Paper SAS4432-2020

Bringing Computer Vision to the Edge: An Overview of Real-Time Image Analytics with SAS®

Maggie Du, Juthika Khargharia, Shunping Huang, and Xunlei Wu, SAS Institute Inc.

ABSTRACT

In the Internet of Things (IoT) era, when large amounts of streaming data are generated continuously, it is extremely impractical and inefficient to store all these data in a data center. Furthermore, the majority of these data are irrelevant; for example, only streaming data that contain anomaly events are worth storing or transmitting for further investigation. For real-time image or video processing, moving the analytics to edge devices not only saves a device-to-cloud data round trip but also improves data privacy and governance.

This paper presents the real-time image analytics solutions offered in SAS® software for image processing, image classification, object detection, and segmentation. It also describes the general workflow of real-time image analytics, from preprocessing images, to training deep learning models by using SAS® Viya®, to deploying an image analytics pipeline on edge devices by using SAS® Event Stream Processing. The paper discusses the following applications: real-time semantic segmentation analysis, with an example of autonomous driving; real-time defect detection for quality inspection in the manufacturing industry specific to surface mount technology (SMT); and loose ballast detection in railway tracks for monitoring track health in the transportation industry.

INTRODUCTION

With the massive amount of streaming data being generated and processed every day, edge computing has become an exciting facet of IoT. Edge computing helps break the limits of cloud computing, particularly when dealing with computer vision. In this paper, three examples demonstrate what computer vision is, why it is important to process computer vision on the edge, and the general workflow of edge computing supported.

OVERVIEW OF COMPUTER VISION

Computer vision is the area of computer science that enables computers to understand the visual world through digital images and videos. It is the process of understanding images and videos and reacting appropriately, in the same way that human vision does. This technology has been widely applied to autonomous driving, production line automation, facial recognition, medical diagnostics, agriculture intelligence, and more. In these applications, computers are trained to achieve one or several of the following basic tasks, which are illustrated in Figure 1:

- *Image classification* refers to the task of labeling an image as belonging to one of several predefined categories, based on the main content in the image. The image in Figure 1 could be classified into categories such as animal, dog, or cat—this is typically how image classification works.
- *Keypoints detection* involves detecting multiple interest points in an image simultaneously, such as facial landmark detection. In Figure 1, the eyes, ears, and noses of the dog and cat are detected.

- *Object detection* can achieve classification and localization of different objects in an image at the same time. Object detection finds the objects of interest and draws a bounding box around each object, so you know which object it is and where it is located.
- *Semantic segmentation* classifies each pixel of an image into one of the predefined categories, creating a mask image that shows the exact boundaries of each object. Figure 1 has three semantic categories: dog, cat, and background. In semantic segmentation, each pixel is assigned to one of these three categories, so you get the exact boundaries of the dog and the cat. In addition, *instance segmentation* differentiates pixels that belong to different instances of the same object type.

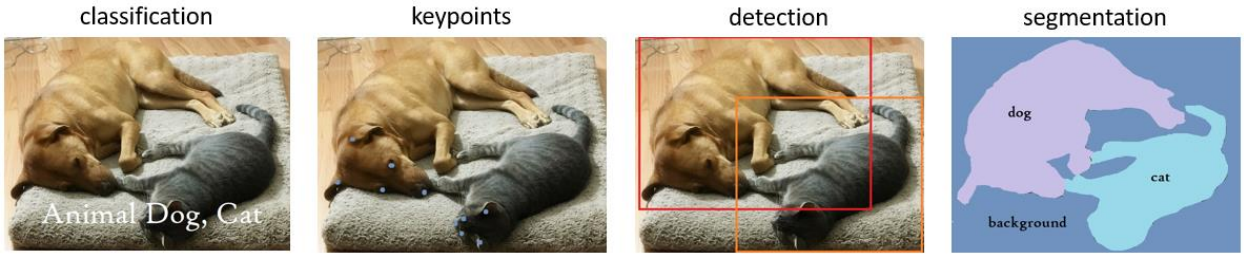


Figure 1. Basic Tasks of Computer Vision

COMPUTER VISION ON THE EDGE

Edge computing is recognized as "a part of a distributed computing topology in which information processing is located close the edge—where things and people produce or consume that information" (Gartner Glossary). Unlike cloud-based computing, where a set of images are stored in a SAS data set and are scored in batches, edge computing allows a stream of images from camera or other input buffers to be scored through SAS Event Stream Processing. Edge computing is advantageous in three aspects:

- Speed: The strongest driving force for edge computing is its speed. Without edge computing, an autonomous car would need to scan the road using local cameras, send the images to a cloud data center for analysis, and then receive the computed data from the cloud for display. Completing that entire process would take a considerable amount of time, whereas edge computing can reduce latency by fulfilling all the steps on the car's computer. The processing algorithm runs locally and saves the device-to-cloud round trip, thus making it possible to build more responsive applications that can achieve real-time reactions by avoiding data transfer.
- Security and privacy: Edge computing improves security by reducing the distance data has to travel for storing and processing, thus lowering the risk of hackers intercepting the data during transmission. In addition, storing all the data in a data center makes the data center especially vulnerable to any kind of attack. Edge devices can enforce security at the device level so that there would be fewer attacks on the cloud server. Keeping sensitive data only on the edge devices instead of on cloud servers also increases data privacy.
- Cost-effectiveness: With massive amounts of data generated and processed each day, it is not practical to constantly build or upgrade data centers for data storage and transfer. In fact, it is nearly impossible to store all data that are generated continuously nowadays, especially high-dimensional data such as images and videos. Furthermore, most of the data are completely irrelevant and only a small portion is worth storing. The development of edge devices makes things much easier. After

data stream in and are analyzed, they can simply be discarded except for data that trigger an alarm.

GENERAL WORKFLOW OF EDGE COMPUTING

This section describes the general SAS workflow of computer vision on the edge, as summarized in Figure 2. Generally, the whole process consists of two parts: model training by using batches on the server side with SAS Viya, and edge computing with streaming data on edge devices with SAS Event Stream Processing.

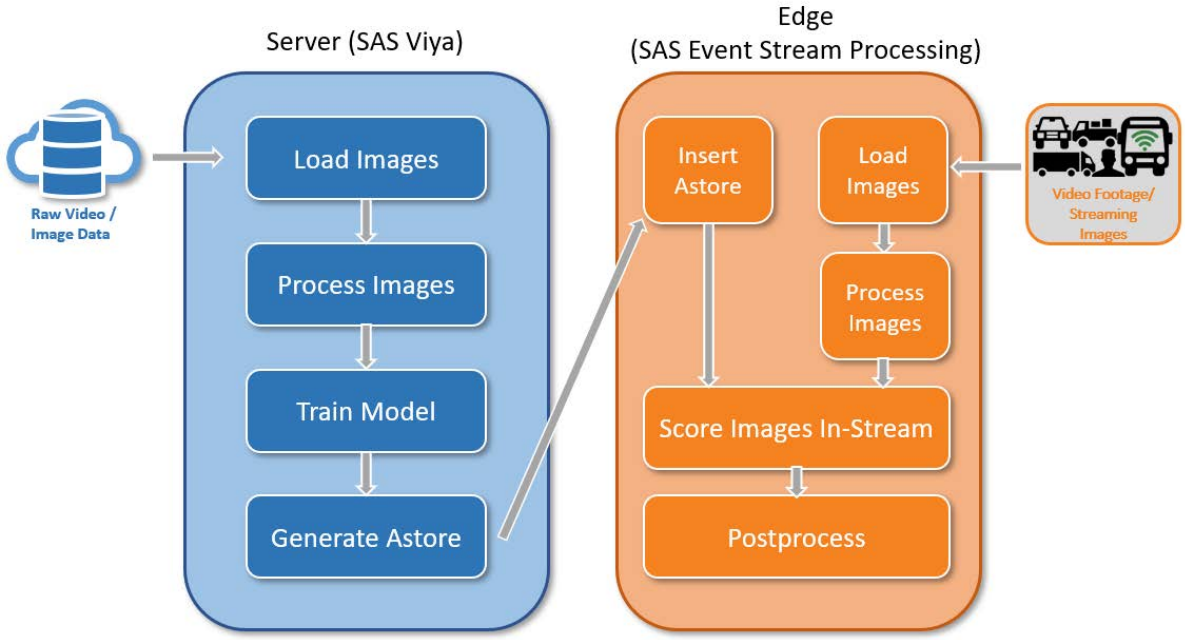


Figure 2. Computer Vision Architecture

SERVER-SIDE TRAINING WITH SAS VIYA

Models are trained and tested on servers by using SAS® Visual Data Mining and Machine Learning before they are deployed onto SAS Event Stream Processing for scoring. You can either use the functionality of the Image and Deep Learning action sets in SAS Visual Data Mining and Machine Learning, or use DLPy, which is an open-source, high-level Python package for deep learning that you can use to construct the model. For more information about DLPy, see https://github.com/sassoftware/python-dlpy.

In most cases, the training process includes the following four steps:

1. Load images from raw files (such as .png or .tif image files) to create the training SAS data set. This could be done by using the `image.loadImages()` action in SAS Visual Data Mining and Machine Learning or by using the DLPy API `ImageTable.load_files()`.
2. Process the images that were used for training. This might include image resizing, cropping, flipping, mutating, and possible image augmentation steps. These steps can be achieved by calling different image functions in the `image.processImages()`action.
3. Build a deep learning model by using the `addLayer()` action or DLPy sequential APIs. Train and test the model by using the data set that was created in steps 1 and 2.
4. Generate an analytic store (astore), which is a file that contains model and weights information to be used for deployment, by using the `dlexportmodel()` action or DLPy

API `model.deploy()`.

An analytic store is a binary file that stores information about the trained model and that can be transported from one platform to another. Therefore, a model that is trained on SAS Visual Data Mining and Machine Learning is portable to any edge device when it's saved as an astore, and it can be used to score new images.

EDGE DEPLOYMENT WITH SAS EVENT STREAM PROCESSING

Once the model is trained and tested on SAS Visual Data Mining and Machine Learning, it can be readily deployed to SAS Event Stream Processing through the generated astore. You can use an XML file to define a model that contains the necessary windows and edges between windows. A typical model contains the following windows:

1. A Source window takes in the input images through a connector or adaptor. Images or frames of videos are converted to Base64 format and then published to the Source window. If the next window requires a blob as the input, the encoded images are automatically decoded as binary formats in memory.
2. An Image Processing window is used when the published images need some preprocessing before they can be sent to the Score window. For example, if the provided images are larger than what the astore model requires, then an image-resizing window would be needed to resize the images to the designated dimensions.
3. A Model Reader window reads in the astore that was generated during training and provides the model and associated parameter weights to the Score window.
4. The Score window runs the model and scores the incoming images. An external client such as Python or ESPPy can subscribe to this window for displaying results and postprocessing them. For more information about ESPPy, see https://sassoftware.github.io/python-esppy/.

APPLICATIONS

This section describes three different edge computing applications that use SAS Event Stream Processing to achieve real-time image analytics.

AUTONOMOUS DRIVING WITH REAL-TIME IMAGE SEGMENTATION

This example trains a lightweight semantic segmentation model that uses labeled street scene images and potentially could be deployed to vehicle cameras and sensors (Paszke et al. 2016). It demonstrates how to perform real-time semantic segmentation by using street scene images that are generated by the CARLA car simulator (Dosovitskiv et al. 2017). CARLA provides RGB (red, green, blue) images and labeled mask images that can be used to train models that can be applied to autonomous vehicles. It is essential for the self-driving cars to segment objects on the street (such as other vehicles, pedestrians, road lines, and so on) so that the car can follow the roads and avoid pedestrians and other vehicles. The segmentation model also has to be implemented in a real-time manner, because any latency could possible lead to unpredictable outcomes.

Data Overview

The training set contains 4,800 color images that are all resized to 512×512. The mask images are of the same dimension, with pixels labeled as belonging to one of 13 predefined categories. Figure 3 shows four sample raw images and corresponding mask images, where a colormap is applied to mask images for better visualization. The predefined classes are shown in Table 1. Objects that belong to the same category are marked in the same color; for example, all pixels of vehicles (including the dashboard of the driving one) are in medium blue and pixels of roads are in green.

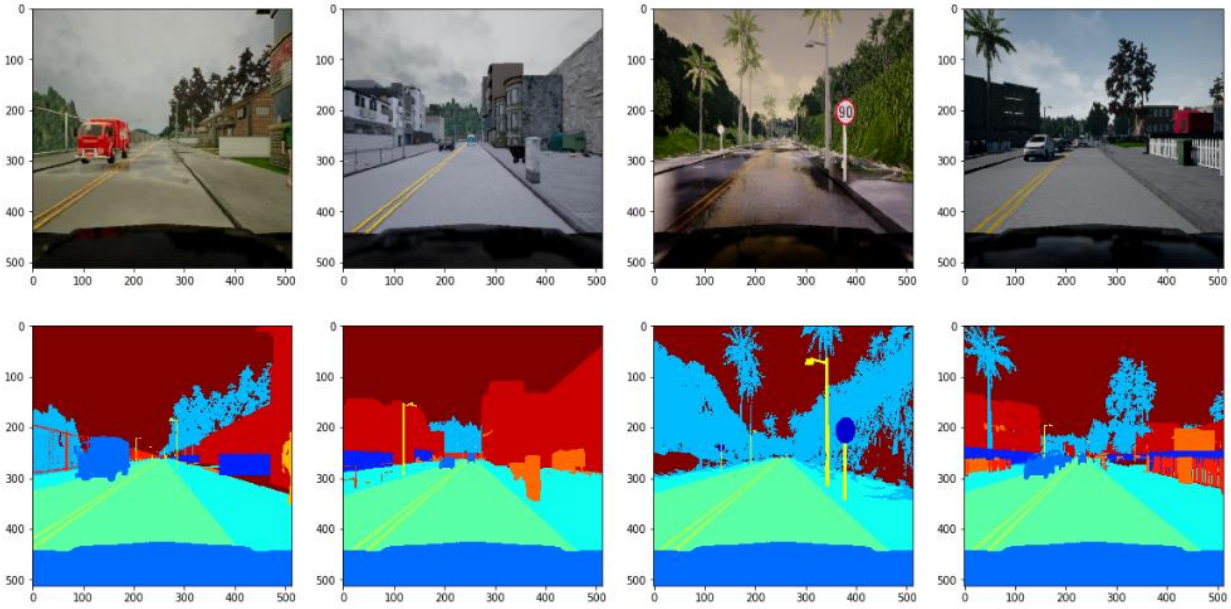


Figure 3. Training Data Visualization (raw images in the top row and ground truth masks in the bottom row)

Value	Label
0	Unlabeled
1	Building
2	Fence
3	Other
4	Pedestrian
5	Pole
6	Road line
7	Road
8	Sidewalk
9	Vegetation
10	Car
11	Wall
12	Traffic sign

Table 1. Predefined Categories

Model Architecture

The deep learning model architecture for this application is based on EfficientNet (ENet). The architecture can be divided into several stages, and a diagram of each stage is shown in Figure 4. The initial block contains an input layer, followed by a 3×3 convolution layer with stride 2 and a max-pooling layer, followed by a concatenation layer. In the downsampling bottleneck module, there is a 3×3 convolution layer with stride 2 to decrease the feature size, and an extra max-pooling layer followed by a 1×1 expansion. In the upsampling and regular bottleneck modules, a 1×1 projection is used to reduce the dimensionality. Then, the main convolution layer or transpose convolution layer (denoted by Tconv in Figure 4) is

followed by another 1 × 1 expansion. In all modules, each convolutional layer is followed by a batch normalization layer (not shown).

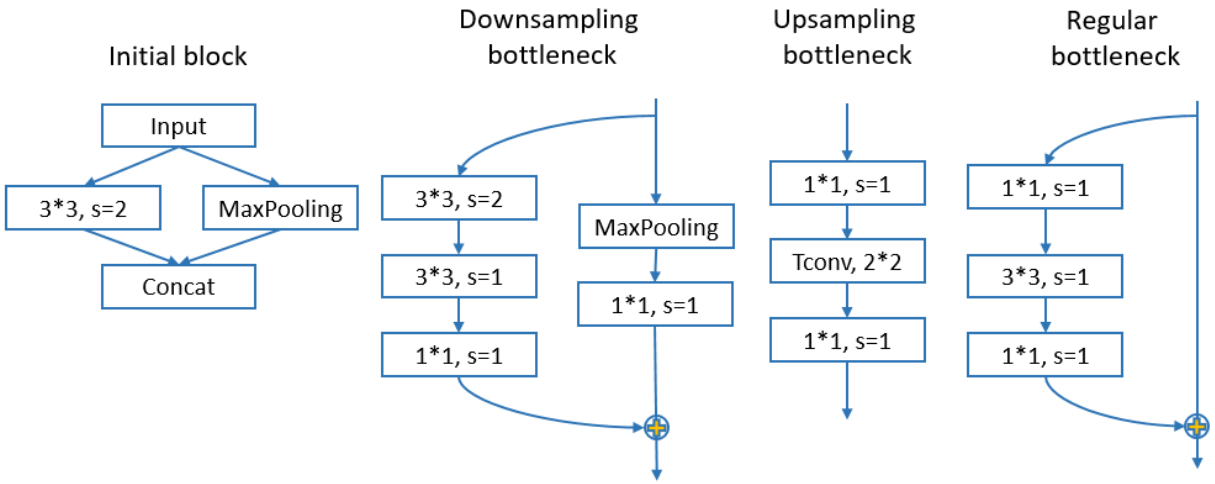


Figure 4. Diagram of Each Module

The architecture is shown in Table 2. With input images of 512 × 512, the initial block and the bottlenecks in Table 2 quickly downsample the feature size to 64 × 64. Strong downsampling is avoided because reduced resolution hurts prediction accuracy, and strong upsampling increases computational cost. The last convolution layer adjusts the number of channels to match the number of categories (13) in the data set.

Stage	Name	Type	Output Size
0	Initial	Initial	256 × 256 × 16
1	BNeck1.0	Downsampling	128 × 128 × 64
	BNeck1.1–BNeck1.4	Regular	128 × 128 × 64
2	BNeck2.0	Downsampling	64 × 64 × 128
	BNeck2.1–BNeck2.4	Regular	64 × 64 × 128
3	BNeck3.1–BNeck3.4	Regular	64 × 64 × 128
4	BNeck4.0	Upsampling	128 × 128 × 64
	BNeck4.1–BNeck4.2	Regular	128 × 128 × 64
5	BNeck5.0	Upsampling	256 × 256 × 16
	BNeck5.1	Regular	256 × 256 × 16
6	BNeck6.0	Upsampling	512 × 512 × 16
	Conv	Convolution	512 × 512 × 13

Table 2. Model Architecture

This is a lightweight model for semantic segmentation, with only 0.2M parameters and 1.88 GFLOPS (1.88 billion floating point operations). The model in SAS Event Stream Processing contains three windows: a Source window with a connector through which images stream in; a Model Reader window, which reads the astore file; and the Score window, which

performs real-time scoring. The overall workflow (shown in Figure 5) can be processed entirely on edge devices, with the Source, Model Reader, and Score windows running in SAS Event Stream Processing and the Colormap and Display steps running directly on the device. Basically, SAS Event Stream Processing reads a new image from cameras through an adaptor and scores by using the semantic segmentation model in the astore. Then a Python client that subscribes to the Score window applies a colormap to the mask image and displays it for visualization.

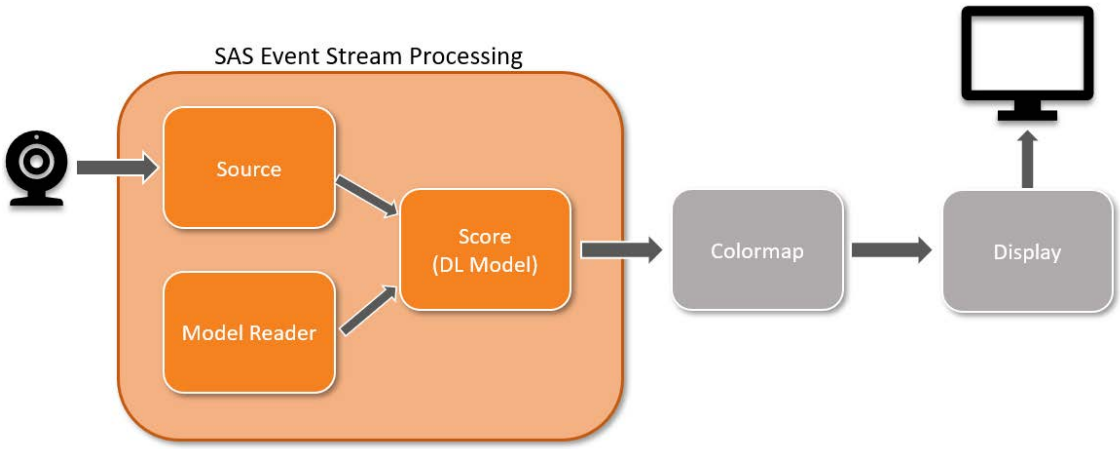


Figure 5. Scoring Flow on Edge Device

Performance on Edge Devices

The test images are similar street scene images that are not used for training. The pixel accuracy on test images is 92.1%, meaning that less than 8% of pixels are misclassified. For the most important categories (road and car), the accuracies are 94.2% and 97.1%, respectively. Figure 6 shows the comparison among raw images (first row), ground truth images (second row), and predicted images (third row).

Table 3 reports the computing power and scoring FPS (frame-per-second) on different NVIDIA devices. Scoring achieves 8 FPS on a Jetson TX2 and is adequate for road scene applications.

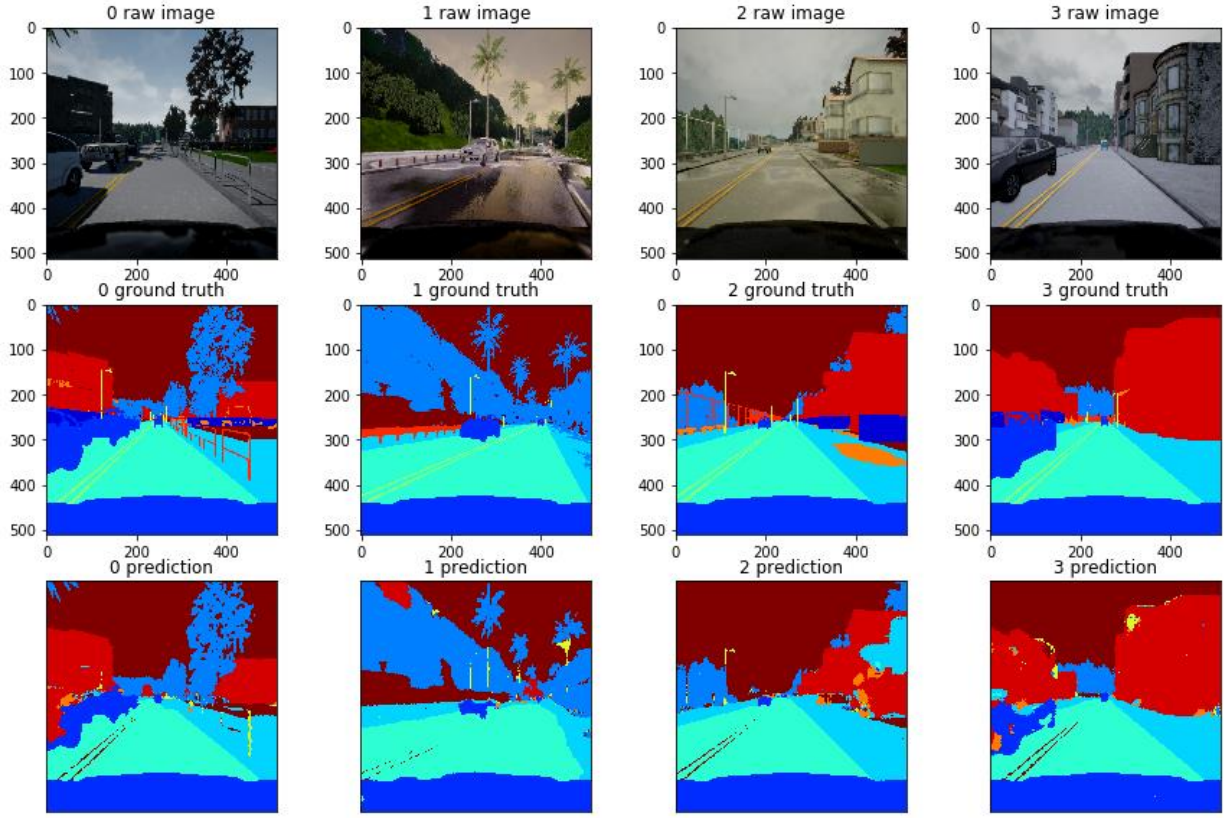


Figure 6. Prediction Using Test Images (input images in the top row, ground truth masks in the middle row, and predicted images in the bottom row)

	Tesla V100 (cloud)	Tesla T4 (cloud)	Jetson AGX Xavier (edge)	Jetson TX2 (edge)
TFLOPS	100	8	11	1.5
FPS	34	18	19	8

Table 3. Frames per Second of the Same Model on Different Devices

DEFECT DETECTION IN SURFACE MOUNT DEVICES

This example detects defects in the manufacturing industry with a pretrained VGG16 model (Simonyan and Zisserman 2015). Surface mount technology (SMT) inspection machines such as advanced optical inspection (AOI) and advanced X-ray inspection (AXI) machines are often used for quality inspection of printed circuit boards (PCBs). AOI machines inspect visually available components such as missing or skewed components in PCBs, and AXI machines can look at defects that result from solder joints. In addition to providing images of component parts or solder joints, these machines also provide several measurements of the joints, such as diameter, thickness, eccentricity, and so on. These measurements can be used as additional inputs along with the computer vision models for defect classification. One application of computer vision technology is in the detection of head-in-pillow (HiP) defects from AXI machines. Head-in-pillow is an assembly defect in which the bumps from a ball grid array (BGA) don't coalesce with the solder paste on the PCB pad. Figure 7 compares a HiP joint with a good joint. It appears that for a HiP defect the solder has melted but has not joined together. A HiP defect can be caused by several factors such as

surface oxidation, poor wetting of solder, or distortion of the integrated circuit package or circuit board by the heat of soldering process.

Figure 7. Head-in-Pillow Defect Compared with a Good Solder Joint

Figure 8 (left) presents a sample image taken from an AXI machine that shows different joints. The joint enclosed in the red square in Figure 8 represents a HiP-defective joint. Figure 8 (right) shows the training sample, which consists of two defects and two nondefects after they were cropped and resized to 224 × 224. To the ordinary eye, it is impossible to tell the difference between a defect and a nondefect.

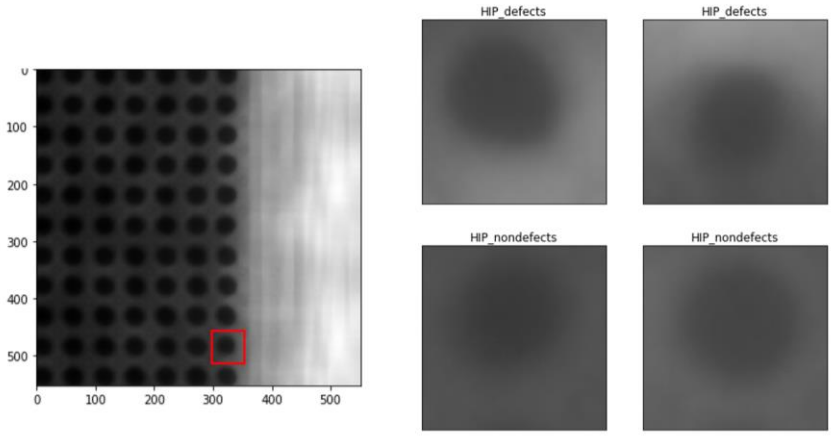


Figure 8 (Left) Defective Joint Shown in the Enclosed Red Box. (Right) Cropped Training Samples That Consist of Two Defective and Two Nondefective Joints

In the next step, several image preprocessing techniques were applied to the data to reveal interesting features in the joints. Figure 9 shows the application of histogram equalization on the image data. Histogram equalization is typically used to improve contrast in images by effectively spreading out the intensity range of the pixels over the entire image. In Figure 9, differences start to emerge as the defective data are compared with the nondefective data. In addition, other techniques such as image comparison were applied between samples of defective and nondefective data to compute a self-similarity index between the images. All these methods pointed to inherent differences between the defects and nondefects that are not visible to the human eye from the raw images alone.

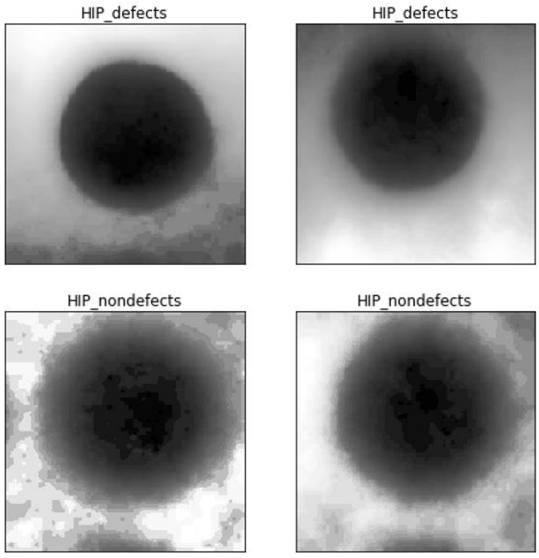


Figure 9. Histogram Equalization Applied to the Raw Image Data

The ground truth for labeling the defects was obtained from manual measurements (such as diameter, thickness, eccentricity, and so on) that were performed on individual joints by the AXI machines. The objective is to determine whether training deep learning models on the image data can sufficiently classify HiP defects without the time-consuming effort of taking additional measurements on the joints. Because the number of defects were relatively small (less than 5% of the total data), techniques of image augmentation were applied to increase the size of the training data set. Next, a convolutional neural network (CNN)—specifically, a predefined VGG16 model architecture with pretrained weights—was used to detect and classify the HiP defects in the images. This following SAS DLPy code uses a pre-defined VGG16 model architecture named `model_vgg16` with pretrained weights to detect and classify HiP defects from nondefects.

```
model_vgg16 = VGG16(
    conn, model_table='VGG16_notop',
    scale=1, random_flip='HV', n_channels=1,
    width=224, height=224, n_classes=2, offsets=[80.0, 80.0, 80.0],
    pre_trained_weights=True, include_top=False,
    pre_trained_weights_file='vgg16_hip_new.sashdat')
```

A low misclassification error (less than 10%) was achieved on the validation data set over several trials, indicating that deep learning models for classification of HiP defects can indeed work very well for this type of scenario. This will significantly reduce the time spent in making manual measurements of the joints, thereby augmenting human effort. Table 4 shows the confusion matrix, which indicates that of the 138 total images in the validation data set, 100% of the defects and 91% of the nondefects were correctly classified.

Ground Truth labels	Predicted Labels	
	HiP_defects	HiP_nondefects
HIP_defects	48.0	0.0

HIP_nondefects	8.0	82.0

Table 4. Confusion Matrix Showing Distribution of HiP Classification on the Validation Images

SAS DLPy can be easily used to visually inspect images that were correctly and incorrectly classified. Figure 10 shows an example of a joint that was classified as a HiP defect with 91.38% accuracy.

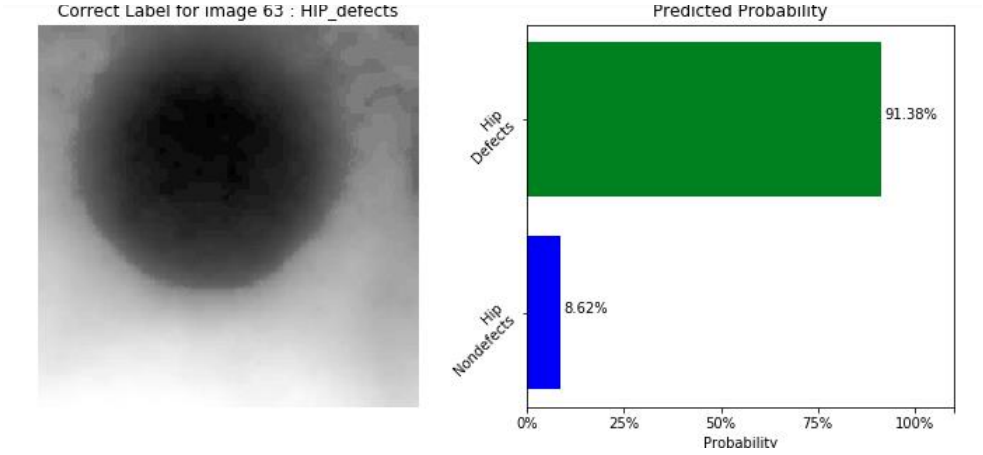


Figure 10. Scoring Hold-Out Data Using a Trained VGG-16 Model That Shows Correct Classification of a HiP Defect

There are several advantages of deploying models that can process image and video streams in real time on an edge device. In the context of HiP defect detection with AXI machines, the ideal deployment scenario is one where the trained model can score new data on the machine itself. Real-time analytics using computer vision also has the benefit of augmenting human effort by running side-by-side with skilled workers. Thus, you can take advantage of a SAS Event Stream Processing engine deployed on the AXI machine itself. In order to achieve that, generate the model astore from the trained deep learning model by using the following code:

```
model_vgg16.deploy(path='<path>', output_format='ASTORE')
```

Figure 11 shows how a Model Reader window (model_reader) in SAS Event Stream Processing Studio receives requests from a Request window (w_request), uses the request information to fetch the specified model, and publishes the model event to the Score window (score) for scoring. As new images are scored successfully, they can become part of the training data. Offline models can be retrained on a regular basis to improve model accuracy and robustness.

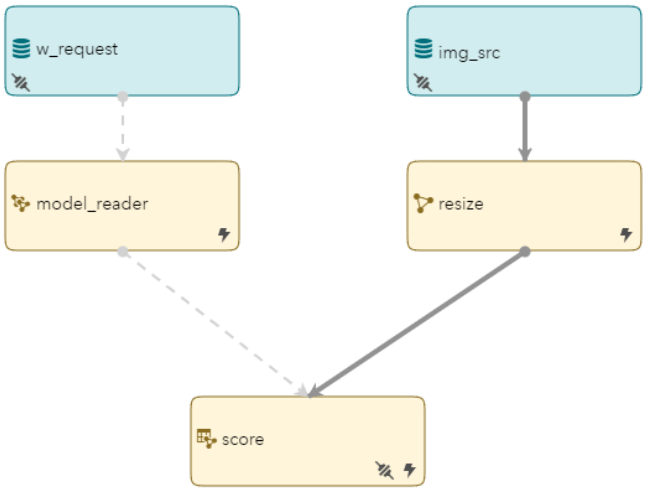


Figure 11. Process Flow Showing Input of an Analytic Store and Scoring New Data Using SAS Event Stream Processing Studio

LOOSE BALLAST DETECTION FOR INSPECING TRACK HEALTH

Have you ever looked at railroad tracks and wondered why they are covered with jagged little stones? The stones are called track ballast, and their purpose is to keep the tracks in place, providing protection from different weather conditions, vibrations, ground movement, and weed growth that could render the tracks unstable over time. Insufficient gravel underneath the tracks leads to a condition called loose ballast, which can be a risk to stable operation of trains. Figure 12 compares normal ballast and loose ballast conditions. This example uses computer vision to detect loose ballast conditions in real time. It uses a pretrained Resnet-50 model (He et al. 2015).

Figure 12. Normal Ballast (left). Loose Ballast (right) Can Create Safety Issues for Trains.

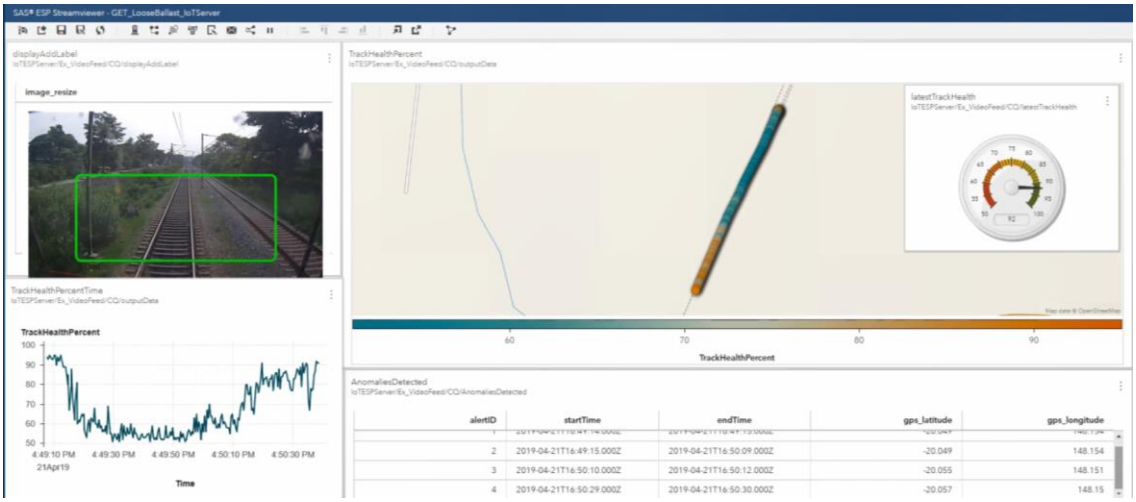
Typically, trains have a camera mounted to the front that can capture high-definition video of the railway tracks. Being able to run SAS Event Stream Processing on an edge device (a camera that has enough computing power) and store the raw video feeds along with additional information such as geolocation, speed, and other telematics data enable real-time analytics on the edge. This particular use case follows the two-phased approach that is shown in Figure 2. A server-side (offline) analysis is performed in SAS Visual Data Mining and Machine Learning; this analysis trains a deep learning model to classify loose ballast

from a normal ballast condition. For the edge deployment (online analysis), the analytic store from the deep learning model is pushed into SAS Event Stream Processing for real-time scoring of new images as they are being collected by the on-board camera system. The loose ballast classification score is then combined with other metadata and telematics data to compute a robust track health score that can be monitored in real time.

For the model training phase, a predefined RESENET50 Caffe model architecture with pre-trained weights was used to detect and classify loose ballast in the images. The images were resized to 224 × 224. The following code uses SAS DLPy to create a Resnet50 Caffe model architecture named `model_ResNet50` that has two classes in the final prediction layer:

```
model_ResNet50 = ResNet50_Caffe(
                 conn, model_table='RESNET50_CAFFE', n_channels=3,
                 pre_trained_weights_file='ref_weights_resnet.sashdat',
                 pre_trained_weights=True,
                 width=224, height=224, offsets=tr_img.channel_means,
                 include_top=False, n_classes=2)
```

The astore from the trained model is fed into SAS Event Stream Processing for scoring new video feeds from the train track. This information can be combined with geolocation and other telematics data to compute a robust track health score. Display 1 shows a snapshot of SAS Event Stream Processing Streamviewer. The green enclosed box on the image feed shows the region of interest that is used in training the deep learning model for classification of loose ballast. In the same display, geolocation information and track health index are also available. The combined information can be monitored in real time for track health. In the future, this use case can be expanded to include the following tasks, which are very important for optimal operation of trains: analyzing defects and monitoring the track's geometry, positive train control, and accurate location of signals, switches, mileposts, and crossings.



Display 1. SAS Event Stream Processing Streamviewer Monitoring Loose Ballast Conditions and Track Health

CONCLUSION

As explained in the three applications of autonomous driving, manufacturing, and public transportation, processing and analyzing image data close to its point of generation has great potential for improving operational efficiencies and achieving a better return on investment. SAS Viya and SAS Event Stream Processing provide an end-to-end platform for training computer vision models at the server side and deploying them to the edge. A variety of computer vision models (including image classification, keypoints detection, object detection, and image segmentation) are supported in order to build image-based applications of ultra-low latency that run on edge devices or mobile devices.

For more information about building and deploying computer vision models using SAS, see the list of the recommended readings.

REFERENCES

Combaneyre, F. 2018. "Real-Time Image Processing and Analytics Using SAS Event Stream Processing." *Proceedings of the SAS Global Forum 2018 Conference.* Cary, NC: SAS Institute Inc. Available https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2103-2018.pdf.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. 2017. "CARLA: An Open Urban Driving Simulator." arXiv preprint arXiv:1711.03938.

Gartner Glossary, "Edge Computing." Available at https://www.gartner.com/en/information-technology/glossary/edge-computing.

Gyarmathy, K. 2019. "The Benefits and Potential of Edge Computing." Available at https://www.vxchnge.com/blog/the-5-best-benefits-of-edge-computing.

He, K., Zhang, X., Ren, S., and Sun, J. 2015. "Deep Residual Learning for Image Recognition." arXiv preprint arXiv:1512.03385.

Ioffe, S., and Szegedy, C. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." arXiv preprint arXiv:1502.03167.

Long, X., Du, M., and Hu, X. 2019. "Exploring Computer Vision in Deep Learning: Object Detection and Semantic Segmentation." *Proceedings of the SAS Global Forum 2019 Conference.* Cary, NC: SAS Institute Inc. Available https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3317-2019.pdf.

Miller, P. 2018. "What Is Edge Computing?" May 7, 2018. Available at https://www.theverge.com/circuitbreaker/2018/5/7/17327584/edge-computing-cloud-google-microsoft-apple-amazon.

Moganti, M., and Ercal, F. 1996. "Automatic PCB Inspection Algorithms: A Survey." *Computer Vision and Image understanding*, col. 63, no. 2, pp. 287-313.

Paszke, A., Chaurasia, A., Kim, S., and Culurciello, E. 2016. "Enet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation." arXiv preprint arXiv:1606.02147.

Shaw, K. 2019. "What Is Edge Computing and Why It Matters." November 13, 2019. Available at https://www.networkworld.com/article/3224893/what-is-edge-computing-and-how-it-s-changing-the-network.html.

Simonyan, K., and Zisserman, A. 2015. "Very Deep Convolutional Networks for Large-Scale Image Recognition." *International Conference on Learning Representations (ICLR)*, 1409-1556.

Solomon, B. 2001. *Railway Maintenance Equipment: The Men and Machines that Keep the Railroads Running*. MBI Publishing Company. ISBN 0-7603-0975-2.

Stubbles, C. "Design Guidelines for Cypress Ball Grid Array (BGA) Packaged Devices." Available at https://www.cypress.com/file/45826. Accessed on February 14, 2020.

Weedy, S. 2017. "Maintenance vs Total Renewal – A Methodology for Assessing Track Ballast Condition." RailTech.com. Available at https://www.railtech.com/railtech/railtech2017news/2017/03/22/maintenance-vs-total-renewal-a-methodology-for-assessing-track-ballast-condition/.

RECOMMENDED READING

- *SAS® Visual Data Mining and Machine Learning: User's Guide*
- SAS Deep Learning Python (DLPy) package
- SAS Event Stream Processing Python Interface (ESPPy)

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Maggie Du
SAS Institute
+1 919-531-5291
Maggie.Du@sas.com

Juthika Khargharia
SAS Institute
+1 919-531-8893
Juthika.Khargharia@sas.com

Shunping Huang
SAS Institute
+1 919-531-3261
Shunping.Huang@sas.com

Xunlei Wu
SAS Institute
+1 919-531-2606
Xunlei.Wu@sas.com