mobility, CommDSP, DSP&Multimedia and Embedded Systems domains for 17 years. Presently an Architect with Wipro's Industrial and Engineering Services division, he is working on the development and deployment of AI/ML based projects. He is the architect for the PipeSleuth product for detection and classification of anomalies in the pipeline assessment videos. For more information, contact him at saravanan.solaiyappan@wipro.com.

Rupesh Kumbhare is Director of System Software & DSP engineering group at Wipro Ltd. He has over 21 years of industry experience in developing and leading products in the domain of Digital Signal Processing, Embedded Software, Multimedia, Voice over IP and Wireless. He is leading the product development of Pipe Sleuth solution. In the last 3 years his focus areas are Artificial Intelligence, Computer Vision and Machine Learning at Edge and 5G physical layers. For more information, contact him at rupesh.kumbhare@wipro.com.

INTEL®
AI BUILDERS
MEMBER

Wipro is a member of the **Intel® AI Builders Program**, an ecosystem of industry-leading independent software vendors (ISVs), system integrators (SIs), original equipment manufacturers (OEMs), and enterprise end users, which have a shared mission to accelerate the adoption of artificial intelligence across Intel® platforms.

1. https://arxiv.org/abs/1506.01497
2. https://towardsdatascience.com/fasterrcnn-explained-part-1-with-code-599c16568cff
3. https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8
4. https://www.tensorflow.org/deploy/distributed
5. https://software.intel.com/en-us/openvino-toolkit
6. https://www.nassco.org/content/pipeline-assessment-pacp
7. https://www.nassco.org/sites/default/files/pacpcodechart_0.pdf
8. https://en.wikipedia.org/wiki/Deep_learning
9. https://en.wikipedia.org/wiki/Convolutional_neural_network
10. https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/
11. http://vision.stanford.edu/teaching/cs231b_spring1415/slides/ssearch_schuyler.pdf
12. http://www.robots.ox.ac.uk/~tvg/publications/talks/fast-rcnn-slides.pdf
13. http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf
14. https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/
15. https://www.pyimagesearch.com/2016/09/12/softmax-classifiers-explained/
16. ResNet - https://arxiv.org/pdf/1512.03385.pdf
17. https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624
18. http://image-net.org/challenges/ilsvrc+mscoco2015
19. https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
20. https://software.intel.com/en-us/articles/OpenVINO-ModelOptimizer
21. https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/
22. COCO Dataset
23. http://docs.openvinotoolkit.org/2019_R1/_docs_MO_DG_prepare_model_Model_Optimization_Techniques.html
24. https://software.intel.com/en-us/articles/OpenVINO-Inference-Engine-Optimization-Guide
25. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173
26. https://software.intel.com/en-us/articles/boosting-deep-learning-training-inference-performance-on-xeon-and-xeon-phi
27. https://docs.microsoft.com/en-us/cognitive-toolkit/object-detection-using-faster-r-cnn
28. https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture

0819/BA/HBD/PDF ♲ Please Recycle 340931-002US