sas

Computer Vision with SAS®

Special Collection

Foreword by
Susan Kahler

The correct bibliographic citation for this manual is as follows: Kahler, Susan. *Computer Vision with SAS®: Special Collection*. Cary, NC: SAS Institute Inc.

Computer Vision with SAS®: Special Collection

ISBN 978-1-952365-04-1 (Paperback)
ISBN 978-1-952365-01-0 (Web PDF)

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

July 2020

Table of Contents

Free SAS® e-Books: Special Collection

In this series, we have carefully curated a collection of papers that introduces and provides context to the various areas of analytics. Topics covered illustrate the power of SAS solutions that are available as tools for data analysis, highlighting a variety of commonly used techniques.

Discover more free SAS e-books!
support.sas.com/freesasebooks

sas.com/books
for additional books and resources.

sas
THE POWER TO KNOW®

Foreword

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos and deep learning models, machines can accurately identify and classify objects and then react to what they "see." SAS offers many different solutions to train computers to "see" by identifying and classifying objects, and several groundbreaking papers have been written to demonstrate these techniques.

Early experiments in computer vision took place in the 1950s using some of the first neural networks to detect the edges of an object and to sort simple objects into categories like circles and squares. These days, computer vision rivals and surpasses human visual abilities in many areas. A few factors have converged to bring about a renaissance in computer vision. Thanks to mobile technology, we have a saturation of photos and videos at our disposal. Computing power has become more affordable and easily accessible. Hardware designed for computer vision and analysis is more widely available. Lastly, algorithms like convolutional neural networks are optimized to fully take advantage of these capabilities.

Computer vision works by going through a multi-stage process. The first stage is about acquiring the images through video, photos, or 3-D technology for analysis. Then the images need to be analyzed. Deep learning models automate much of this process, but the models are often trained by first being fed thousands of labeled or pre-identified images. In the final step, the interpretive step, the deep learning mode is deployed to score new images or video feed.

We have carefully selected a handful of these from recent SAS Global Forum papers to introduce to the topics and let you sample what each has to offer. A great starting point is the paper, "Analytics with Computer Vision on the Edge." This paper includes computer vision in the IoT across multiple industries. We summarize several applications in industries such as manufacturing, retail, health care, insurance, transportation, and utilities.

Next up is "Exploring Computer Vision in Deep Learning: Object Detection and Semantic Segmentation." In this paper, you will learn about object detection and image segmentation models and the details behind how those models work. Those models are used extensively in the health care domain as illustrated in the papers "Medical Image Analyses in SAS® Viya® with Applications in Automatic Tissue Morphometry in the Clinic," which applies U-Net models, and "Medical Image Analytics in SAS® Viya® with Applications in the Treatment of Colorectal Cancer Spread to the Liver," which applies a YOLOv2 model for analysis. You will be amazed at how computer vision can be used to augment medical clinicians with the goal to provide better patient outcomes.

In "Deploying Computer Vision by Combining Deep Learning Action Sets with Open Source Technology," ScottishPower Energy Retail has applied computer vision to automatic meter reading (AMR) from customer submitted photos. They are using the Deep Learning action set in SAS® Visual Data Mining and Machine Learning (VDMML) with SAS Deep Learning with Python (DLPy) and Keras. This is a great example of combining multiple deep learning models into a singular mobile application using the Open Neural Network Exchange (ONNX).

The last two papers, "Bringing Computer Vision to the Edge: An Overview of Real-Time Image Analytics with SAS®" and "How to Use Deep Learning with Your Internet of Things (IoT) Digital Twin**"** are focused on analyzing image data close to its point of generation, via the use of edge devices and streaming analytics. The last paper also includes an example of computer vision applied to visual inspection.

Deploying Computer Vision by Combining Deep Learning Action Sets with Open Source Technology
By Jonny McElhinney and Duncan Bain, ScottishPower Energy Retail Ltd, and Haidar Altaie

Whilst early computer vision dates back as far as 1927, it has gained momentum in the last years due to the developments in the fields of deep learning and artificial intelligence. With the desire for applying computer vision in ever more general and flexible contexts, the challenge arises how we can push image processing models to production in a robust way.

This paper focuses on doing Automatic Meter Reading (AMR) using customer-submitted photos of their meters. The challenges in this context are two-fold. First, we need to localise the box containing the digits, and then we must classify each digit with the correct label, 0-9. Many approaches are available, but both of these challenges can be addressed by combining the Deep Learning action set in SAS® Visual Data Mining and Machine Learning (VDMML) with DLPy and Keras.

However, when applying these models in a real-world business context, an additional challenge arises, which is how to deploy and keep track of these models in a consistent way. This paper shows how SAS® and open source tools can be used together to provide a consistent approach both to creating as well as managing and deploying models.

Bringing Computer Vision to the Edge: An Overview of Real-Time Image Analytics with SAS®
By Maggie Du, Juthika Khargharia, Shunping Huang, and Xunlei Wu

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.

How to Use Deep Learning with Your Internet of Things (IoT) Digital Twin
By Brad Klenz

With the Internet of Things (IoT), a digital twin is created to have a virtual representation of a remote device or system. The digital twin shows you the device's operating condition, no matter where it is physically located. IoT devices have a number of sensors installed on them, as well as sensors for the environment around them. Analytics can bring this sensor data together to create a true real-time digital twin. A previous paper showed how streaming analytics are used for device state estimation and anomaly detection. This paper explains how deep learning can be added to your digital twin for more understanding. Image and video analytics are used to capture operating conditions that are missed by regular sensors. Recurrent neural networks (RNN) add temporal data analysis and pattern detection in real-time data streams that are prevalent in digital twins. With these deep learning capabilities, your digital twin provides a new level of insight for your remote devices.

We hope these selections give you a practical overview how computer vision can be used to solve a variety of business problems. There are many ideas as to how you can incorporate computer vision into your organization. Enjoy!

Susan Kahler is a Global Product Marketing Manager for AI at SAS, focusing on deep learning and computer vision. She has her Ph.D. in Human Factors and Ergonomics, having used analytics to quantify and compare mental models of how humans learn complex operations. Throughout her well-rounded career, she has held roles in user-centered design, product management, customer insights, consulting and operational risk. Susan recently completed her Master of Science in Analytics, focusing on health care analytics. She is a regular contributor to the SAS Data Science Blog at https://blogs.sas.com/content/subconsciousmusings/.

Analytics with Computer Vision on the Edge

By Juthika Khargharia and Hamza Ghadyali

Introduction

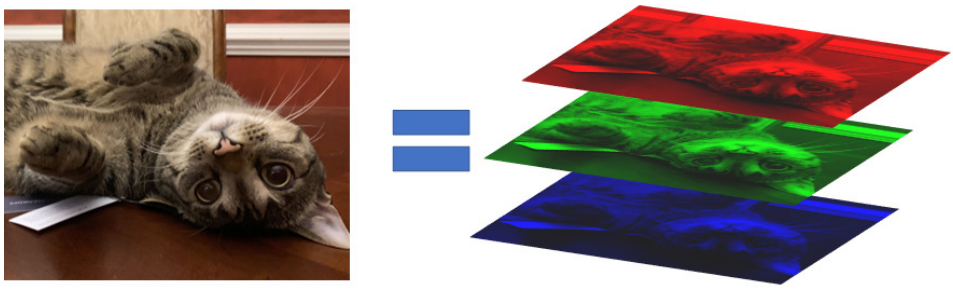
Imagine that every camera is a smart AI-equipped camera. Think of the business value that such cameras would add to the manufacturing, retail, and utilities industries. But this possibility does not need to be left to the imagination. Now that powerful hardware in edge devices has become more affordable, those devices can be used in combination with the latest advances in the SAS platform. We now can create intelligent computer vision systems by building complex deep learning models with SAS Visual Data Mining and Machine Learning (VDMML) on SAS Viya, integrate them with other SAS analytics tools, and deploy those models on edge devices to score streaming data using SAS Event Stream Processing. With that framework in place, we can integrate the newest data into our analysis, and make decisions in *real time.*

This chapter explores how you can use computer vision to solve problems with real business value. It shows how you can build computer vision models with deep learning to solve previously unsolvable problems through examples from specific applications. It also shows the advantages of deploying these models for real-time analytics and provide some resources for getting started in this exciting field.

Computer Vision with Deep Learning

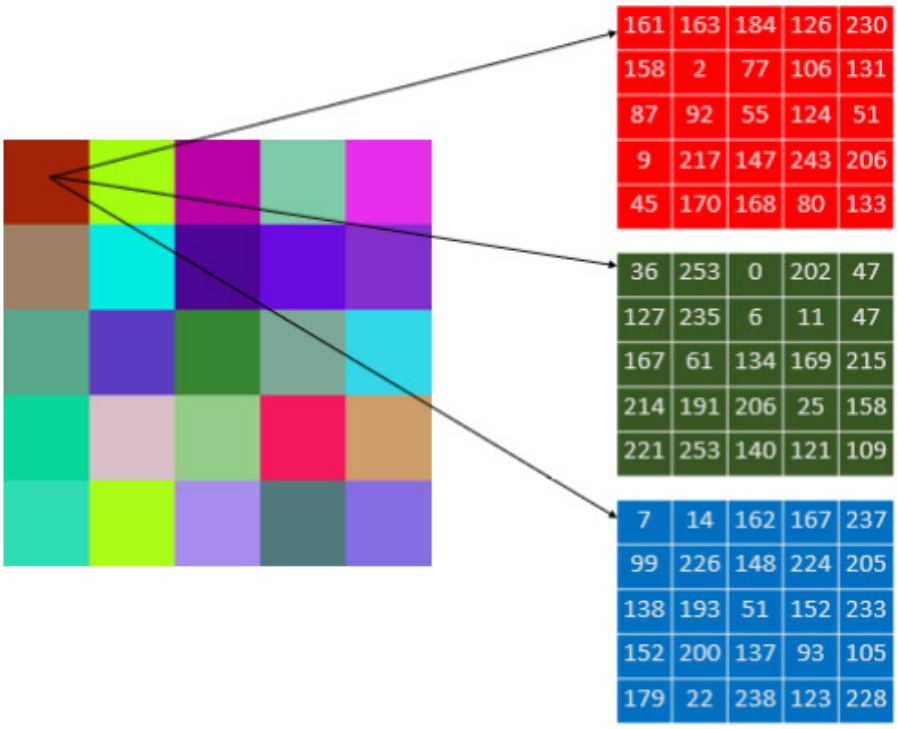
How does a computer "see" an image? Why has it been a challenge to perform analytics on images? To answer these questions, we first must think of an image as a grid of colored dots (pixels). Colors can be represented as a mixture of three primary colors, such as **r**ed, **g**reen, and **b**lue (RGB representation). Thus, three numbers are needed to represent a color for a single pixel. If you have a 12-megapixel camera, then each image is represented by 3*12 = 36 million numbers. If colors are stored as 8-bit, then each of these numbers takes on a value between 1 and 2^8=256 (in practice, the range is 0 to 255). Put another way, an image is represented numerically by three matrices (or 2D arrays). Each matrix is called a *channel*, and there is one channel for each of the primary colors red, green, and blue. (See Figure 1.)

Figure 1: Image of a Cat Represented Numerically as Three 2D Arrays



Now imagine zooming into a 5x5 patch of pixels on that image. In Figure 2, the color of the square in the top left corner is represented by three numbers (161 for red, 36 for green, 7 for blue). Seen on the right, there are three 5x5 arrays that explicitly show the color components for each square in the left image.

Figure 2: Zooming into a Patch of Pixels



A typical movie or television show might stream at 30 images every second. This volume can leady to truly big data, which presents two challenges. First, there is managing such a large amount of data that is streaming in at a rapid rate. Second, there is identifying objects of interest, which involves complex relationships between groups of pixels.

Simple rules to classify and track objects do not exist. Deep learning can be used to develop a model that learns complex rules with minimal human intervention. Deep learning makes image classification, object detection,

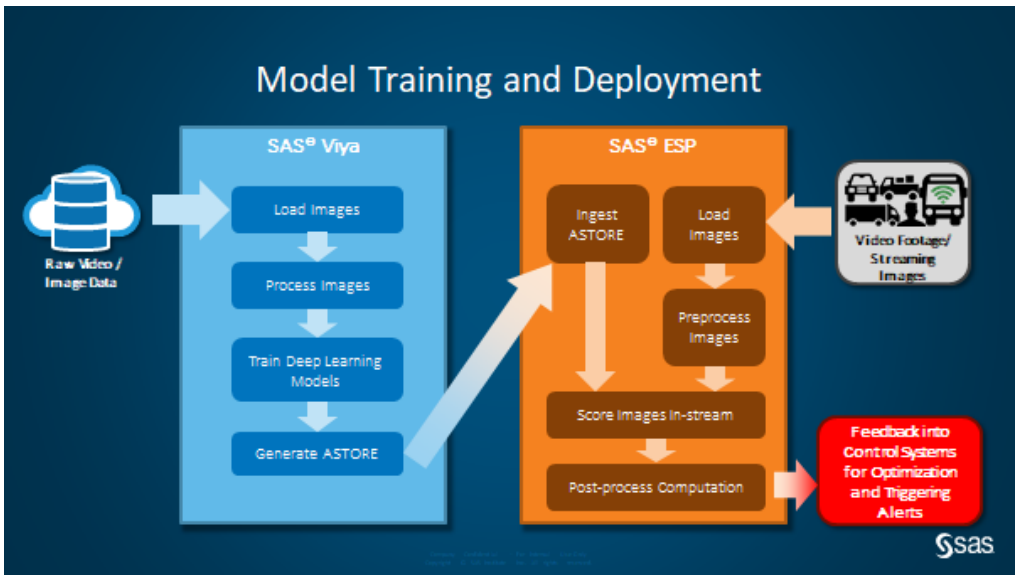
and object tracking much easier. This does not mean that traditional computer vision and image processing techniques are obsolete. Those techniques continue to play an important role in making models robust to the types of variation that is seen in true production environments where the models will ultimately need to perform.

The basic pipeline for building computer vision problems usually involves taking a supervised learning approach. First image or video data is collected. Then, a subset of that data is selected for labeling that is used for training the model offline. A model's performance can be validated with the remaining data. After a model is performing well, it is deployed. In most use cases, real-time deployment has the most value.

Powerful models that leverage cutting-edge analytics can be built using SAS VDMML and are best deployed for real-time applications with SAS Event Stream Processing. When you combine these two products with the rest of the SAS platform, further advanced analytical models can be built, and after deployment, both data and models can be managed and monitored for continued performance.

Figure 3 shows the end-to-end computer vision pipeline starting with model training in SAS Viya and real-time deployment of these models in SAS Event Stream Processing.

Figure 3: Model Training and Deployment



To achieve the pipeline described in Figure 3 using SAS software, one can specifically take advantage of DLPy in SAS Viya for model training and SAS ESP for model deployment. DLPy is a high-level Python package for the SAS Deep learning features available in SAS Viya. DLPy is designed to provide an efficient way to apply deep learning methods to image, text, and audio data.

Example Code 1 shows a Python code snippet from DLPy that generates and saves an ASTORE file from a pre-trained VGG16 model.

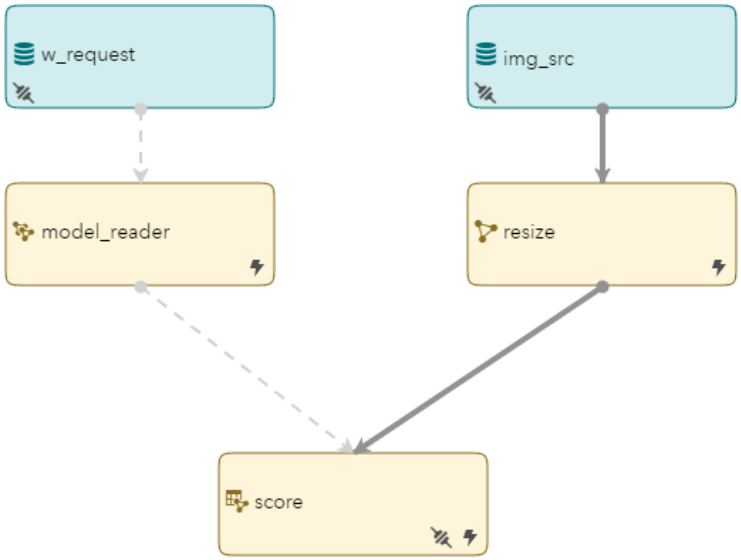
Example Code 1

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

An *analytic store file* (ASTORE) is a binary file that contains the state of a trained predictive analytic model. A key feature of an analytic store file is that it is transportable between platforms and hence can be easily ingested by SAS Event Stream Processing for real-time deployment.

Figure 4 shows the ingestion of the ASTORE file using the Model Reader window in SAS Event Stream Processing Studio to score new images. A Model Reader window (model_reader) receives requests from a Request window (w_request), fetches the specified model using the request information, and publishes the model event to the Score window (score) for scoring.

Figure 4: Process Flow Showing Ingestion of an ASTORE File and Scoring of New Images



Example Code 2 shows the XML code underlying the Model Reader window in SAS Event Stream Processing Studio.

Example Code 2

```
<window-model-reader name='model_reader' model-type='astore'/>
```

Advantages of Real-time Analytics on the Edge

In the context of computer vision, there are several additional advantages of building and deploying models that can process image and video streams in real time. From a computational point of view, there are the advantages of smarter and more efficient data management. Instead of streaming in large quantities of data into a storage device for batch analytics at a later date, real-time processing reduces the network burden and processes the data as it arrives, and rules and models can be designed to store only the most important data for future use.

Real-time analytics using computer vision also has the benefit of augmenting human effort by running side-by-side with skilled workers. Real-time analytics reduces the delay between data collection and data analytics. This reduction is crucial for applications such as the detection of suspicious activity in surveillance footage or the detection of defects in products during manufacturing, or the other use cases discussed next.

Luckily, we can reap the rewards of these advantages by deploying computer vision models with SAS ESP. SAS ESP is optimized to run on affordable Nvidia GPU hardware for real-time model scoring. The outputs of the deployed models can be monitored with SAS ESP Streamviewer, where you can build custom dashboards to provide additional insights augmenting the plain footage and images of the camera.

Computer Vision Applications in the IoT

Applications of computer vision in the IoT span multiple industries. We summarize a few applications below in industries such as manufacturing, retail, health care, insurance, transportation, and utilities. These applications show how the breadth and depth of the SAS platform allow us to build a wide variety of models for many use cases, and we will see how we transform unstructured image and video data into structured metrics providing actionable analytics to deliver value, support innovation, and complement human skills.

A key advantage to building these models with deep learning is that we can build robust models using multiple inputs, such as images or videos from cameras combined with sensor data, or other process data.

The value for all these use cases is in real-time deployment where streaming data (images, video, sensors) is analyzed with sophisticated models providing actionable intelligence that can automatically send out alerts or feed back into control systems in real time, which is powered by SAS ESP.

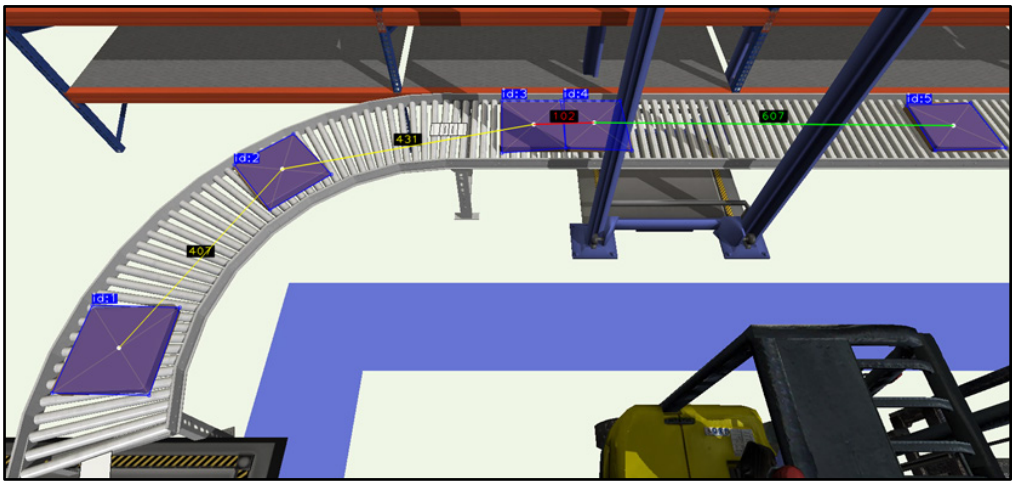
Manufacturing

Downtime Reduction in Production Line Monitoring

Deep learning object detection models can be used in manufacturing to avoid downtime in production lines. By robustly tracking products as they move through the production line, we can monitor inefficiencies in the flow; detect defects; or detect, predict, and prevent costly collisions. Collisions can result in product damage, or worse, a shutdown of the continuous production line, resulting in major losses in productivity.

Figure 5 displays the result of a computer vision model that is detecting objects, tracking them with IDs, and determining relative positions and velocities of objects. The output of the computer vision model can then be further processed with models built using the broad advanced analytics capabilities afforded by the SAS platform to trigger an alert in the event of a detected defect, irregular flow, or a collision.

Figure 5: Production Line Monitoring

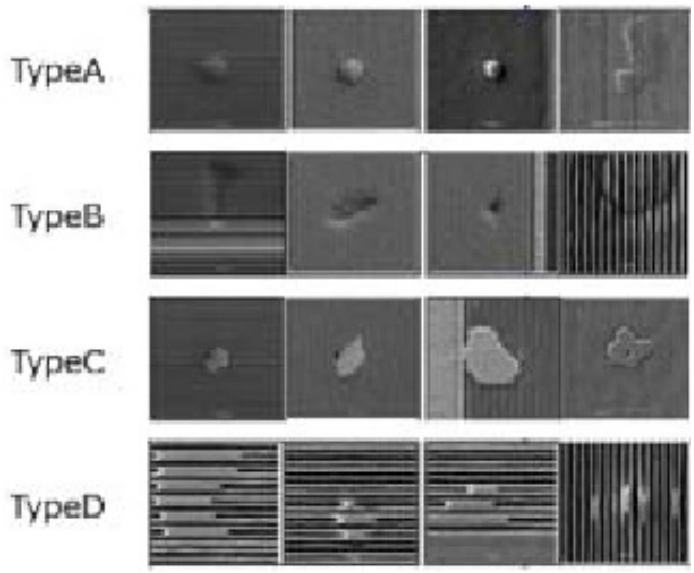


Defect Detection in the Semi-conductor Industry

In order to improve yield and optimize productivity in semiconductor devices, it is necessary to implement reliable inspection methodologies. Vision inspection technologies have been developed to analyze images of wafers for defects. Defect detection is an important part of the wafer fabrication process. Defects can be classified into different categories based on specific patterns, geometries, wafer scratch, and so on.

Figure 6 shows an example of such defect categories. After the defect is detected, it needs to be classified to one of several defect categories; this enables correction of the fabrication process.

Figure 6: Example of Wafer Defects



(Source: Kazunori Imoto, Tomohiro Nakai, Tsukasa Ike, Kosuke Haruki, Yoshiyuki Sato. 2019."A CNN-Based Transfer Learning Method for Defect Classification in Semiconductor Manufacturing," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 455-459.)

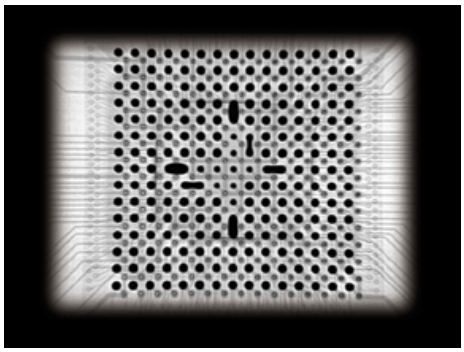
Deep learning models such as convolutional neural networks can be trained against historical image data to learn the defect features accurately. The trained model is then applied to new incoming defective image data to categorize them into one of the several defect types. It is important to note that the defects themselves might evolve over time, and new defects can emerge. Although it is important to re-train the models using new data frequently, unsupervised learning models can also go a long way toward identifying new classes of defects. SAS provides a full stack of machine learning, deep learning, and statistical models that can be used to train image data. The trained models can then be used to classify new defects in real-time thereby enhancing the productivity of the operator.

Quality Inspection in Surface Mount Technology (SMT) Printed Circuit Boards (PCBs)

To enhance the quality of PCB production, it is imperative to perform robust inspection of the boards. Good quality SMT inspection machines are often used to look at missing components (for example, capacitors, resistors, and so on), inspection of skewed components, classification of components into different categories, and so on. Advanced Optical Inspection (AOI) machines as well as Advanced X-ray Inspection machines (AXI) are often used in conjunction to captures images of the boards in the optical and X-ray wavelengths respectively. This enables the operator to analyse not only the visually available components but also the solder joints. With X-ray technology, one can obtain a direct view of the solder joints; data from this could be used for analysis of joint quality, Head-in-Pillow (HiP) defects, and so on.

Figure 7 shows sample images taken from an AXI machine. It highlights issues with the solder join such as too much solder, misplaced solder, or errant solder balls.

Figure 7: Sample Images from AXI Machines



(Source: Jonathan Titus. 1999. "X-Rays Expose Hidden Connections." *Test & Measurement World*: 28-35.)

The data from known defects and non-defects are captured to train convolutional neural networks. After the defects are learned by the model, the scoring logic is sent to a real-time inference engine software that is installed on the inspection machines. This enables real-time scoring of the image components as they are scanned. It also enhances operator efficiency by relieving operators of their manual visual inspection work. One of the challenges faced is collection of enough data that represents defects so that deep learning models can be trained effectively. A possible work-around to this challenge is implementation of several image augmentation techniques that SAS provides out-of-the-box to increase the size of the training data set. Models can be continually improved as new defects are added to the defects database.

Government

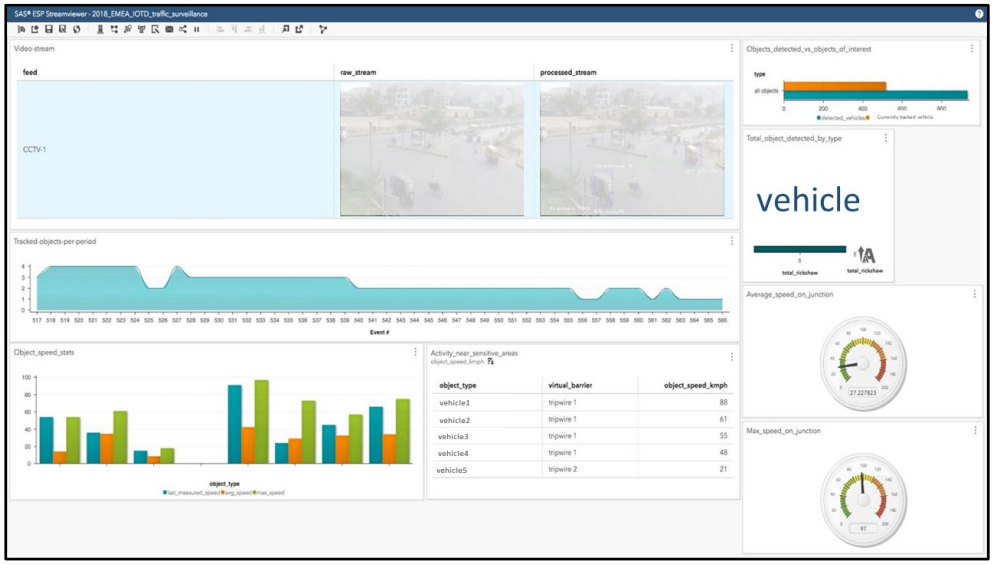
Smart City Applications

A *smart city* is defined as an urban area where sensor data is collected from various connected devices and assets in the city for efficient allocation of resources and assets to bring overall welfare to the citizens. While use cases around smart cities abound, here we focus on one such case: monitoring and management of traffic flow and transportation systems. Many cities around the world have invested heavily in installing CCTV cameras at several busy intersections in their cities. As part of the smart city initiatives, city officials want to apply machine learning and deep learning on the video feeds coming off these cameras. The city officials are interested specifically in several metrics such as vehicle classification at traffic junctions; counts based on vehicle type; real-time speed monitoring; traffic congestion analysis based on time of the day, day of the week, and holidays; monitoring vehicles that violate rules such as wrong turns; U-turn violations; and so on. The city command and control center are interested in collecting these key metrics from multiple video feeds, aggregating the data, and monitoring real-time reports.

This use case is solved using a two-pronged approach using SAS multi-phase analytics. The first part involves training deep learning object detection models that learn to differentiate between vehicle types, pedestrians, and so on. The second part is concerned with model deployment. Once we have a fully trained model, it can be deployed on a GPU-enabled edge device that will score new incoming data against the prebuilt deep learning model. Using edge software, one can build out the logic for lane violation using geofencing in real-time, speed computation, monitoring counts of different vehicle types based on user-defined conditions.

Figure 8 shows a real-time dashboard in SAS Event Stream Processing Streamviewer displaying real-time values of traffic monitoring metrics such as vehicle identification, vehicle counts over a certain time frame, and speed computation.

Figure 8: SAS ESP Streamviewer



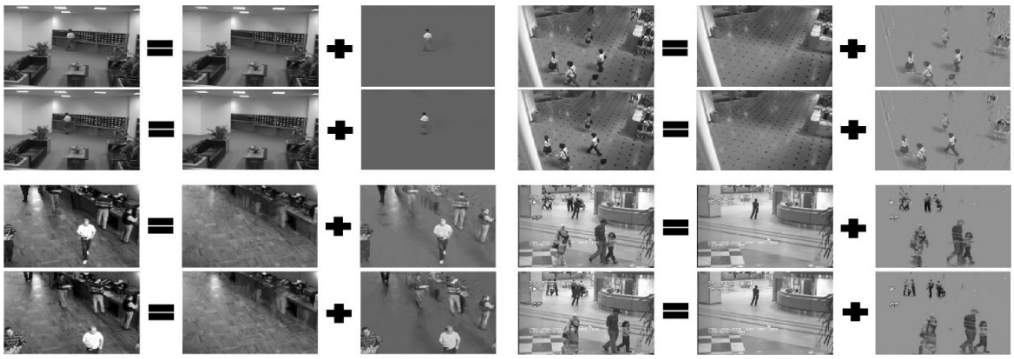
Data Management for Video Surveillance

Although video surveillance data can be used for a variety of analysis, it does come with a cost associated with video data storage. With the emergence of high-definition cameras and longer retention times, many government agencies are dealing with the cost of storing this data. Because video data is unstructured in nature, it requires special storage to be cost-effective. One way to combat this challenge is post-processing of video data to significantly reduce the amount of storage. Not all data might be valuable to store. Determining how much data to transmit and store is a concern surrounding many IoT applications.

One method for reducing video footage data is called *robust principal component analysis* (RPCA), an algorithm supported by SAS under SAS VDMML. RPCA decomposes an input matrix into the summation of a low-rank matrix and a sparse matrix. The robustness of the method comes from its ability to handle anomalies. Anomalies are captured within the sparse matrix and can be used to examine potentially abnormal behavior within the data set. RPCA is a great option for dimensionality reduction in video data.

In Figure 9, the original matrix captures the raw video footage (left panel). The low rank matrix captures the static background (middle panel) and the sparse matrix captures the moving background (right panel). In many cases, storing the moving foreground footage can provide enough data for analysis and thereby reduce storage.

Figure 9: Decomposition of Video Content



(Source: Zhou, Tianyi. 2011. "GoDec: Randomized Low-rank & Sparse Matrix Decomposition in Noisy Case." *Tianyi Zhou's Research Blog.* https://tianyizhou.wordpress.com/2011/04/20/learn-low-rank-sparse-structures-via-randomized-alternating-projections/)

Transportation

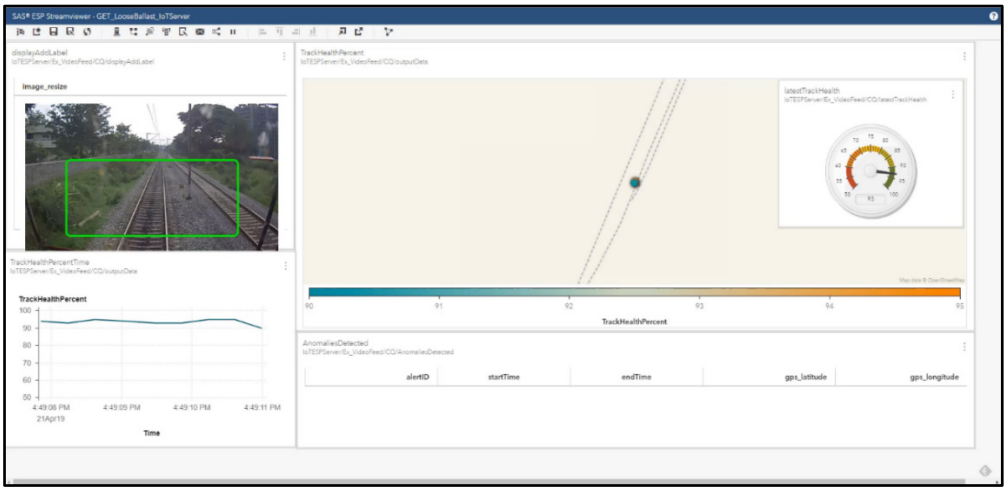
Loose Ballast Detection in Railroads

Have you ever wondered why trains pass through railroad tracks that are covered with jagged little stones? The reason is that these stones are the most efficient and cost-effective way for a robust foundation for the train tracks. They are called track ballast and they seek to keep the tracks in place. Railway tracks can change over time due to weather conditions, vibrations, ground movement, and weed growth that makes the tracks unstable. The train ballast protects them from such events. Figure 10 compares normal ballast relative to loose ballast (less gravel in the train tracks) within the red highlighted area of interest.

Figure 10: Normal Ballast (Left) Versus Loose Ballast (Right)

A computer vision application surrounding this use case has to do with loose ballast detection. The objective is to identify loose ballast and monitor track health conditions in real time using cameras mounted on the trains. These cameras are continuously capturing video of the train tracks. A deep learning model trained on historical video data learns to identify the features associated with the loose ballast condition. This model is then applied to real-time video feeds to monitor the track health alongside other structured metadata captured with the video. Figure 11 shows SAS Event Stream Processing Streamviewer monitoring track health along with geo-location in real-time. Other great applications in the field of transportation involve monitoring traffic, identifying vehicles that are driving the wrong way, and spotting detritus or other adverse conditions on roads or highways.

Figure 11: SAS Event Stream Processing Streamviewer Monitoring Track Health



Health Care

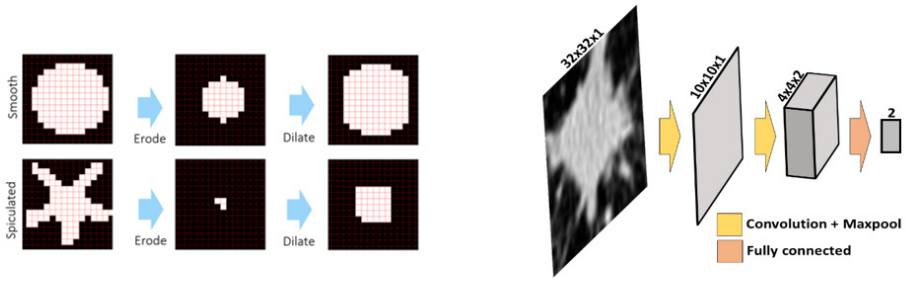
Biomedical Image Analysis

According to GE Healthcare, “Hospitals are producing 50 petabytes of data per year. A staggering 90 percent of all health care data comes from medical imaging” (GE Healthcare 2018). Biomedical image processing intersects with multiple fields such as computer science, machine learning, image processing, medicine, and other fields. Typically, radiologists would interpret biomedical images to diagnose medical conditions. However, with the growing volume of imaging studies, a radiologist’s workload is extremely demanding.

Here we discuss how SAS tools can potentially be used to alleviate a radiologist's time-consuming and demanding task. SAS supports various health and life sciences use cases ranging from management of clinical health care data to taking advantage of medical images along with statistical, data mining, text analytics, and optimization techniques for better clinical diagnosis. SAS can directly read the commonly available biomedical imaging formats such as Digital Imaging and Communication in Medicine (DICOM), Neuroimaging Informatics Technology Initiative (NIFTI), nearly raw raster data (NRRD), and so on. One can then take advantage of the biomedical image pre-processing, segmentation, visualization, and deep learning frameworks in SAS for various use cases.

Below is an example of lung nodule classification on image data provided by Society of Photographic Instrumentation Engineers (SPIE), American Association of Physicists in Medicine (AAPM), National Cancer Institute (NCI), and TCIA. Figure 12 shows an example of lung nodule classification.

Figure 12: Lung Nodule Classification Based on Metrics Derived from Engineered Features (Left) and Deep Learning Analysis (Right)



Two approaches were used in Figure 12. The left panel shows how metrics derived from engineered features can be used for classifying a benign versus a malignant nodule. Subsequent image processing steps of dilation and erosion preserves the shape of a round (possibly benign) nodule. Similar image processing steps on a spiculated (possibly malignant) nodule does not preserve the initial geometry of the nodule. Figure 12 (right) shows deep learning analysis applied to the sample benign and malignant image data, giving a 10% misclassification rate (Vadakkumpadan and Sethi 2018).

Utility

Vegetation Encroachment Monitoring on Power Lines Using Drones

Management of overgrown trees and vegetation that interfere with transmission lines is perhaps one of the most expensive operational costs for distribution of electricity. These vegetation encroachments are monitored through time-consuming visual inspections periodically, but it has proven neither to be accurate nor cost effective. Energy and utility companies are now starting to explore new technologies such as Unmanned Aerial Vehicles (UAVs) also known as drones.

For example, Duke Energy is using drones to conduct infrared equipment inspections, survey storm damage, and inspect tall structures (Wells 2018). Drones have also brought about advantages in terms of safety. Since drones are unmanned, no one needs to be inside an aircraft flying at low altitudes parallel to transmission lines. Drones can cover large distances in a single flight and provide detailed and accurate aerial images of transmission lines, surrounding vegetation, and other structures. This could lead to automation of a very expensive traditional manual process.

This use case can be tackled in three different stages, each of which requires detailed analytics drawn from computer vision, traditional machine learning, and operations research. The first stage typically relies on analysis of the images collected by the drones. One can build robust object detection models for classification of vegetation into several species. The trained models can be deployed on the drone equipment to classify these vegetation species in real time. In stage two, analytical models can be built to predict growth rates of these species. Stage three is concerned with joining the vegetation growth forecast models with maintenance and cost information to provide vegetation management intervention in terms of routing and allocation of resources.

This shows how computer vision can be combined with optimization models, where computer vision first transforms unstructured image data into useful metrics, which then are used to optimize complex systems and processes. This integration with SAS Optimization shows the advantages of the breadth of analytic capabilities of the SAS platform.

Financial Services

Insurance Claims

Effective insurance claims processing is key to insurers' operational efficiency, because they have only a short window within which to process these claims. Computer vision is starting to become inevitable for any insurer wanting to have detailed information about insured property.

By using the SAS Scripting Wrapper for Analytics Transfer (SWAT) package and SAS DLPy (a high-level Python library for the SAS deep learning features available in SAS Viya), a user can quickly perform image processing and SAS deep learning to classify defects. The SWAT package is a Python interface to the SAS Cloud Analytic Services (CAS) engine. It includes the ability to call CAS actions and process the output in various ways. These range from simple (calling CAS actions as Python methods and getting a dictionary of results back) to complex (invoking CAS actions in multiple sessions and handling server responses from them directly).

Figure 13 shows image preprocessing techniques being applied to damaged vehicle data. In the code snippet shown in the figure, images of damaged vehicles are being loaded into an in-memory CAS Table. The resulting table named "inputTable" in the example contains all the information about the images in a binary large object (blob) format.

Figure 13: SAS Edge Detection Applied to Damaged Vehicle Data

Load images and resize

```
conn.image.loadImages(casout={'name':'inputTable', 'replace':True}, path= path_source_images)
conn.image.processimages(casout={'name':'inputTable_resized', 'replace':True},
                         imagefunctions=[{'functionoptions':{'width':1000,'functiontype':'RESIZE','height':6
00}}],
                         imagetable={'name':'inputTable'})
imageTable = conn.CASTable('inputTable_resized')
imageShow(conn, imageTable, 0, 4)
```

```
NOTE: Loaded 4 images from /viyafiles/turtui/image_processing/car_accident into Cloud Analytic Services ta
ble inputTable.
NOTE: Table INPUTTABLE contains compressed images.
NOTE: 4 out of 4 images were processed successfully and saved as compressed images to the Cloud Analytic S
ervices table inputTable_resized.
```

Pre-processing techniques such as resizing of the image data (shown in the example) can be applied directly via CAS actions surfaced as Python methods. Since CAS can be used on a local desktop or in a hosted cloud environment, you can analyze extremely large data sets using as much processing power as you need, while still retaining the ease-of-use of Python on the client side. Typical use cases in the insurance industry are listed below:

- Images of damaged vehicles that aid the insurer to predict the target event. Is it a write-off or a claim?
- Satellite images of homes and buildings that help in evaluating conditions of roofs for insurers at the time of underwriting. This could possibly automate scheduled inspections.

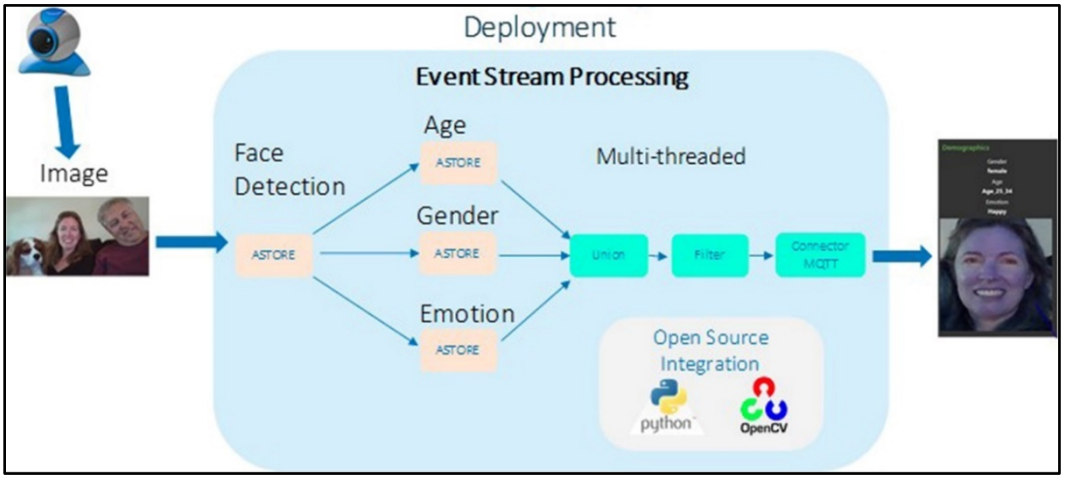
Retail

Customer Experience

Providing a personalized and enhanced customer experience in the store in real time is key in ensuring customer loyalty. Applications of machine learning, computer vision (CV), and streaming analytics covers the entire gamut from enhancing real-time, in-store customer experience, to grid surveillance and security, to improved manufacturing quality.

Facial recognition can be used to identify loyalty members as they step in the store. A camera that can scan shoppers as they enter the store combined with sophisticated deep learning models are able to identify them along with other metrics such as their gender, age, and emotion. This information could further be combined with the individual's taste profile and intelligent product recommendations could be made in real time. Facial recognition techniques can also quickly and accurately count the number of people in the store providing the data to determine the busiest days and times for a certain business. Figure 14 shows an implementation methodology for understanding key metrics about customers in real time.

Figure 14: SAS Pipeline Integrating Video Feeds from a Camera with Real-time Analytics



Here, a video stream is first captured by a camera. This stream is directly fed into SAS ESP Engine using the UVC connector. ASTORE files created from offline deep learning models that can identify customer's age, gender, and emotions are used to score the incoming video stream. Score results are then aggregated, enhanced, and published to an MQ Telemetry Transport (MQTT) connector. MQTT is a lightweight, publish-subscribe network protocol that transports messages between devices. To replicate the steps of this implementation framework please see https://communities.sas.com/t5/SAS-Communities-Library/Deploy-analytics-using-computer-vision-model-training-and/ta-p/582753) .

The analysis helps in making informed decisions around staffing in real time, how the customers move through the store, where customers spend the most time, and how efficient the check-out lines are. CV-enabled inventory management applications also exist that can scan products by their logo, shape, color, and other features. Finally, augmented reality applications are also being used to make recommendations about what looks good based on a customer's body type.

Data for Good

Wildlife Tracking

Applications of computer vision around Data for Good are many. Here we will cover one particular use case around wildlife monitoring. SAS works with nonprofit organizations such as WildTrack (wildtrack.org) that use non-invasive techniques to monitor endangered species.

Using digital footprints of animals, different species of animals can be identified with 90% accuracy. So far, these algorithms from SAS have been used to identify up to 15 different endangered species. This data has been collected to determine the distribution of species in their respective habitats so that conservation programs can be put in place and better strategies about which species needs monitoring can be implemented. SAS software has been used to import available images of labeled footprints and then perform a footprint classification task using SAS DLPy.

After the relevant features are learned, the deep learning model can be applied against new footprints to classify them accurately. However, accuracy is strongly dependent on the available data at hand. For detailed information, please check https://www.sas.com/en_us/explore/analytics-in-action/impact/wildtrack.html.

Figure 15 shows an example image of a footprint that can be ingested into the SAS deep learning framework to learn features and apply on new data.

Figure 15: A Footprint That Can Be Ingested by a SAS Deep Learning Model

(Source: www.wildtrack.org)

Safety Compliance

Personal Protective Equipment Verification

For a multitude of industries like construction, transportation, manufacturing, or any field where worker safety is crucial, we can build object detection models to detect people and to detect whether they are wearing the requisite safety gear (vests, hard hats, shoes, and so on). We can also build models to make sure that people are not in or moving through dangerous zones. These models can be used to ensure higher safety compliance to prevent workplace injury or death.

Conclusion

Currently, there are excellent opportunities in so many different industries to build CV models with SAS Viya, deploy models for real-time scoring with SAS ESP, and integrate with existing systems to provide substantial business value. Practical use cases of AI should also improve the lives of workers and consumers and should be thought of as a tool to assist and augment human activities. As mentioned in this chapter, the use cases of

computer vision in IoT span multiple industries. We have seen how SAS software plays a crucial role in every step of the IoT Analytics life cycle – data management and processing of image data, building deep learning models from scratch as well as taking advantage of transfer learning, and deployment of these models in real-time. Integration of SAS software with open-source software provides yet another level of flexibility, enabling the user to work with the best of both worlds. Armed with these tools, an AI practitioner can start to tackle any business problem in a systematic and robust manner.

References

GE Healthcare. 2018. "Beyond Imaging: the paradox of AI and medical imaging innovation." *GEHealthcare.com.* https://www.gehealthcare.com/article/beyond-imagingthe-paradox-of-ai-and-medical-imaging-innovation

Wells, J. 2018. "*5 ways Duke Energy is using drone technology*," Duke Energy Corporation. https://illumination.duke-energy.com/articles/5-ways-duke-energy-is-using-drone-technology.

Vadakkumpadan, F. and S. Sethi. 2018. "Biomedical Image Analytics using SAS Viya," *SAS Institute,* Paper SAS1961-2018.

About the Contributors

Juthika Khargharia is a Principal Solutions Architect in the SAS IoT Division specializing in machine learning, deep learning and artificial intelligence. In her role, she assists customers in defining their business problems and uses SAS advanced analytics solutions to help them reach their business goals and objectives. Juthika holds a PhD in Astrophysics and Planetary Sciences from the University of Colorado.

Hamza Ghadyali, Computer Vision Lead at the AI Center of Excellence at SAS, assists the world's largest organizations in identifying the best opportunities for leveraging AI. By building models that strategically align business goals with cutting-edge analytical capabilities, Hamza generates results that deliver value. Hamza earned his PhD in Mathematics from Duke University where he invented geometric techniques to analyze data from dynamical systems and discovered unique signatures of neuroelectrical activity during epileptic seizures.

Paper SAS3317-2019

Exploring Computer Vision in Deep Learning: Object Detection and Semantic Segmentation

Xindian Long, Maggie Du, and Xiangqian Hu, SAS Institute Inc., Cary, NC

ABSTRACT

This paper describes the new object detection and semantic segmentation features in SAS Deep Learning, which are targeted to solve a wider variety of problems that are related to computer vision. The paper focuses on algorithms that are supported on SAS® Viya®, specifically Faster R-CNN and YOLO (you only look once) for object detection, and U-Net for semantic segmentation. This paper shows how to use the functionality of the Deep Learning action set in SAS® Visual Data Mining and Machine Learning in addition to DLPy, an open-source, high-level Python package for deep learning. The paper demonstrates applications of object detection and semantic segmentation on different scenarios, and it shows how to prepare data, build networks, select parameters, load or train the weights, and display results. Future development and potential applications in different areas are discussed.

INTRODUCTION

Computer vision is about understanding the visual world around us through digital images and videos. In applications such as self-driving cars, production line automation, face recognition, and medical image processing, computer software analyzes the image content and accomplishes one or several of the following basic tasks: image classification, keypoints detection, object detection, segmentation, and so on. Briefly, image classification represents the task of given an image, discovering the main content in the image. Object detection locates the positions and categories of objects in a given image. Semantic segmentation classifies the pixel-level category assignments, while instance segmentation, assigns different labels for pixels belong to different instances of the same object type.

Traditionally, computer vision tasks are accomplished by manually designed features, including Gabor filter, Gaussian filter, Scale Invariant Feature Transform (SIFT) filter, and so on. Recently, Deep Learning (that is, deep neural networks) has been proven to boost the field tremendously. Not only can computers complete the above tasks much faster, more accurately, but also with little human crafted features since these features are learned automatically using the input data.

SAS Viya® supports computer vision through SAS Deep Learning with features including image classification, keypoints detection, object detection, and semantic segmentation. For all these features, two interfaces are available:

- CAS (Cloud Analytic Services) actions: These give users more granular control over the various options.
- DLPy (https://github.com/sassoftware/python-dlpy): It has a Keras-type Python interface with a higher abstraction.

From this paper, you will learn how to use the features of object detection and semantic segmentation by working through some real-world examples. The object detection examples use different CAS actions through our CAS action set python programming interface, while the semantic segmentation is illustrated through DLPy.

OBJECT DETECTION

Object detection analyzes complex images that contain a mixed multitude of objects, at different distances and locations, amidst varying, often visually noisy backgrounds. Objects can appear anywhere within the visual frame, be near or far, and can overlap with each other. Object detection locates and classifies unknown objects, as well as determining their boundaries as shown in Figure 1.

Figure 1. The Concept of Object Detection

Object detection is a challenging and one of the most fundamental tasks in computer vision. Lately CNN (Convolutional Neural Networks) based deep learning algorithms like YOLO [1] (You only look once), SSD [2] (Singleshot multibox detector), R-CNN [3] (Region proposal networks), Faster R-CNN [3], RetinaNet [4], and so on, have been implemented to address this problem and have been very successful.

Object detection algorithms can be categorized as below:

1. The first algorithm category is to do region proposal first. This means regions highly likely to contain an object are selected either with traditional computer vision techniques (like selective search), or by using a deep learning based region proposal network (RPN). Once you have gathered the small set of candidate windows, you can formulate a set number of regression models and classification models to solve the object detection problem. This category includes algorithms like Faster R-CNN, R-FCN [5], and FPN-FRCN [6]. Algorithms in this category are usually called two-stage methods. They are generally more accurate, but slower than the single-stage method introduced below.
2. The second algorithm category only looks for objects at fixed locations with fixed sizes. These locations and sizes are strategically selected so that most scenarios are covered. These algorithms typically separate the original images into fixed size grid regions. For each region, these algorithms try to predict a fixed number of objects of certain, pre-determined shapes and sizes. Algorithms belonging to this category are called single-stage methods. Examples of such methods include YOLO, SSD, and RetinaNet. Algorithms in this category usually run faster but are less accurate. This type of algorithm is often used for applications requiring real-time detection.

SAS deep learning supports two representative algorithms, Faster R-CNN and YOLO, which belong to the two above algorithm categories respectively.

YOLO

YOLO (You Only Look Once) is the representative algorithm in single-stage object detection method. The steps it follows to detect objects are represented in Figure 2 and in the list below:

Figure 2 Steps illustrating the YOLO Algorithm

1. Separate the original image into grids of equal size.
2. For each grid, predict a preset number of bounding boxes with predefined shapes centered around the grid center. Each prediction is associate with a class probability and an object confidence (whether it contains an object, or it is just the background).
3. Finally, select those bounding boxes associated with high object confidence and class probability. The object category is the object class with the highest class probability.

The preset number of bounding boxes with pre-defined shapes are called anchor boxes. They are obtained from the data by the K-means algorithm. The anchor box captures prior knowledge about object size and shape in the data set. Different anchors are designed to detect objects with different sizes and shapes.

For example, in Figure 3, there are three anchors at one location. The red anchor box turns out to detect the person in the middle. In other words, the algorithm detects the object with the approximate size of this anchor box. The final prediction is usually different from the anchor location or size itself; an optimized offset obtained from the feature map of the image is added to the anchor location or size.

Figure 3 Anchor Boxes

FASTER R-CNN

Faster R-CNN is a two-stage object detection algorithm. Figure 4 illustrates the two stages in Faster R-CNN. Although "faster" is included in the algorithm name, that does not mean that it is faster than the one-stage method. The name is a historical artifact – it simply indicates that it is faster than it is previous versions, the original R-CNN [7] algorithm, and the Fast R-CNN [8], by sharing the computation of feature extraction for each region of interest (RoI), and by introducing the deep learning-based region proposal network (RPN).

After using many CNN layers to extract feature maps, the region proposal network (RPN) generates many windows that are highly likely to contain an object. The algorithm then retrieves the feature maps inside each window, resizes (or polls) them into fixed sizes (RoI pooling), and predicts the class probability and a more accurate bounding box for the object.

One question to consider is how the RPN generates these windows. Like YOLO, RPN also uses anchor boxes. Unlike YOLO, the anchor boxes are not generated from data but instead are of fixed sizes and shapes selected strategically to cover main object shapes and sizes. The anchor boxes can also cover the image more densely. Note that instead of performing a classification on many object categories, the RPN only does a binary classification on whether the window contains an object or not.

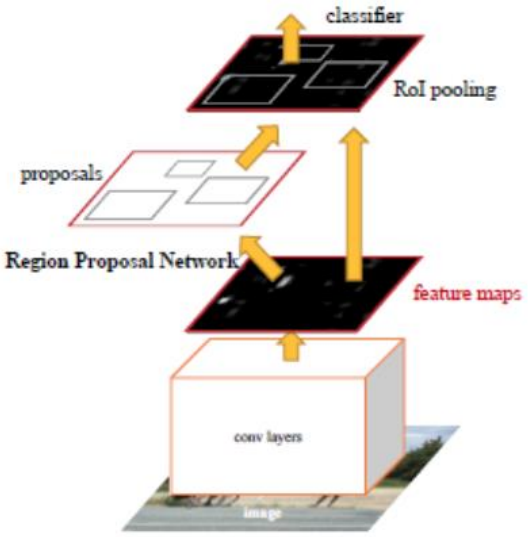


Figure 4 Stages in the Faster R-CNN Object Detection Algorithm

***Picture from the Original Faster R-CNN Paper* [3]**

BUILD DEEP LEARNING MODELS

Building and using any deep learning model involves four steps illustrated in Table 1. In the following sections, you can see how these steps are completed using SAS deep learning toolkit.

1. Preparing and Loading the Data
2. Building the model architecture, namely, the model DAG (Directed Acyclic Graph) consisting of many layers.
3. Loading or Training the weights
4. Inference and Visualization

Table 1 Steps in Building and Using a Deep Learning Model

DATA EXPLORATION AND PREPARATION

An essential part of any data science project is to explore the data, complete any pre-processing if needed, and prepare it for training or inferences. CAS provides toolsets to help you through the process.

Images and Labels

The images and label data need to be organized into a CAS table before training or scoring. Each image can contain more than one labeled object. Each label should contain the object category and the object bounding box.

In this paper, we assume images and associated labels are already joined and put in a CAS table. In the table there is a column for the image, and there are many columns for bounding box and category labels. Figure 5 shows some records for the table `trainset`.

§ Fetch

Selected Rows from Table TRAINSET

	image	_nObjects_	_Object0_	_Object0_x	_Object0_y	_Object0_width	_Object0_height	_Object1_	_Object1_x	_Object1_y	_Object1_width	_Object1
598	b'\xff\xd8 \xff\xe0\x00 \x10JFIF\x00 \x01\x01 \x00...	2.0	Ball	0.563672	0.566667	0.046094	0.060417	SciSports	0.496484	0.276042	0.075781	
599	b'\xff\xd8 \xff\xe0\x00 \x10JFIF\x00 \x01\x01 \x00...	3.0	Ball	0.162109	0.660937	0.077344	0.076042	SciSports	0.784766	0.347917	0.082031	
600	b'\xff\xd8 \xff\xe0\x00 \x10JFIF\x00 \x01\x01 \x00...	1.0	SAS	0.460938	0.553646	0.403125	0.892708		NaN	NaN	NaN	

elapsed 0.00543s · user 0.00539s · sys 1.2e-05s · mem 1.39MB

Figure 5 The Joined Table Containing Images and Labels

Data Format

In Figure 5, you can see that for record `598`, there are two labeled objects, as shown in the `_nObjects_` column; the first object category, and bounding box location is stored in columns `_Object0_`, `_Object0_x`, `_Object0_y`, `_Object0_width`, `_Object0_height`. The values in location columns in the table are smaller than 1, because they are in YOLO format, which are normalized according to the input image size. YOLO is the recommended format since it is easier to do data augmentation with it.

Visualize the Images and Labels

To check if your label is correct visually, you can use the `extractDetectedObjects` action to extract the object location/category and generate images with the bounding boxes, class names, and score values (when the table is the output of the `dlscore` action), annotated on the image.

```
s.image.extractdetectedobjects
(casout={'name':'trainSetAnnoted','replace':True},
 coordType='Yolo', maxobjects=30, table=s.CASTable('trainSet'))
```

Figure 6 shows the annotated images generated.

Figure 6 Images and Annotations: The Bounding Boxes and Categories

BUILDING THE MODEL ARCHITECTURE

The Backbone Network

Both YOLO and Faster R-CNN Object Detection model need a backbone network to extract features from the images. The backbone network typically is a well-known network used for image classification, for example, ResNet, VGG16, Darknet, and so on. The backbone network usually consists of the data layer, many convolutional layers, batch normalization and pooling layers.

Table 2 shows how you connect to a CAS server, load the action sets needed, create a CNN model with name `TinyYOLO` and add layers to build the backbone network for the model. Only the first few layers and the last layer for the backbone network is shown for simplicity.

```
import swat  # The python interface to SAS Cloud Analytic Services (CAS).

s = CAS('cas04.unx.sas.com', 29990)    # connect to the CAS server
s.loadactionset('image')               # load the image action set
s.loadactionset('deepLearn')           # load the deep learning action set

s.buildModel(model=dict(name='TinyYOLO',replace= True),type=CNN')
s.addLayer(model=modelName, name='data',
          layer=dict(type='input', nChannels=3,width=imgWidth,
           height = imgHeight, scale = 1.0/255))
s.addLayer(model=modelName, name='conv1',
           layer=dict(type='convolution', nFilters=16, width=3, height=3,
                 stride=1, includeBias=False, std=1e-1, act='identity'),
           srcLayers = ['data'])
s.addLayer(model=modelName, name='bn1',
           layer=dict(type='batchnorm', act='leaky'),
           srcLayers = ['conv1'])
s.addLayer(model=modelName, name='pool1',
```

```
            layer=dict(type='pooling',width=2, height=2,stride=2, pool='max'),
            srcLayers = ['bn1']
……
s.addLayer(model=modelName, name='conv9',
            layer=dict(type='convolution', nFilters=125,
                  width=1, height=1,    # filter width and height
                  stride=1, includeBias=False, std=1e-1, act='identity'),
            srcLayers = ['bn8'])
```

Table 2 Code Snippet to Build the Backbone Network for YOLO Object Detection Model

The YOLO Detection Layer

Table 3 shows how to add the YOLO detection layer following the last layer of the backbone network, and some typical parameters used. In the last convolutional layer `conv9`, the width and height of the output feature map should both equal to `gridNumber (13)`, and the depth (`nFilters`) should be equal to:

```
predictionsPerGrid * (classNumber + coordNumber + 1),
```

in which `gridNumber`, `predictionsPerGrid`, `classNumber` are parameters in the detection layer, and `coordNumber` is equal to 4, which is the number of values needed to represent a rectangle bounding box. Anchors are given directly here, which is pre-calculated using K-means algorithm. DLPy provides a function helping you to calculate proper anchors from a given data set.

```
s.addLayer(
    model = modelName,
    name = 'detection0',
    layer = dict(
        type = 'detection',
        detectionModelType = "YOLOV2",
        classNumber = 20,
        gridNumber = 13,
        predictionsPerGrid = 5,
        anchors=(1.08,1.19,3.42,4.41,6.63,11.38,9.42,5.11,16.62,10.52),
        coordType = "YOLO",
        detectionThreshold = 0.3,
        iouThreshold = 0.45,
    ),
  srcLayers = ['conv9']
 )
```

Table 3 YOLO Detection Layer

The Faster R-CNN Region Proposal and Object Detection

The Faster R-CNN network architecture is a little bit more complicated. It consists of a CNN backbone network, followed by several parts:

1. A Region Proposal Layer and two special convolutional layers preceding it,
2. A Region Pooling Layer,
3. Several layers of fully connected layers to generate data for the final FastRCNN layer
4. the FastRCNN layer.

The major code components are shown in Table 4 and Table 5. In Table 4, You can see how the `rpn_score` layer's feature map depth (`nFilters`) is related with some parameters of

Region Proposal Layer. The variable `OrigAnchorNum` represents the number of anchors used in the Region Proposal Layer on each pixel on its input feature map. The actual anchors (in this example 3*3=9 anchors on each pixel) are generated according to these parameters: `baseAnchorSize, anchorScale, anchorRatio` in three steps:

1. Generate a base square anchor with width and height equal to `baseAnchorSize` (in number of pixels in the original input image scale) and centered around the first pixel.
2. Generate a number of anchors with different aspect ratios listed in `anchorRatio`, and with the same area as the base anchor.
3. From each anchor obtained from step 2, generate a number of anchors by multiplying the anchor width and height with the value in the array `anchorScale`.
4. Replicate the anchors generated from the steps 1-3 by shifting to each pixels in the feature map.

```
#Create a CNN model, and add layers for the VGG16 backbone network
#Assuming the last layer of the backbone network has the name 'conv5_3'
……
modelName = 'FasterRCNNModel';
Add_VGG16_FELayers(s, modelName, width=1000, height=496)
……
# Add two additional convolutional layers to extract features for the
Region Proposal Layer

nclasses = 2; # Region Proposal Layer only has 2 classes:
object/background
anchorScaleV=[8,16,32];      # Anchor size multiples
anchorRatioV=[0.5,1,2];      # Anchor aspect ratio
OrigAnchorNum = len(anchorScaleV) * len(anchorRatioV)

s.addLayer(
    model= modelName, name= 'rpn_conv_3x3',
    layer = dict(type='convolution',nFilters=512, width=3,
                 height=3, stride=1, act ='relu'),
    srcLayers = ['conv5_3'])

s.addLayer(
   model=modelName, name='rpn_score',
   layer=dict(type='convolution',
              nFilters = (nclasses + 4) * OrigAnchorNum,
              width=1, height=1, stride=1, act = 'identity'),
   srcLayers = ['rpn_conv_3x3'])

# Add the region proposal Layer
s.addLayer(
   model = modelName,
   name = 'rois',
   layer = dict(
       type = 'REGIONPROPOSAL',
       act = 'identity',
       coordType='COCO',
       baseAnchorSize = 16,
       anchorNumToSample = 256,
       anchorScale=anchorScaleV,
       anchorRatio=anchorRatioV,
```

```
        ),
        srcLayers = ['rpn_score']
    )
```

Table 4 The Region Proposal Layer and its Feature Extraction Layers

In Table 5, the `roipooling` layer is added with two source layers:

- the `conv5_3` layer, which is the last layer of the backbone network,
- the `rois` layer, which is the region proposal layer.

The order of two source layer defines their usage here.

After two additional fully connection (FC) layers, the output of `cls_score` layer is used to provide data to the final FastRCNN layer for classification of the object in the RoI, and the output of `bbox_pred` layer is used for object bounding box regression in the RoI. You can see how the output size `n` of the FC layers is related with the number of object categories.

The last layer for this model is the FastRCNN layer, which has three source layers; they are in order the FC layer with classification data, and FC layer with bounding box regression data, and the Region Proposal layer.

```
classNum = 20;  # Number of Object Categories in the Model

s.addLayer(model=modelName, name='pool5',
        layer = dict(type='roipooling', poolWidth=7, poolHeight=7),
        srcLayers = ['conv5_3', 'rois']
)
s.addLayer(model=modelName, name='fc6',
        layer = dict(type='fullconnect', n=4096, act='relu'),
        srcLayers = ['pool5'])
s.addLayer(modelName, name='fc7',
        layer = dict(type='fullconnect', n=4096, act='relu'),
        srcLayers = ['fc6'])
s.addLayer(modelName, name='cls_score',
        layer = dict(type='fullconnect', n=(classNum+1), act='identity'),
        srcLayers = ['fc7'])
s.addLayer(modelName, name='bbox_pred',
        layer = dict(type='fullconnect',
              n=4*(classNum+1),  # The +1 is for the background category
              act='identity'), srcLayers = ['fc7'])
s.addLayer(
        model = modelName,
        name = 'fastrcnn',
        layer = dict(
            type = 'fastrcnn',
            nmsIouThreshold = 0.3,
            detectionThreshold = 0.8
        ),
        srcLayers = ['cls_score', 'bbox_pred', 'rois'])
```

Table 5 The ROI (Region of Interest) Pooling Layers, and FastRCNN Layers

TRAIN THE MODEL

After you prepared the images, the labeled data, and defined the model DAG as shown in the previous sections, you can now start to train the object detector. Here we use a tiny YOLO detector to show the process.

You can use the action `dlTrain` to train the detector. In the example listed in Table 6, the model DAG is `TinyYOLO`, which we built before using actions `buildModel` and `addLayer`. The training process uses data from the CAS table `trainSet`. The actual images and labels are read from different columns in the table, and the column names are specified in the `dataspecs` field.

This example uses pre-trained weights `yolov2InitdWeights_tiny` and continues to optimize on it. The final weights are saved in the CAS table `yolov2TrainedWeights_tiny`". The `optimizer` defines the algorithms used to search for the best solution while training the network. For details about the `optimizer`, you can refer to the *SAS® Visual Data Mining and Machine Learning DOC*.

```
# Define the optimizer
optimizer=dict(miniBatchSize=10, logLevel=3,debugLevel=2, maxEpochs=10,
                algorithm=dict(method='momentum',# momentum=0.9,
                                clipGradMax=100, clipGradMin=-100,
                                learningRate=0.001, lrpolicy='step',
                                stepsize=20, gamma=0.9)
                              )

 # Train the network
 r = s.dlTrain(table=dict(name='trainSet'),
            model = 'TinyYOLO',
            nThreads=1,
            gpu=1,
            initWeights=dict(name = 'yolov2InitWeights_tiny'),
            modelWeights=dict(name='yolov2TrainedWeights_tiny',
                              replace=True),
            dataspecs=[
                   dict(type='IMAGE', layer='data', data=inputVars),
                   dict(type='OBJECTDETECTION', layer='detection0',
                        data=targets)
                   ],
            optimizer=optimizer,
            forceEqualPadding = True,
            seed=13308
          )
```

Table 6 Invoking dlTrain to Train the Model

DataSpecs for the Detection Layer

In the `dlTrain` action in Table 6, the `dataSpecs` field specifies the names of the columns where the data needed for the layer is stored.

In the `dataspecs` statement, the variable `targets` and `inputVars` are two column name lists whose values are populated in Table 7. It is clearer if you look at the printed-out values of the variables in Table 8.

```
# Define the inputVars and targets that needed in dataspec in dlTrain
inputVars = [];
inputVars.insert(0, '_image_');
targets = ['_nObjects_'];
for i in range(0,10):
    targets.append('_Object%d_'%i);
    for sp in ["x", "y", "width", "height"]:
        targets.append ('_Object%d_%s'%(i, sp));
```

```
print ("targets")
print (targets);
print ("inputVars");
print (inputVars);
```

Table 7 Code to Generate the Variables Used in Dataspec

```
targets
['_nObjects_', '_Object0_', '_Object0_x', '_Object0_y', '_Object0_width',
'_Object0_height', '_Object1_', '_Object1_x', '_Object1_y',
'_Object1_width', '_Object1_height', '_Object2_', '_Object2_x',
'_Object2_y', '_Object2_width', '_Object2_height', '_Object3_',
'_Object3_x', '_Object3_y', '_Object3_width', '_Object3_height',
'_Object4_', '_Object4_x', '_Object4_y', '_Object4_width',
'_Object4_height', '_Object5_', '_Object5_x', '_Object5_y',
'_Object5_width', '_Object5_height', '_Object6_', '_Object6_x',
'_Object6_y', '_Object6_width', '_Object6_height', '_Object7_',
'_Object7_x', '_Object7_y', '_Object7_width', '_Object7_height',
'_Object8_', '_Object8_x', '_Object8_y', '_Object8_width',
'_Object8_height', '_Object9_', '_Object9_x', '_Object9_y',
'_Object9_width', '_Object9_height', '_Object10_', '_Object10_x',
'_Object10_y', '_Object10_width', '_Object10_height']
inputVars
['_image_']
```

Table 8 Output of the Print Statement: Values of the Dataspec Variables

Inside the dataSpecs statement in Table 6, the statement:

```
dict(type='IMAGE', layer='data', data=inputVars)
```

tells the training process that the input layer named `data` uses image data, and the name of the column containing the image is represented by the variable `inputVars`, which in this case means the image data needed is in the column `_image_` in the input CAS table `trainSet`.

The statement

```
dict(type='OBJECTDETECTION', layer='detection0', data=targets)
```

tells that the detection layer (with name `detection0`) uses data of type `OBJECTDETECTION`, which consists of a set of columns in the input table.

Data type `OBJECTDETECTION` defines the meaning of each field in the list `targets` as following:

- The number of labeled objects for each image is stored in a column whose name is given in the first string item in the list `targets`, in this example, in the column named `_nObjects_`;
- Each labeled object in the image uses five columns, whose names are in five consecutive items in the list, for example, in the column with names `_Object0_`, `_Object0_x`, `_Object0_y`, `_Object0_width`, `_Object0_height`
- The order, not the name, of the five consecutive items, determines the usage of the columns. Specifically, the first item points to the column for the object category, and the 2-5 items, if in YOLO format, points to the columns for the x, y position,

and width and height of the object bounding box in order respectively.

Monitoring the Training Process

Table 9 shows some information you can see in the training process. The Fit Error currently is calculated as an average of the value 1-IOU (Intersection over Union) for all images; the IOU for each image is the average IOU for all the *labeled object v.s. best matching prediction pairs* in the image, regardless of whether the prediction is selected as one of the final detections or not.

```
WARNING: Only 1 out of 2 available GPU devices are used.
NOTE:  The Synchronous mode is enabled.
NOTE:  The total number of parameters is 15861648.
NOTE:  The approximate memory cost is 357.00 MB.
NOTE:  Loading weights cost       0.00 (s).
NOTE:  Initializing each layer cost       1.47 (s).
NOTE:  The total number of threads on each worker is 1.
NOTE:  The total minibatch size per thread on each worker is 10.
NOTE:  The maximum minibatch size across all workers for the synchronous
mode is 10.
NOTE:  Epoch           Learning Rate     Loss    Fit Error      Time (s)
NOTE:          0           0.001       44.367     0.7135         0.41
NOTE:          1           0.001       16.287     0.6829         0.40
NOTE:          2           0.001       10.311     0.6061         0.39
NOTE:          3           0.001       7.0372      0.542         0.40
NOTE:          4           0.001       6.2692     0.4923         0.39
NOTE:          5           0.001       5.3297     0.4786         0.39
NOTE:          6           0.001       5.1382     0.4639         0.40
NOTE:          7           0.001       4.9569     0.4291         0.41
NOTE:          8           0.001       4.5718     0.3865         0.40
NOTE:          9           0.001       4.3239     0.3875         0.40
NOTE:  The optimization reached the maximum number of epochs.
NOTE:  The total time is       3.99 (s).
```

Table 9 Monitoring the Training Process

It is an art to train a deep neural network. For object detection network like this, you can use pretrained weights that are trained on general publicly available classification data set like IMAGENET, since they have a huge amount of data. After that, you can transfer the weights into the detection network, and train it with your specific data set. When training a new model, always start with a small sample of the data and try to overfit it.

The L2 Norm of the pre-trained weights should be small, otherwise, it is an indication that the model lacks generalization capability. It is recommended to use L2 Norm and `randomMutation` to prevent overfitting. If L2 Norm is set, the value should decrease and be small during training.

During the detection network training, it is usually a good practice to start with a small learning rate, and after a few epochs, increase the rate by 10~50 times.

SCORE USING TRAINED WEIGHTS

Scoring Using trained weights is similar with other deep learning tasks. In the following scripts, the scoring results are saved in the CAS table `detections`.

```
s.dlscore(model='TinyYOLO',
    initWeights='yolov2TrainedWeights_tiny',
    table = 'scoringSet',
    copyVars=['_path_', '_image_'],
```

```
    nThreads=10,
    miniBatchSize=1,
    casout={'name':'detections', 'replace':True}
)
```

You can use the `extractDetectedObjects` action to extract and display the detection results; the bounding box, the object category, and the scored values are all added onto the image. Figure 7 shows some examples of such annotated images.

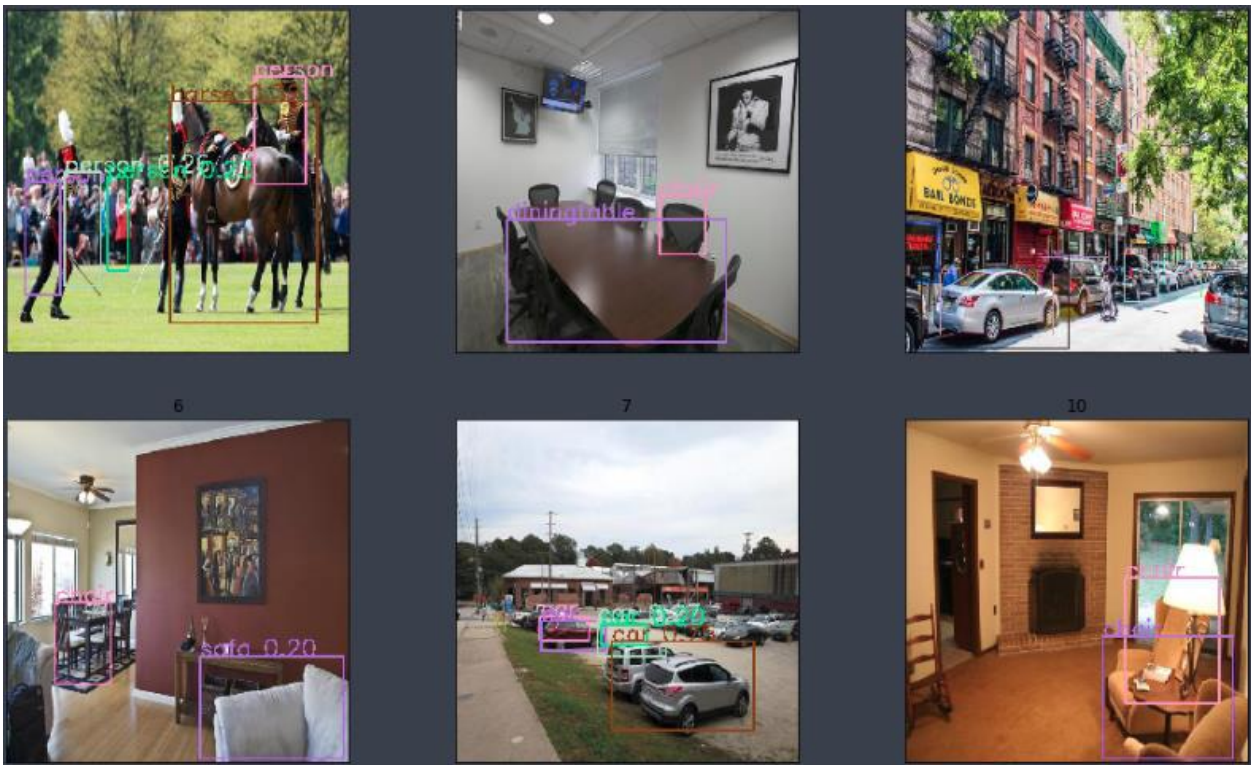


Figure 7 Score Results Displayed as Annotated Images

SEMANTIC SEGMENTATION

Except locating objects in images, analysis of the images at pixel level is useful and widely used to solve many real-world problems, especially in areas such as self-driving, biomedical diagnosis, and so on, as illustrated in Figure 8.

Figure 8 Application of semantic segmentation includes biomedical 3-D image segmentation, self-driving, super resolution, and so on. Images are from:

https://arxiv.org/pdf/1803.08691.pdf,

https://www.kaggle.com/c/cvpr-2018-autonomous-driving/overview

https://paulvanderlaken.com/2017/11/23/super-resolution.

Image semantic segmentation is one of the techniques to understand an image at pixel level. Specifically, it attempts to partition the image into semantically meaningful parts, and to classify each part into one of the pre-defined classes. That is, each pixel in the image is assigned to an object class as shown in Figure 9.

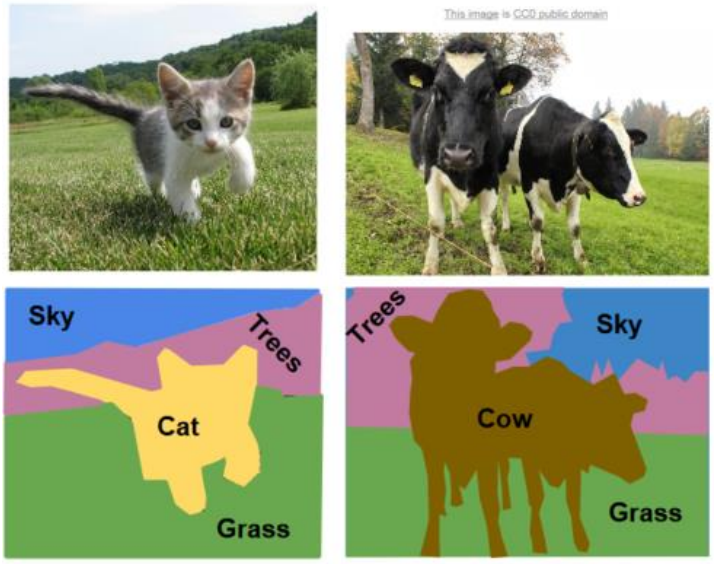


Figure 9 An example of semantic segmentation. One of the four pre-defined labels is given to each pixel in the image to show the boundaries and shape of each object. Image is from https://www.analyticsvidhya.com/blog/2017/11/heart-sound-segmentation-deep-learning.

There are many promising semantic segmentation models, including FCN [9], U-Net [10], SegNet [11], and DeepLab [12]. This paper focuses on U-Net model, which consists of an encoding path to capture context, and a symmetric decoding path that enables pixelwise prediction. This paper also introduces the semantic segmentation feature in SAS Deep Learning through a practical example.

MODEL SPECIFICATION

Fully connected (FC) layers connect every neuron in one layer to every neuron in another layer and are widely used in traditional neural networks to flatten the matrices and to classify the images. However, it fixes the dimension and throws away the spatial structure of the layers. Since for semantic segmentation the inference is at pixel level, it is crucial to maintain the dimensional structure, and naturally the FC layers are replaced by fully convolutional layers.

Based on this idea, the U-Net model contains two parts: the down-sampling encoding part, including convolution layers and pooling layers, that gradually reduces the spatial dimension of the input images, and the up-sampling decoding part, including convolution layers and transpose convolution layers, the recovers the object details. In order to inherit localization information from the encoding process, concatenation layers are also used in the model, as shown in Figure 10. This model is named after its U-shape, as the encoding part and decoding part are symmetric. Starting from the bottleneck layer, which in this case is the 8*8*1024 layer at the bottom of the U-shape, the up-sampling process is the reversed image of the down-sampling process, with pooling layers replaced by transpose convolution layers.

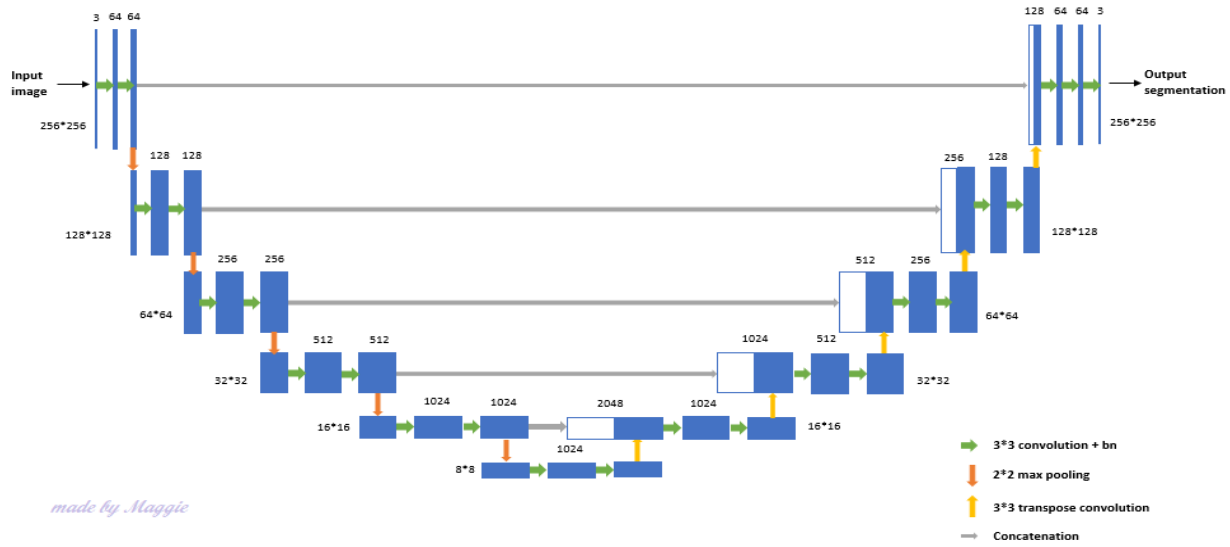


Figure 10 U-Net architecture. The input layer contains 256*256 color images. The resolution of bottleneck layer is 8*8. Each blue box is a feature map of denoted size.

GROUND TRUTH FORMAT

Both image and wide format data are supported in the segmentation model. If using the wide format, each column represents the value of one pixel of the input image. Specifically, the first column gives the class label of the top left pixel (0, 0), the second column represents the second pixel (0, 1). The column for the last pixel in the first row of the image is followed by that of the first pixel in the second row (1, 0) of the image. The last column gives the class of the bottom right pixel of the image. The values could be either numeric (0, 1, 2, …) or categorical (people, car, background, …).

If using image type ground truth for pixel-wise classification, then the pixel values should be an integer no more than the number of classes. For example, if there are four different classes in the image, then each pixel of the ground truth should be a number in [0, 1, 2, 3].

SOCCER PLAYER DATA SET

The data set contains 170 256*256 color images and annotations as shown in Figure 11. They are divided into three parts: training (70%), validation (20%) and testing (10%).

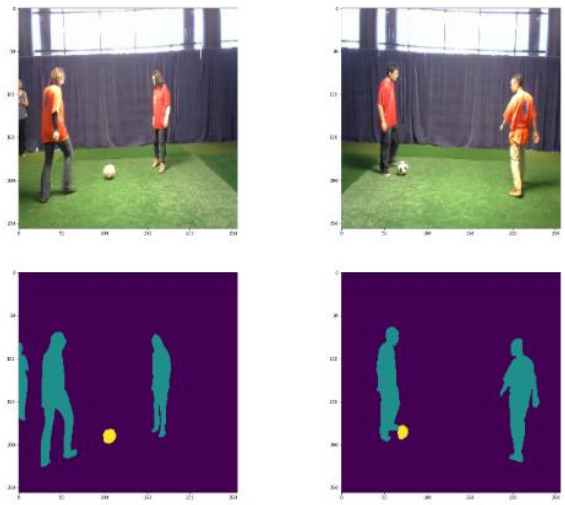


Figure 11 The data set contains 170 color images and annotations. Three classes are pre-defined: soccer player, ball, and background.

Image type ground truth is used for this data set as in Figure 12. Specifically, the data would contain two columns of images, _image_ and _labels_. The first is the input, which contains images of 256*256*3 taking values between [0, 255]. The second is the ground truth, which are 256*256*1 images taking values in [0, 1, 2], representing their three classes: soccer player, ball, and background.

	Column	ID	Type	RawLength	FormattedLength	NFL	NFD
0	_filename_0	1	varchar	27	27	0	0
1	_image_	2	varbinary	118999	118999	0	0
2	_label_	3	varchar	0	1	0	0
3	_id_	4	int64	8	12	0	0
4	labels	5	varbinary	3173	3173	0	0
5	idx	6	double	8	12	0	0

Figure 12 Columns in the data set.

BUILDING MODEL DAG

The model DAG is built using DLPy, an open-source, high-level Python package for deep learning. An example of the syntax is given below.

```
Inputs = InputLayer(3, 256, 256, scale = 1.0 / 255,
         random_mutation='random', name='InputLayer_1')
conv1 = Conv2d(64, 3, act = 'identity', init=init)(inputs)
bn1 = BN(act = 'relu')(conv1)
conv1 = Conv2d(64, 3, act = 'identity', init=init)(bn1)
bn1 = BN(act = 'relu')(conv1)
pool1 = Pooling(2)(bn1)
……
tconv7 = Transconvo(1024, 3, stride = 2, act='relu', padding = 1,
output_size = (16, 16, 1024), init=init)(bn6)
merge7 = Concat(src_layers = [bn5, tconv7])
conv7 = Conv2d(1024, 3, act = 'identity', init=init)(merge7)
bn7 = BN(act = 'relu')(conv7)
conv7 = Conv2d(1024, 3, act = 'identity', init=init)(bn7)
bn7 = BN(act = 'relu')(conv7)
……
conv12 = Conv2d(3, 3, act = 'relu', init=init)(bn11)
seg1 = Segmentation(name='Segmentation_1', act='softmax',
       error='entropy')
```

Since the tensor dimension of the segmentation layer is entirely inherited from its source layer, the feature map size of its source layer should be equal to that of the ground truth, while the number of channels should be equal to the number of classes. In this case, the output size of layer `conv12` is 256*256*3. The default activation function is softmax and the default error type is cross-entropy.

TRAINING AND SCORING

The model is trained using ADAM algorithm for 60 epochs, with mini-batch size = 10 and number of threads = 1. Sample code for training is as below. Part of the training process is shown in Figure 13.

```
Inputs = InputLayer(3, 256, 256, scale = 1.0 / 255,
         random_mutation='random', name='InputLayer_1')
dataspecs=[dict(type='image', layer='InputLayer_1', data=['_image_']),
```

```
            dict(type='image', layer='Segmentation_1', data=['labels'])]
optimizer = dict(miniBatchSize=10, regL2=0.0005,
                 algorithm=dict(method="adam", lr=2e-4, lrPolicy='step',
                                gamma=0.9, stepSize=10),
                 maxEpochs=60, logLevel=2)
s.dlTrain(model=model_name, table=train, validtable=valid, nthreads=1,
          modelWeights = dict(name = 'seg_weights', replace = True),
          dataspecs=dataspecs,
          optimizer = optimizer)
```

```
NOTE:  The total number of parameters is 81716035.
NOTE:  The approximate memory cost is 5320.00 MB.
NOTE:  Loading weights cost       0.00 (s).
NOTE:  Initializing each layer cost       6.65 (s).
NOTE:  The total number of threads on each worker is 1.
NOTE:  The total mini-batch size per thread on each worker is 10.
NOTE:  The maximum mini-batch size across all workers for the synchronous mode is 10.
NOTE:  Number of input variables:     1
NOTE:  Number of numeric input variables:      1
NOTE:  Epoch Learning Rate        Loss  Fit Error  Validation Loss Validation Error   Time(s)
NOTE:  0        0.0002       6.568e+04     0.4404        7.389e+04          0.4229    13.05
NOTE:  1        0.0002       3.813e+04    0.05841        4.033e+04          0.1464     7.28
NOTE:  2        0.0002       2.581e+04    0.04767         2.97e+04          0.06102    7.27
NOTE:  3        0.0002       1.838e+04    0.03793        1.472e+04          0.02501    7.28
NOTE:  4        0.0002       1.287e+04     0.0249        1.332e+04          0.03527    7.27
NOTE:  5        0.0002            9257    0.02333          1.1e+04          0.03559    7.27
NOTE:  6        0.0002            6680    0.02156             8101          0.02656    7.27
NOTE:  7        0.0002            5073    0.02141             6852          0.03326    7.27
NOTE:  8        0.0002            3645    0.01691             3364          0.01582    7.28
NOTE:  9        0.0002            3088    0.01658             2748          0.01483    7.27
NOTE:  10       0.0002            2507    0.01392             2677          0.01499    7.27
```

Figure 13 Part of the training process

The loss error is the sum of cross-entropy of all pixels, while the fit error is the misclassification rate averaged on all pixels. Both errors are supposed to decrease during training process, as indicated in Figure 13.

For segmentation models, dataspecs must be specified for input layers and segmentation layers to define the data types and columns. In this example, images are used for both input and segmentation layers.

The following code is for scoring on the testing data. The testing output is also given below in Figure 14. For each image, the output table contains the pixel-wise prediction, along with the predicted probability. For example, the first pixel is assigned label 0 with probability higher than 0.99, as shown in columns _DL_PredName0_ and _DL_PredP0_. Visualization of the scoring results can be easily achieved based on the output tables.

```
s.dlscore(modeltable=model_name, initweights='seg_weights', table=test,
nthreads=1, casout=dict(name='output', replace=True))
```

§ ScoreInfo

	Descr	Value
0	Number of Observations Read	22
1	Number of Observations Used	22
2	Misclassification Error (%)	0.758362
3	Loss Error	1307.178

	_filename_0	_DL_PredName0_	_DL_PredP0_	_DL_PredName1_	_DL_PredP1_	_DL_PredName2_	_DL_PredP2_	_DL_PredName3_	_
0	0307_image78.png	0	0.990635	0	0.997753	0	0.999453	0	
1	1358_image00163.png	0	0.990079	0	0.996777	0	0.998342	0	
2	0307_image94.png	0	0.991555	0	0.997786	0	0.999458	0	
3	0307_image23.png	0	0.994385	0	0.999123	0	0.999731	0	
4	0307_image78_flipped.png	0	0.993368	0	0.998213	0	0.999360	0	
5	0307_image24.png	0	0.991580	0	0.997847	0	0.999442	0	
6	0307_image15_flipped.png	0	0.994278	0	0.998573	0	0.999483	0	
7	0307_image17.png	0	0.990696	0	0.997499	0	0.999325	0	
8	0307_image21_flipped.png	0	0.994408	0	0.998811	0	0.999602	0	

Figure 14 Scoring output on testing data.

The scoring mis-classification rate on the testing data is 0.76%, which means out of 65,536 pixels in each image, only less than 500 pixels are miss-labeled. Some of the scoring visualization results are given in Figure 15.

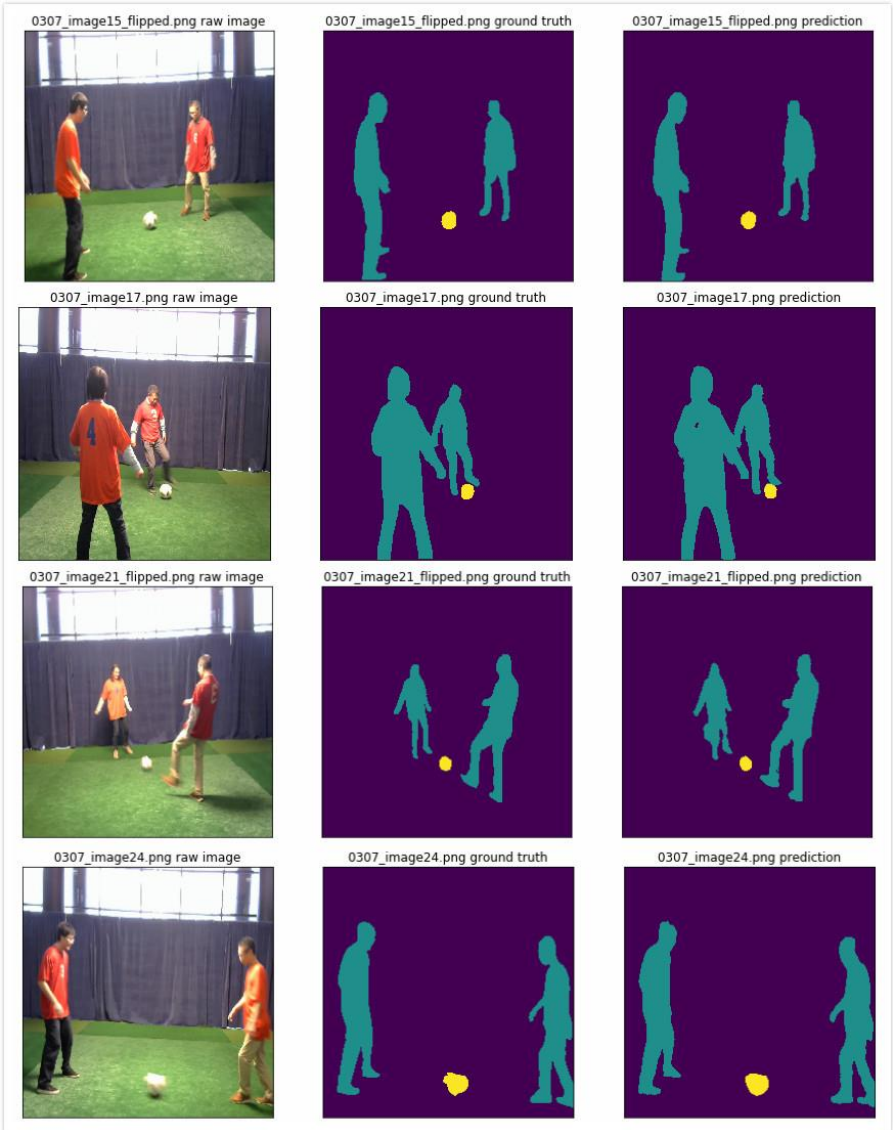


Figure 15 Scoring results visualization. The raw images are shown in the first column, followed by ground truth annotation in the second column. The third column contains predictions.

CONCLUSION

SAS has extended its deep learning toolkit to support object detection and semantic segmentation in its recent release. This new functionality is available through CAS action sets in SAS Visual Data Mining and Machine Learning, as well as in the DLPy open-source project.

With the new extension, SAS deep learning empowers customers to build an end to end solutions to computer vision problems involving tasks of image classification, keypoint detection, object detection, and semantic segmentation.

Specific examples demonstrate how some new layers, when combined with other deep learning, image processing action sets, enable customers to load, explore the data, build the model architecture, train the network, perform the inference, and visualize the results.

Development efforts involving instance segmentation is in progress and will be available to customers in the future.

REFERENCES

[1] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, 2017.

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision* , 2016.

[3] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 1 June 2017.

[4] T. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal Loss for Dense Object Detection," in *IEEE International Conference on Computer Vision*, Venice, 2017.

[5] J. Dai, Y. Li, K. He and J. Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks," in *Neural Information Processing Systems Conference*, 2016.

[6] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature Pyramid Networks for Object Detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, 2014.

[8] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015.

[9] J. E. S. a. T. D. Long, "Fully convolutional networks for semantic segmentation," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. pp. 3431-3440, 2015.

[10] O. P. F. a. T. B. Ronneberger, "U-net: Convolutional networks for biomedical image segmentation," *International Conference on Medical image computing and computer-assisted intervention*, pp. pp. 234-241, 2015.

[11] V. K. A. &. C. R. Badrinarayanan, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence, 39(12), 2481-2495.*, 2017.

[12] L.-C. G. P. I. K. K. M. a. A. L. Y. Chen, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence, 40(4), 834-848.*, 2018.

[13] V. a. F. V. Dumoulin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285 (2016).*.

RECOMMENDED READING

- *SAS® Visual Data Mining and Machine Learning DOC, available at*

  http://support.sas.com/software/products/visual-data-mining-machine-learning/index.html#s1=2

- *DLPy - SAS Viya Deep Learning API for Python, available at*

  https://github.com/sassoftware/python-dlpy

CONTACT INFORMATION

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

Xindian Long
SAS Institute
+1 919-531-2594
Xindian.Long@sas.com

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

Xiangqian Hu
SAS Institute
+1 919-531-1423
Xiangqian.Hu@sas.com

Paper SAS3341-2019

Medical Image Analytics in SAS® Viya® with Applications in the Treatment of Colorectal Cancer Spread to the Liver

Fijoy Vadakkumpadan and Joost Huiskens, SAS Institute Inc.

ABSTRACT

The powerful analytics in SAS® Viya® have been recently extended to include the ability to process medical images, the largest driver of health-care data growth. This new extension, released in SAS® Visual Data Mining and Machine Learning 8.3 in SAS® Viya® 3.4 enables customers to load, visualize, analyze, and save health-care image data and associated metadata at scale. As SAS continues to build on this foundation, several substantially new medical image processing capabilities are planned for future versions of SAS Visual Data Mining and Machine Learning. In particular, these new capabilities will enable customers to perform the following tasks: process generic data files, such as radiotherapy (DICOM-RT) files, under the Digital Imaging and Communications in Medicine (DICOM) standard; process images in a single SAS® Cloud Analytic Services (CAS) action call even when the processing parameters vary from one image to another in the input table; use highly advanced techniques to perform image segmentation; and quantify the size and shape of tissue regions in binary images. This paper demonstrates the new capabilities by applying them to colorectal liver metastases (CRLM) morphometry with computed tomography (CT) scans to assess patients' response to chemotherapy. This study is a collaborative effort between SAS and Amsterdam University Medical Center (AUMC) for improving CRLM treatment strategies.

INTRODUCTION

Imaging has been revolutionizing medicine for decades. From simple 2-D X-rays at the dentist's office to exquisite 4-D ultrasounds of a baby in its mother's womb, it has touched and saved countless lives. Physicians routinely rely on image-based data to diagnose diseases, guide therapeutic procedures, and monitor patients' response to treatment. The benefits of these imaging technologies come with a hefty price, however, since medical image data are notoriously large. As of 2016, over 600 million imaging procedures were performed annually in the US alone, generating millions of terabytes of data. This constitutes over 90% of total health-care data, making imaging the largest driver of health-care data growth. To make matters worse, the extraction of relevant information from medical images, for example, the boundaries of tumors, is typically done manually by health-care professionals using a labor-intensive and subjective process. The sheer volume of the data combined with the extensive cognitive input required for its processing make it extremely challenging for the health-care industry to convert the images into objective insights that can drive decisions. Further, as the volume of medical imaging data continues to rise, radiologists are subject to excessive cognitive workloads, which leads to fatigue and increased risk of medical errors. A typical radiologist combs through thousands of images in a single day, as he or she examines tens of patients, each with hundreds of cross-sectional image slices. Therefore, any automated assistance offered to the radiologists can dramatically improve their lives and the quality of the health care their patients receive.

SAS has a rich history of supporting health and life sciences customers for their clinical data management, analytics, and compliance needs. SAS® Analytics provides an integrated environment for collection, classification, analysis, and interpretation of data to reveal patterns, anomalies, and key variables and relationships, leading ultimately to new insights for guided decision-making. The application of SAS® algorithms has enabled patients to transform themselves from being passive recipients to becoming active participants in their own personalized health care. With the release of SAS® Viya® 3.4, SAS customers can now

extend the analytics framework to take advantage of medical images along with statistical, visualization, data mining, text analytics, and optimization techniques for better clinical diagnosis. Images are supported as a standard SAS data type in SAS® Visual Data Mining and Machine Learning, which offers an end-to-end visual environment for machine learning and deep learning—from data access and data preparation to sophisticated model building and deployment in a scalable distributed framework. It provides a comprehensive suite of programmatic actions to load, visualize, process, and save health-care image data and associated metadata at scale in formats such as Digital Imaging and Communication in Medicine (DICOM), Neuroimaging Informatics Technology Initiative (NIfTI), nearly raw raster data (NRRD), and so on.

The goal of this paper is twofold. First, it provides a comprehensive overview of end-to-end medical image analytics capabilities in SAS® Visual Data Mining and Machine Learning in SAS Viya, with special focus on future releases of this product. Second, it illustrates these capabilities by working through a real-world, clinically significant use case that is part of a collaboration between SAS and Amsterdam University Medical Center (AUMC) aimed at improving treatment strategies for patients with colorectal cancer spread to the liver, known as colorectal liver metastases (CRLM).

END-TO-END BIOMEDICAL IMAGE ANALYTICS IN SAS VIYA

SAS Viya uses an analytic engine known as SAS Cloud Analytic Services (CAS) to perform various tasks, including medical image analytics. Building end-to-end solutions in SAS Viya typically involves assembling CAS actions, which are the smallest units of data processing that are initiated by a CAS client on a CAS server. CAS actions are packaged into logical groups called action sets. At this time, two action sets, Image and BioMedImage, host actions that directly operate on medical imagery.

The Image action set contains two actions that directly operate on medical image imagery: the loadImages action loads biomedical images from disk into memory, and the saveImages action saves the loaded images from memory to disk. These actions support all common biomedical image formats, including the DICOM standard, which is widely used in clinical settings. The BioMedImage action set currently includes three actions, processBioMedImages, segmentBioMedImages, and buildSurface, with two new actions, loadDicomData and quantifyBioMedImages, to be added in future releases. The loadDicomData action is for loading generic DICOM files, including non-image files, into memory. The other actions facilitate preprocessing, segmentation, visualization, and quantification of medical images. At this time, full support is available only for two- and three-dimensional (2-D and 3-D), single-channel medical images in these action sets.

The output produced by some of the actions in the Image and BioMedImage action sets can be used as input to other actions, such as those in action sets for traditional machine learning (ML) and deep learning (DL), to derive insights that inform decisions. Figure 1 presents an end-to-end biomedical image analytics pipeline in SAS Viya. On one end of the pipeline are raw image data and metadata on disk, and on the other end are helpful insights that can inform decisions. The major steps in the pipeline, along with the primary action sets (in italics) that can be used to implement those steps, are displayed in rectangular boxes. The examples of ML and DL action sets include the RobustPCA action set, which performs principal component analysis (PCA), and the DeepLearn action, which performs deep learning. Steps in which new actions or major upgrades to existing actions will be available in upcoming releases of SAS® Visual Data Mining and Machine Learning are highlighted in red. The green arrows signify critical features available in future releases of the Image and BioMedImage action sets to facilitate the use of generic data processing action sets such as DataStep and FedSQL to process tables containing image data. These generic action sets are critical to integrating binary image-based data from various sources

and to processing non-binary image-based metadata while retaining the corresponding binary image-based data.

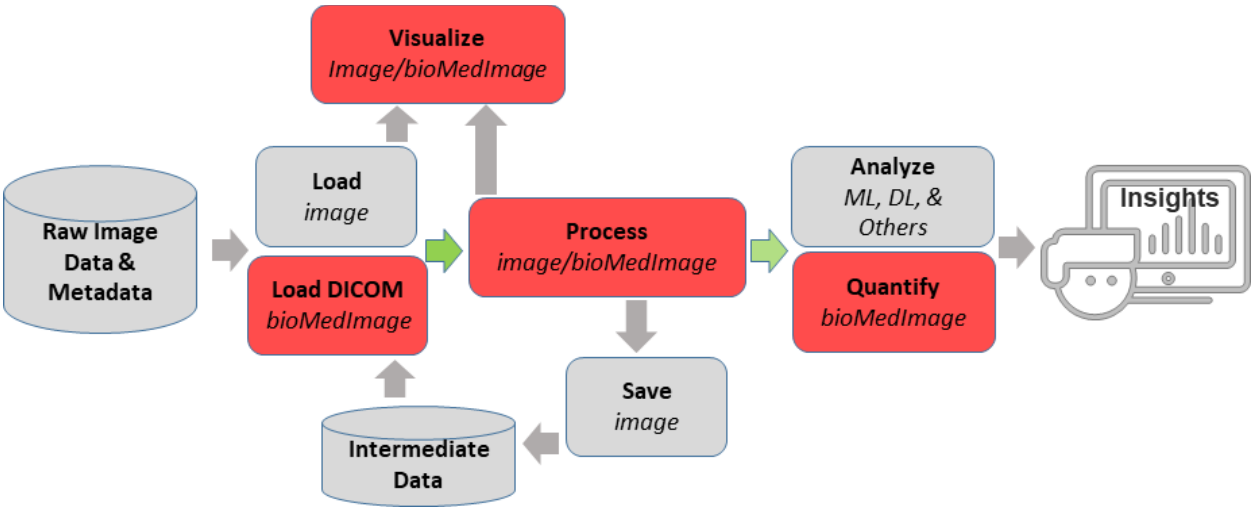


Figure 1. Processing Pipeline for End-to-End Biomedical Image Analytics in SAS Viya

CRLM MORPHOMETRY: AN EXAMPLE USE CASE

This section provides an example for the pipeline shown in Figure 1 by demonstrating how to build an end-to-end solution that can assist with a real-world biomedical image analytics problem, specifically assessment of therapy response in patients with CRLM, based on 3-D computed tomography (CT) images of the patients' torsos. Colorectal cancer is the third most common cancer and the second leading cause of cancer-related deaths worldwide (Bray et al. 2018). Most cancer deaths are the result of progression of metastases, that is, spreading of the cancer. Approximately 50% of colorectal cancer patients will develop metastases to the liver (Donadon et al. 2007). Patients with liver-only colorectal metastases can be treated with curative intent. Complete surgical resection of CRLM is considered the only chance for cure for these patients (Angelsen et al. 2017). Initially unresectable liver metastases can become resectable after downsizing of the lesions by systemic therapy, the main component of which is chemotherapy (Lam et al. 2012). However, there is no consensus regarding the optimal systemic therapy regime.

Accordingly, assessment of patient response to treatment is a crucial feature in the clinical evaluation of systemic therapy. The widely accepted and applied criterion for such assessment is the Response Evaluation Criteria In Solid Tumors (RECIST), which aims to measure the objective change of anatomical tumor size (Eisenhauer et al. 2009). The RECIST assessment is performed by measuring changes in one-dimensional (1-D) diameter of two target lesions before and after therapy. Though RECIST is a clinical standard worldwide, it is highly limited. Firstly, its measurement is manual and labor intensive. Secondly, it is very subjective, as the target lesions, image slices for measurement, and the line segment for lesion diameter measurement are all selected by a radiologist subjectively (Yoon et al. 2016). Finally, RECIST ignores the exquisite 3-D and gray-scale information provided by modern CT scanners. Some of this information has been proven to be significantly associated with pathologic response and overall survival in patients with CRLM (Chun et al. 2009).

This paper demonstrates the medical image analytics capabilities of SAS Viya by using the SAS® Platform to compute new criteria that can potentially assist radiologists with improving assessment of CRLM treatment response. Two different approaches, one based on semi-automatic image segmentation, and the other on automatic object detection with

deep learning, are demonstrated. All client-side source code in this demonstration was written in Python. The SAS Scripting Wrapper for Analytics Transfer (SWAT) package was used to interface with the CAS server, and the Mayavi library (Ramachandran and Varoquaux 2011) was used to perform 3-D visualizations of image-based data.

DATA SELECTION AND PREPROCESSING

All patient data used in this paper was collected as part of the Treatment Strategies in Colorectal Cancer Patients with Initially Unresectable Liver-Only Metastases (CAIRO5) clinical trial (Huiskens et al. 2015) and provided by AUMC. The data consisted of 3-D, abdominal and thoracic transaxial CT images of patients in DICOM format (Figure 2A), and expert radiologist segmentations of liver and lesions in each scan. The in-plane pixel size of the images ranged from 0.6 to 0.8 mm, and the slice thickness ranged from 3 to 5 mm. The expert segmentations were performed semi-automatically using the Philips IntelliSpace Portal software at AUMC, and the resulting 3-D organ contours were stored as DICOM radiotherapy (DICOM-RT) files (Figure 2B). Each patient received a baseline scan before therapy and regular follow-up scans throughout therapy. Most patients received one follow-up scan, with some receiving two. Since the goal of this paper is to demonstrate the capabilities of SAS Viya and not to statistically show clinical significance, only a small set of 10 patients was included in the analyses.

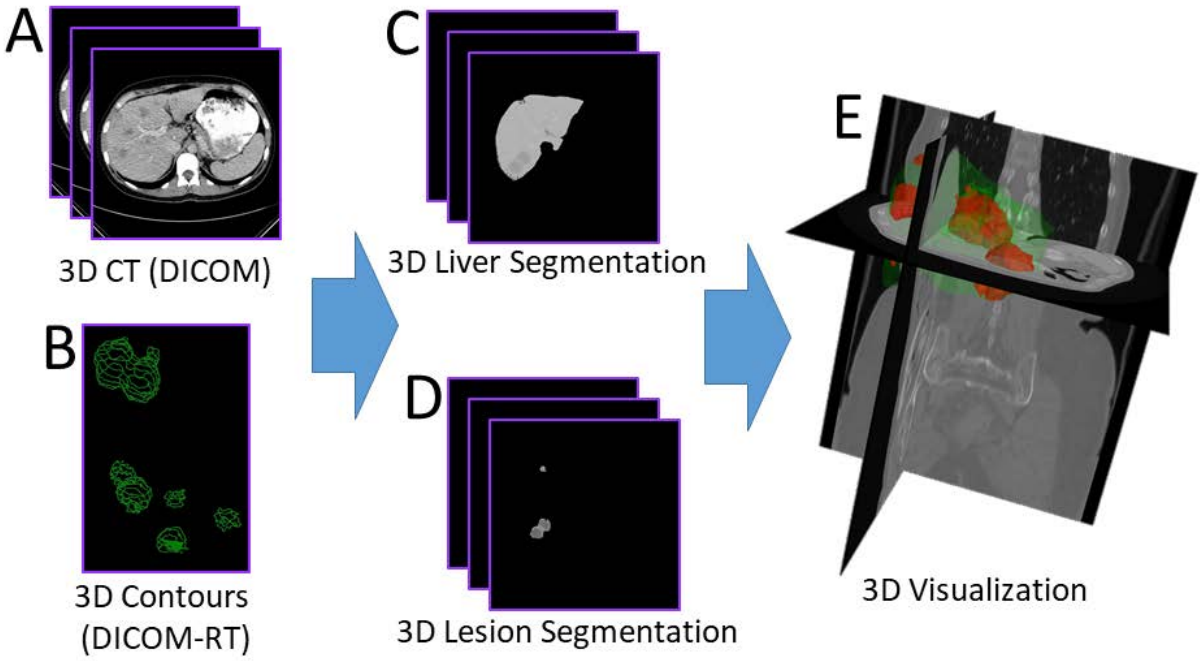


Figure 2. Steps in Data Processing

To preprocess the data, all 3-D images were recursively loaded on the CAS server as illustrated by this code snippet:

```
# DICOM Keywords for images
imsuid = 'SeriesInstanceUID'
impn = 'PatientName'
imad = 'AcquisitionDate'
impa = 'PatientAge'
impx = 'PatientSex'
all_keys_im = [imsuid, impn, imad, impa, impx]
```

```
# Load the DICOM series
imdata = s.CASTable(name='imdata_orig', replace=True)
r = s.image.loadimages(
    path='AUMCImages',
    casOut=imdata,
    recurse=True,
    addcolumns=dict(
        general={'position', 'orientation', 'spacing'},
        dicomattributes=dict(keywords=all_keys_im)),
    series=dict(dicom=True),
    decode=True
)
```

Note that several DICOM attributes of each image, including `SeriesInstanceUID`, were loaded by specifying the `dicomattributes` parameter. Next, the DICOM-RT data were loaded as follows:

```
# DICOM keywords for RT data
rtsuid =
'ReferencedFrameOfReferenceSequence{1}RTReferencedStudySequence{1}RTReferen
cedSeriesSequence{1}SeriesInstanceUID'
rtcsq = 'ROIContourSequence'
rtsdesc = 'SeriesDescription'
all_keys_rt = [rtsuid, rtcsq, rtsdesc]

# Call the new loadDicomData
rtdata = s.CASTable(name='rtdata', replace=True)
r = s.biomedimage.loaddicomdata(
    path='AUMCDicomRt/',
    casOut=rtdata,
    addColumns=dict(keywords=all_keys_rt)
)
```

Here, loadDicomData is an action that will be available in future releases of SAS Visual Data Mining and Machine Learning in the BioMedImage action set. This action can load user-provided attributes from DICOM files in a given directory or file path, including nested sequence attributes. For example, the attribute specified by the variable `rtsuid` in the above snippet specifies multiple nesting levels to access the DICOM series universal identifier (UID) of the image that corresponds to each DICOM-RT file being loaded.

The next step was to merge imdata, the table containing the images, with rtdata, the table containing DICOM-RT data, by using the DataStep action set in CAS, as follows:

```
# Function to convert strings into column names
def col(s):
    s = str.replace(str.replace(s, '{', '_'), '}', '_')
    return '_'+s+'_'

# Create views in preparation for data step
r = s.table.view(
    name='rtview',
    tables=[dict(name='rtdata',
        computedvars={'vccsq', 'suid', 'rtid'},
        computedvarsprogram=
            "length vccsq varchar(*); vccsq="+col(rtcsq)+";"
            "length suid varchar(64); suid="+col(rtsuid)+";"
            "rtid=_id_;",
        varlist={col(rtsdesc)})],
```

```
    replace=True)

r = s.table.view(
    name='imview',
    tables=[dict(name='imdata_orig',
        where='_depth_>1',
        computedvars={'vcimage', 'vcres', 'vcpos', 'vcori', 'vcspa',
                      'adate', 'suid'},
        computedvarsprogram=
            "length vcimage varchar(*); vcimage=_image_;"
            "length vcres varchar(24); vcres=_resolution_;"
            "length vcpos varchar(24); vcpos=_position_;"
            "length vcori varchar(72); vcori=_orientation_;"
            "length vcspa varchar(24); vcspa=_spacing_;"
            "length suid varchar(64); suid="+col(imsuid)+";"
            "adate=input("+col(imad)+", yymmdd8.);",
        varlist={'_id_', '_dimension_', '_imageFormat_', col(impn)})],
    replace=True)

# Merge the tables on DICOM series UID
r = s.datastep.runcode(code="data imrt;"
                       "merge imview(in=a) rtview(in=b);"
                       "by suid;"
                       "if a & b;"
                       "run;")
```

Note that the runCode action in the DataStep action set was performed on views of the image data tables generated by the view action in the table action set, and not directly on the tables. This is because the tables contained binary data columns of type varbinary, such as _image_, which the runCode action does not currently support. The views helped cast the binary data as the character type varchar, which runCode supports. To facilitate processing of the table produced by runCode, actions in the BioMedImage action set have been updated to accept columns of type varchar also for variables where it previously required varbinary columns.

Three-dimensional images of liver (Figure 2C) and lesion (Figure 2D) segmentations were then generated by processing the merged table with the processBioMedImages action in the BioMedImage action set, like so:

```
masks = s.CASTable(name='masks', replace=True)
imrt = s.CASTable(name='imrt', replace=True)
s.biomedimage.processbiomedimages(
    images=dict(table=imrt, image='vcimage',
           resolution='vcres',
           position='vcpos',
           orientation='vcori',
           spacing='vcspa'),
    steps=[dict(stepparameters=dict(steptype='roi2mask',
        roi2maskparameters=dict(roi2masktype='dicomrt_specific',
            roicontoursequence='vccsq',
            pixelintensity='image')))],
    casout=masks,
    decode=True,
    copyvars={'_SeriesDescription_'},
    addcolumns={'position', 'orientation', 'spacing'}
)
```

Here, roi2mask is a new step that will be available in future releases of the processBioMedImages action. This step is capable of processing image-specific DICOM-RT

contour data, as indicated by the 'dicomrt_specific' value of the roi2masktype parameter. The value of 'image' given to the pixelintensity parameter directed the action to retain gray-scale values of pixels that belong to the regions delineated by the DICOM-RT contours. The liver and lesion segmentations generated by the processBioMedImages action can be fed into the buildSurface action in the BioMedImage action set to reconstruct highly detailed surfaces of liver and lesions for 3-D visualization. One such visualization presented in Figure 2E illustrates the exquisite 3-D detail captured by the data used in this paper.

CRITERIA USING IMAGE SEGMENTATION

To compute new criteria in assessing CRLM response that overcome some of the limitations of RECIST, quantifyBioMedImages, a new action that will be available in future releases of SAS® Visual Data Mining and Machine Learning in SAS Viya, was applied to the lesion segmentation images generated in the previous section. This action can compute user-specified quantities or metrics from images. The code to invoke this action was as follows:

```
qdata = s.CASTable(name='qdata', replace=True)
s.biomedimage.quantifybiomedimages(
    images=dict(table=masks.query("find(_SeriesDescription_, 'seg')>0")),
    region='component',
    quantities=[dict(quantityparameters=dict(quantitytype='mean')),
                dict(quantityparameters=dict(quantitytype='content'))],
    labelparameters=dict(labeltype='basic', connectivity='face'),
    casout=qdata)
```

The quantities specified above are mean and content, which directed the action to compute mean CT intensity and volume of each lesion region. Note, the masks table was filtered using information contained in one of the DICOM attributes to select only the lesion segmentation images for these calculations. Also, the combination of region and labelparameters options directed the action to compute the quantities for each connected component (Johnson, McCormick, and Ivanez 2015); that is, lesion region, of each image. The result of the quantifyBioMedImages action was then further processed with the summary action in the Simple action set to compute the total lesion volume and mean lesion intensity of each lesion in each scan of each patient. The results are summarized in Figure 1. Scans 0, 1, and 2 referred to in the legend of the figure were the baseline, first follow-up, and second follow-up scans, respectively. Overall, both the lesion volumes and the mean lesion intensities decreased over therapy. Note that the unit for mean lesion intensity is the Hounsfield unit (HU), the unit of pixel values in CT images.

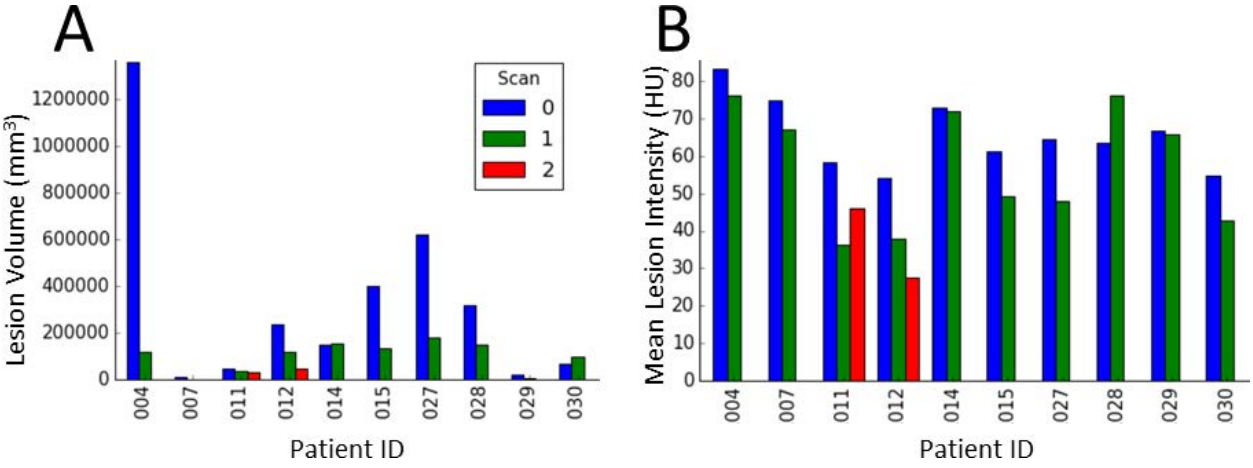


Figure 3. Total Lesion Volumes and Mean Lesion Intensities for Each Scan of Each Patient

CRITERION USING AUTOMATIC OBJECT DETECTION WITH DEEP LEARNING

This section demonstrates preliminary efforts toward using a convolutional neural network-based deep learning object detection model to develop an objective, fully automated surrogate for the RECIST criterion. To prepare the data needed to train the model, the 3-D lesion segmentation images (Figure 4A) were split into individual 2-D slices (Figure 4B) using the export_photo step of the processBioMedImages action in the BioMedImage action set (Vadakkumpadan and Sethi 2018). Then, the bounding box for each 2-D lesion region of each slice was computed (Figure 4C) using the quantifyBioMedImages action, as follows:

```
bbdata = s.CASTable(name='bbdata', replace=True)
s.biomedimage.quantifybiomedimages(
    images=dict(table=masks_exp),
    region='component',
    quantities=[dict(quantityparameters=dict(quantitytype='boundingbox'))],
    labelparameters=dict(labelType='basic', connectivity='face'),
    casout=bbdata)
```

Here, the masks_exp table contained the 2-D slices (Figure 4B). The output table containing the bounding boxes was then merged with corresponding slices from the original 3-D DICOM image (Figure 2A). The merged table contained about 900 rows, each row containing an image slice from one of the 10 patients and bounding boxes of lesions in that slice. Note that only those slices with at least one lesion region was included in this data. These data was then randomly split into training and testing sets of approximately equal size. The YOLOv2 model available in SAS® Visual Data Mining and Machine Learning in SAS Viya was then optimized using the training set to detect CRLM lesions.

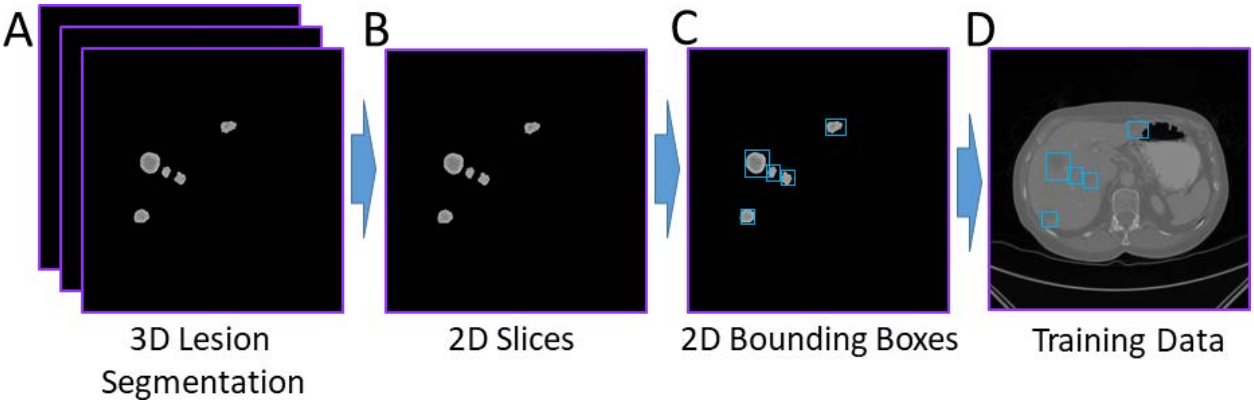


Figure 4. Data Preparation for Training the YOLOv2 Deep Learning Object Detection Model

The model was then scored with the testing set, and the automatically detected lesions were visually examined using the extractDetectedObjects action in the Image action set (Figure 5). It was evident that the model was learning to detect the lesions. At the same time, the model was not perfect, since it missed many lesions, for example, those pointed to by the yellow arrows inFigure 5. To compute a single lesion size metric for each 3-D patient scan, for each detected 2-D bounding box in that scan, the volume of a disc with diameter equal to the average side length the bounding box and thickness equal to the scan's slice thickness was calculated. The volumes of all such discs were then totalled, and the diameter of a sphere with volume equal to this sum was computed. Figure 6 presents this new automatic lesion size metric for each scan of each patient. It is clear from the figure that this new metric captures the reduction in lesion size over therapy, and therefore can be an objective, fully automated surrogate for the RECIST criterion. Note that two of the scans, specifically scan 2 of patient 11 and scan 1 of patient 29, are missing in this plot. This is because the deep learning model failed to detect any lesions in those scans.

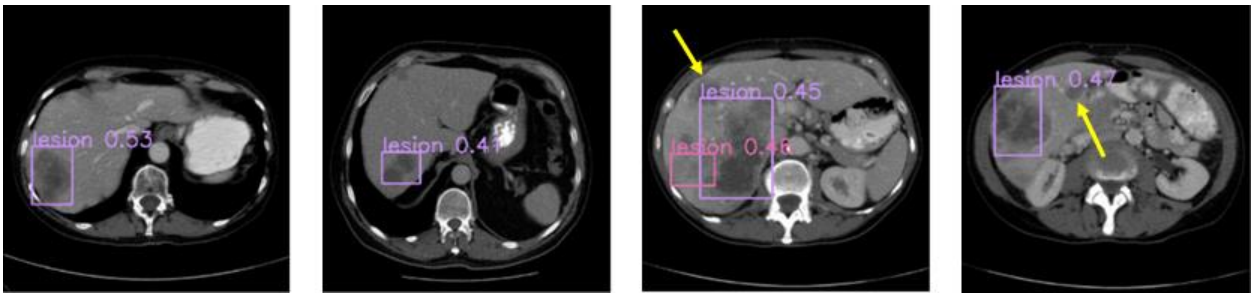


Figure 5. Examples Results from Automatic Detection of CRLM Lesions Using the YOLOv2 Model in This Paper

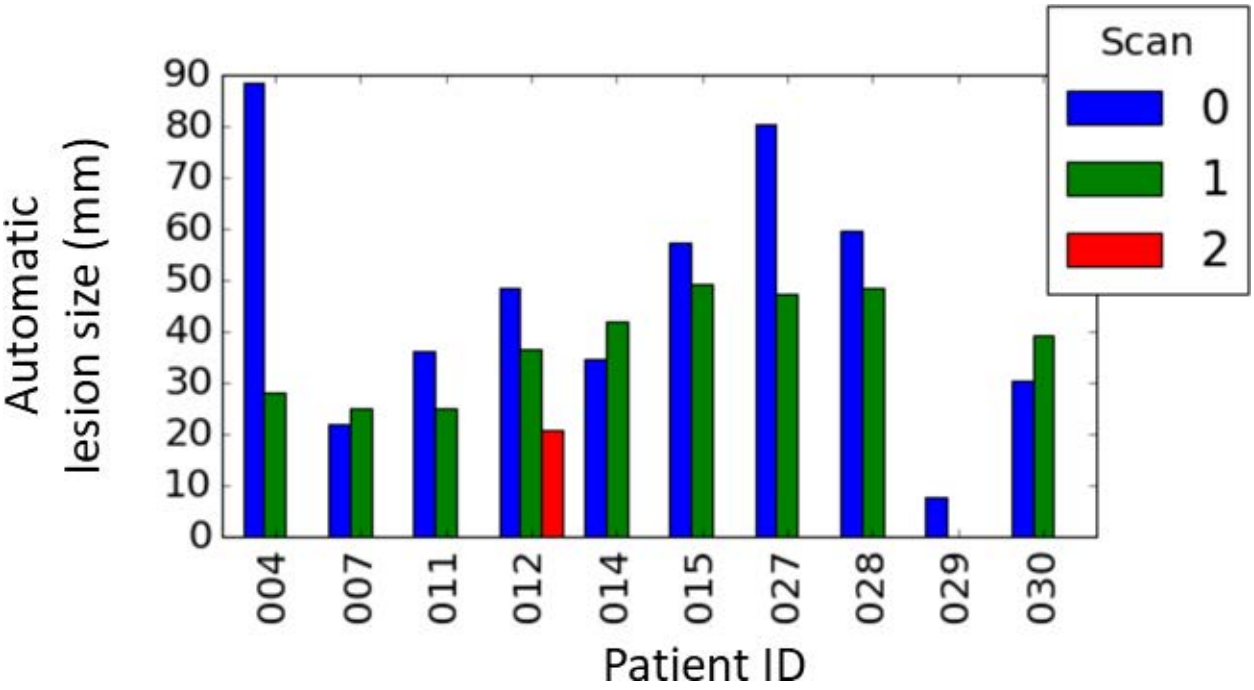


Figure 6. Results from Automatic Lesion Size Measurement Using YOLOv2 Object Detection

DISCUSSION

This paper describes the various SAS Viya components for medical image analytics and to provide illustrations of how to assemble those components to solve real-world problems. Two CAS action sets, Image and BioMedImage, currently host all actions that directly operate on medical imagery. CRLM treatment response assessment is used as an example to illustrate how to assemble these actions in combination with other SAS Viya actions to build complex pipelines that convert raw medical image data and annotations into insights that can help address clinically significant problems. Two image analytic approaches, one using semi-automatic image segmentation and the other using automatic object detection with deep learning, are demonstrated.

The CRLM response assessment metrics presented in this paper can potentially overcome important limitations of the RECIST criterion. The focus of the semi-automatic segmentation approach was to incorporate information ignored by RECIST, while that of the object detection method was to provide a criterion that was fully automated and objective. Specifically, the former approach used the 3-D (Figure 2A) and gray-scale (Figure 4B) information from all lesion regions. In contrast, RECIST is restricted to the usage of simple 1-D diameter measurements made on two target lesions. The object detection approach

provided a lesion size measure (Figure 6) similar to RECIST, but without the subjectivity or labor that is part of RECIST. Such an objective and automated approach, when implemented in a clinic, will help radiologists use their time efficiently and make more consistent decisions across patients. Our preliminary analyses have found quantitative evidence demonstrating that the criteria presented in this paper strongly correlate with, and contain information complementary to, the RECIST measure. These analyses are, however, beyond the scope of this paper and therefore will be published elsewhere.

The CRLM response criteria presented in this paper have some limitations. First, the image segmentation approach involves labor-intensive delineations of the liver and lesion regions from 3-D CT images. However, this limitation will be overcome in the future by using deep U-net style deep learning models (Christ et al. 2016) that will be available in future releases of SAS® Visual Data Mining and Machine Learning. Training of such models was the primary purpose of expert delineations of the liver. Second, the YOLOv2 model used in the object detection approach had limited accuracy. But achieving highly accurate lesion detection was not the goal in this paper. The objective for the model was to attain an accuracy that was sufficient to provide a metric that strongly correlated with the RECIST criterion. Finally, the images in the testing set used to evaluate YOLOv2 deep learning model in this paper strongly correlated with those in the training set since slices from the same 3-D image were included in both sets. However, the goal of this paper was to describe the medical image analytics component of SAS Viya and to demonstrate its potential for solving a clinically significant image analytics problem. The goal was not to develop a model that can be deployed in the clinic.

CONCLUSION

The medical image analytics extension of SAS Viya, available in SAS® Visual Data Mining and Machine Learning, enables customers to load, visualize, process, and save health-care image data and associated metadata at scale. Specific examples provided in this paper demonstrate how the new action sets, when combined with other data analytic capabilities available in SAS Viya, such as deep learning, empowers customers to assemble end-to-end solutions to significant, image-based health-care problems. Upcoming releases of SAS Viya will build on the foundation that this paper demonstrates. These future development efforts will include additional capabilities to process images with image-specific parameters, and to compute more complex quantities from images such as histograms. Also, the BioMedImage action set will be expanded by adding dedicated actions that perform binary operations on images, such as addition and masking.

REFERENCES

Angelsen, J. H., et al. 2017. "Population-based study on resection rates and survival in patients with colorectal liver metastasis in Norway." *The British Journal of Surgery*, 104(5):580-589.

Bray, F., et al. 2018. Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA Cancer J Clin*, 68(6):394-424.

Christ, P. F., et al. 2016. "Automatic Liver and Lesion Segmentation in CT Using Cascaded Fully Convolutional Neural Networks and 3D Conditional Random Fields." In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, ed. Ourselin, S., et al, 415-423. Cham: Springer.

Chun, Y. S., et al. 2009. "Association of computed tomography morphologic criteria with pathologic response and survival in patients treated with bevacizumab for colorectal liver metastases." *JAMA*, 302(21):2338-2344.

Donadon M. et al. 2007. "New paradigm in the management of liver-only metastases from colorectal cancer." *Gastrointestinal Cancer Research*, 1(1):20-27

Eisenhauer E.A., et al. 2009. "New response evaluation criteria in solid tumours: revised RECIST guideline (version 1.1)." *European J Cancer*, 45(2):228-247.

Huiskens J., et al. 2015. Treatment strategies in colorectal cancer patients with initially unresectable liver-only metastases, a study protocol of the randomised phase 3 CAIRO5 study of the Dutch Colorectal Cancer Group (DCCG)." *BMC Cancer*, 15:365.

Johnson, H. J., M. McCormick, and L. Ivanez. 2015. The ITK Software Guide Book 1: Introduction and Development Guidelines – Volume 1. New York: Kitware, Inc.

Lam, V. W., et al. 2012. "A systematic review of clinical response and survival outcomes of downsizing systemic chemotherapy and rescue liver surgery in patients with initially unresectable colorectal liver metastases." *Ann Surg Oncol*, 19(4):1292-1301.

Ramachandran, P., and G. Varoquaux. 2011. "Mayavi: 3D Visualization of Scientific Data." *IEEE*, 13(2):40-51.

Vadakkumpadan, F., and S. Sethi. 2018. "Biomedical Image Analytics Using SAS® Viya®." *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/2018/1961-2018.pdf.

Yoon, S. H., et al. 2016. "Observer variability in RECIST-based tumour burden measurements: a meta-analysis." *Eur J Cancer*, 53:5-15.

ACKNOWLEDGMENTS

We thank Dr. Geert Kazemier at AUMC for providing us with the image data, and Dr. Nina Wesdorp at AUMC for annotating the images.

RECOMMENDED READING

- *SAS® Visual Data Mining and Machine Learning 8.3: Programming Guide*

CONTACT INFORMATION

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

Fijoy Vadakkumpadan
SAS Institute, Inc.
+1 919 531 1943
fijoy.vadakkumpadan@sas.com

Joost Huiskens
SAS Institute, Inc.
+31 35 6996 831
joost.huiskens@sas.com

Paper SAS4491-2020

Medical Image Analyses in SAS® Viya® with Applications in Automatic Tissue Morphometry in the Clinic

Courtney Ambrozic, Joost Huiskens, and Fijoy Vadakkumpadan, SAS Institute Inc.

ABSTRACT

Imaging and image analytics are indispensable tools in clinical medicine today. Among the various metrics that doctors routinely derive from images, measures of the morphology of tissue structures, including their shape and size, are of key significance. Quantifying tissue morphology and linking those quantities to other clinical data enable clinicians to diagnose diseases and plan treatment strategies. Image segmentation, which classifies image pixels into regions of interest, is an important step in such tissue morphology quantification. However, common segmentation methods involve a process that is either fully or partially manual. Accordingly, these methods can be extremely arduous when you process very large amounts of data. This paper illustrates how to build end-to-end pipelines for automatically deriving clinically significant tissue morphology metrics from raw medical images by using powerful tools that were introduced in SAS® Viya® 3.5. Specifically, it shows how you can load medical images and metadata, preprocess the loaded data, build convolutional neural network models for automatic segmentation, and postprocess the results to compute clinically significant 2-D and 3-D morphological metrics. The examples include colorectal liver metastases morphometry in collaboration with the Amsterdam University Medical Center, and normal spinal cord morphometry with data available from the Cancer Imaging Archive, both based on 3-D CT scans.

INTRODUCTION

Imaging and artificial intelligence have shown many practical applications in the clinic in recent years. SAS® Viya® provides users the building blocks to load, process, and visualize biomedical images. Using these building blocks, users can construct end-to-end pipelines to derive important image-based biomarkers that can be used in the clinic. Segmentation, the process of partitioning an image into sets of pixels, is particularly useful in the derivation of such biomarkers, specifically biomarkers that involve tissue morphometry, i.e., quantification of the size and shape of various tissue structures.

In this paper, we demonstrate how to build fully automatic tissue morphometry pipelines in SAS® Viya® that involves pre-processing, model training, segmentation, and quantification. The motivation for such pipelines is to increase accuracy, decrease subjectivity, and decrease labor of medical professionals. We provide two examples for our demonstration, using different data sets and targeting different clinical biomarkers. Both examples utilize computed topography (CT) scan images in the form of Digital Imaging and Communication in Medicine (DICOM) files. The networks are trained on structural annotations given in the form of DICOM-RT files. The first example uses patient data from Amsterdam University Medical Center and tracks colorectal liver metastasis throughout the duration of a patient's treatment. The second pipeline is built for the segmentation of the spinal cord using data from the Lung CT Segmentation Challenge from 2017 that is publicly available. Using quantification tools in SAS® Viya®, users can derive important biomarkers from the model-predicted contours that have significant impact on clinical insights. The action sets needed to execute these pipelines are the image, biomedimage, fedsql, and

deeplearn action sets. The examples are written in Python, but a user could easily recreate the code in CASL, R, or Lua.

CRLM MORPHOMETRY: AN EXAMPLE USE CASE

MOTIVATION

The first demonstration pipeline that we will highlight will provide a morphometric tissue analysis for patients with colorectal liver metastasis (CRLM). Colorectal cancer is a disease that starts in the colon and often spreads to the liver. The best treatment for CRLM is surgical removal. Currently, the clinical standard for determining a patient's candidacy for surgery lies in the RECIST criterion, which tracks the diameter of two target lesions across successive chemotherapy treatments. To help improve treatment strategies for patients with CRLM through advanced analytics and large amounts of data, SAS joined forces with Amsterdam University Medical Center (AUMC). Through this collaboration, we have access to extensive amounts of data for patients with CRLM.

DATA ACQUISITION AND PREPROCESSING

The data for this example consists of 57 patients and includes medical contours in the form of DICOM-RT files that show medical annotations for the liver and lesions of each image. Data for each patient contains multiple images before chemotherapy, after chemotherapy, and, in some cases, after a continued therapy treatment. Previously, this data was used to show how precisely annotated lesion segmentations can overcome the limitations of the RECIST criteria by quantifying important biomedical metrics such as the volume and contrast of lesions through successive treatments (Vadakkumpadan, Huiskens, 2019). This method, however, requires radiologists to perform an extensive amount of manual contouring for each new patient, which can be both time-consuming and labor intensive. By utilizing deep learning to detect the precise locations of livers and lesions, we overcome the need for these manual tasks on new data. In this use-case example, we propose building an end-to-end segmentation pipeline to identify precise locations of lesions and track the metrics of these lesions over the duration of the patient's treatment process.

Liver lesion segmentation is particularly difficult due to low contrast between lesions and other features as well as lesion shape variability. In order to combat the low contrast within CT scans, several steps are taken in image preprocessing. First, the CT scans are windowed based on the unit of pixel values in CT images, the Hounsfield unit (HU). For the CRLM data, we HU-window the images to the range [-100, 400] to both remove any image features that are not of interest and to highlight important structures. This HU-windowing step is done using the `clamp` step within processBioMedImages. The images are then exported into 2-D slices within the same action call:

```
s.biomedimage.processbiomedimages(
    images=dict(table='ct_scans'),
    steps=[dict(stepparameters=
                dict(steptype='CLAMP',
                     clampParameters=dict(clampType='BASIC',
                                          low=-100, high=400))),
                dict(stepparameters=dict(steptype='export'))],
     copyvars=['_label_','_id_'],
     casout = dict(name='ct_scans_export', replace=True))
```

Next, `hist_equalization` is performed within processImages to enhance the contrast within images. This step distributes the intensity values equally across the image's histogram and therefore allows lesion structures within the image to be highlighted. Figure 1 outlines these preprocessing steps by displaying an original CT scan slice, the same slice

after it is HU-windowed to a range of [-100, 400] and then after its histogram equalization step.

Figure 1. CT Scan Slice, After HU-Windowing, and After Histogram Equalization

The data is then divided into validation, test, and training sets. The training set, which is the data the model will be learning from, consists of 41 patients totaling 18918 slices. The validation set consists of 4 patients (1335 slices) and helps tune hyperparameters of the model. The test set includes 12 patients (6894 slices) and provides unseen images to the model for final evaluation.

SEGMENTATION METHODOLOGY

The convolutional neural network model that will be used for this segmentation task is the U-Net model (Ronneberger et al. 2015), available in SAS® Viya® 3.5. This model is a fully convolutional network that is specifically designed for medical image segmentation and consists of a series of convolutional, concatenation, and max pooling layers. Annotated biomedical images are particularly difficult to obtain due to patient confidentiality and the high level of labor and expertise that is required to create them. The U-Net excels at biomedical image segmentation because it can produce accurate results on smaller annotated image sets. This U-Net is built using layer-by-layer calls within DLPy to create a model with a total of 34512258 parameters. The model uses an Adam Solver with a learning rate of 0.0001.

Two separate U-Net models are trained for liver and lesion segmentation using 2-D slices that have a resolution of 512x512. This two-step model scoring process is done to constrain the solution space for the challenging lesion segmentation task. The first U-Net is trained from scratch for liver segmentation and is shown in Figure 2.

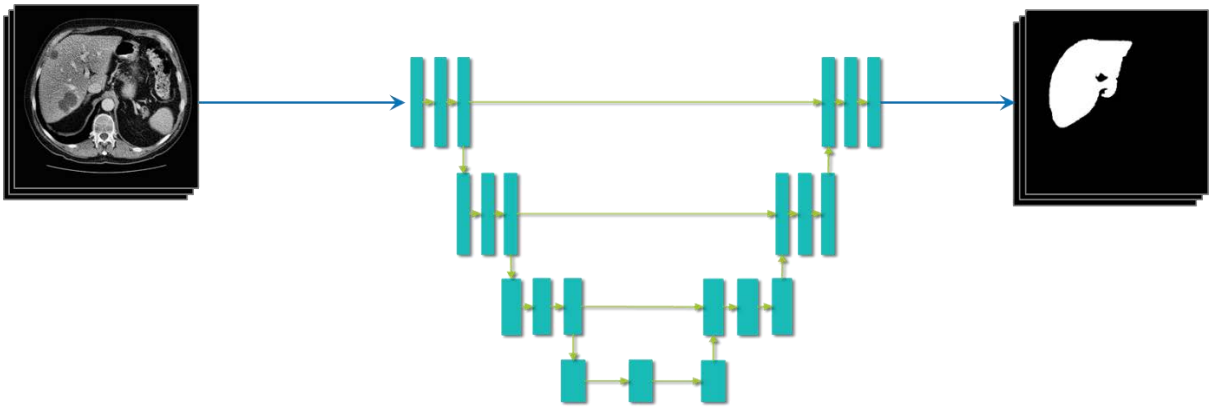


Figure 2. Liver Segmentation Model

The results are cascaded so that the predicted liver segmentations are used as an input test set to the lesion segmentation model. This is accomplished through the `mask_specific` type in the `binary_operation` step of processBioMedImages. Here, the pixel values within the liver region-of-interest (ROI) will be cascaded to the output image and any pixels outside the ROI will have a uniform value. The new `binary_operation` step can read two images from the same input CASTable. The following code demonstrates this process, where the column containing the original CT scan images is named _image_ and the column containing the liver segmentation is labeled "seg":

```
s.biomedimage.processbiomedimages(
    images=dict(table=data_to_be_masked),
    steps=[dict(stepparameters=dict(
                steptype='binary_operation',
                binaryoperation=dict(binaryoperationtype='mask_specific',
                                     image='seg',
                                     outputBackground=-1000,
                                     inputBackground=0)))],
    casout=gray_mask_liver,
    copyvars=['_label_', '_id_'],
)
```

Figure 3 displays the lesion segmentation schematic, where the output of the `mask_specific` type is used as the input to the model.

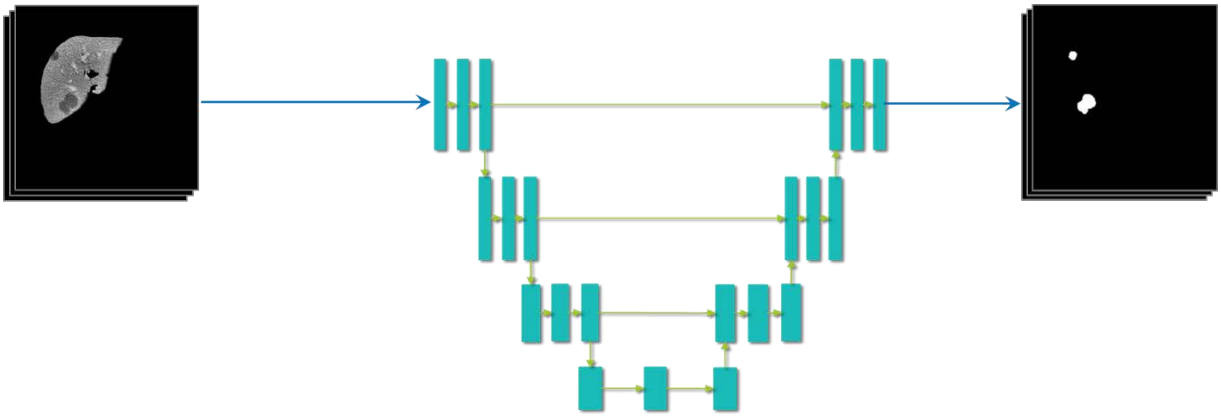


Figure 3. Lesion Segmentation Model

With the solution space constrained to the liver ROI, the model can more accurately predict the lesion regions.

RESULTS

Figure 4 presents the liver segmentation results where blue depicts the liver ROI predicted by the model and the red region is the ground truth liver ROI annotated by the radiologist. These segmentation-overlaid images are created using the annotateImages action.

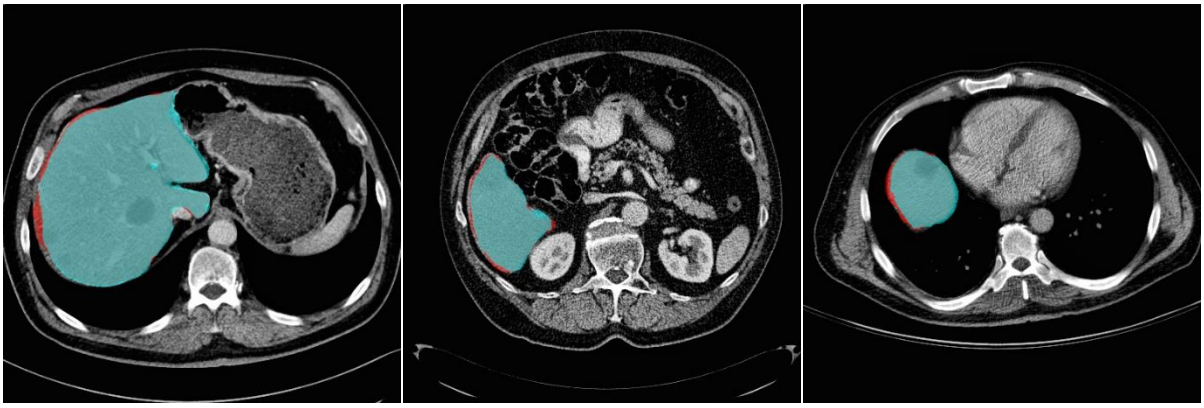


Figure 4. Liver Segmentation Results

The liver segmentation results are then cascaded to be scored for lesion segmentation using the `binary_operation` step outlined previously. In Figure 5, the lesion ROI predicted by the model is shown in blue and the ground truth lesion ROI is displayed in red.

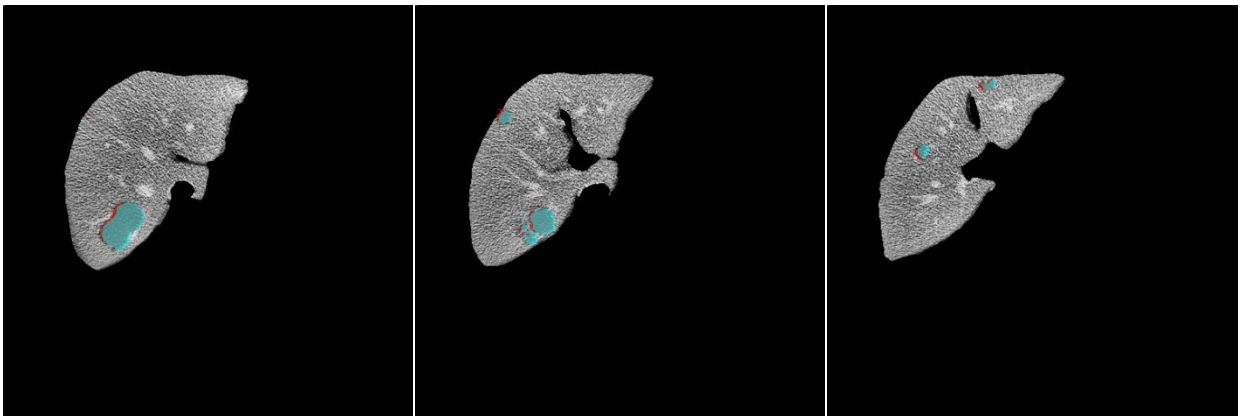


Figure 5. Lesion Segmentation Results

The segmentation predictions are imported back into 3-D using processBioMedImages. These 3-D segmentation images are then built and plotted on the original CT scan using Mayavi software (Ramachandran, 2011). The buildSurface action is utilized for this task to build the surfaces of both the liver and lesion segmentation results. This 3-D visualization for the model prediction is shown in Figure 6. Here, the red surface is the liver ROI and the green surface is the lesion ROI.

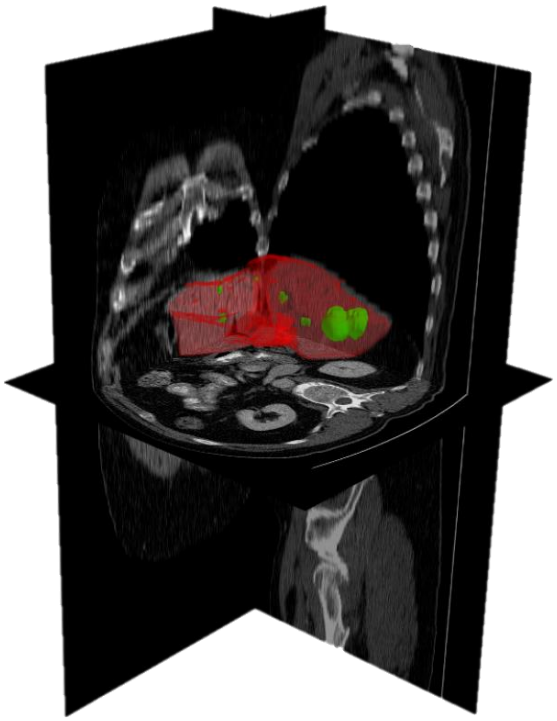


Figure 6. Model-Predicted Liver Lesion 3-D Visualization

The segmentation methods are evaluated against the ground truth for similarity. The main evaluation criteria used for these segmentation images is the DICE coefficient. The DICE score is a much more stringent criterion than misclassification rate and is therefore ideal for quantifying the performance of the model. DICE is defined as two times the area of overlap divided by the total number of pixels:

$$DICE(A,B) = \frac{2|A \cap B|}{|A| + |B|}$$

where a perfect segmentation yields a DICE score of 1. The liver lesion segmentation results are evaluated using the DICE coefficient and the results are shown in Table 1.

	Liver	Lesion
Test set	93.155%	77.703%
Validation set	94.167%	69.975%

Table 1. DICE Coefficients for Liver and Lesion Segmentation

For scoring, we use the DICE global score, which averages the total test set. It is important to note that the global DICE score for lesion segmentation is very dependent on the size of the lesions in the evaluation set. The most competitive models in the Liver Tumor Segmentation Benchmark (LiTS) achieved a DICE score of 96.7% for liver segmentation and 79.40% for lesion segmentation (Bilic et al. 2019). From the predicted contours, metrics can be derived that describe the tissue morphometry. The action quantifyBioMedImages can be used to quantify the lesion segmentation results to analyze volumes and pixel values, both of which are ignored by the RECIST criteria. By specifying the quantify type as `'content'`, the total volume of the lesions is calculated:

```
s.biomedimage.quantifyBioMedImages(
   images=dict(table='bdata_lesion'),
   region='image',
  quantities=[dict(quantityparameters=
```

```
                    dict(quantitytype='CONTENT',usespacing=True)),
                    dict(quantityparameters=dict(quantitytype='MEAN'))],
    inputbackground=-1000,
    labelParameters=dict(labelType='basic', connectivity='vertex'),
    copyvars=['_label_', '_id_'],
    casout=vol)
```

The volume calculations are plotted by patient, ordered by their round of chemotherapy treatment and shown in Figure 7. The first scan before treatment is depicted in blue, the first follow-up scan is depicted in orange, and the second follow-up scan (if it exists) is shown in green.

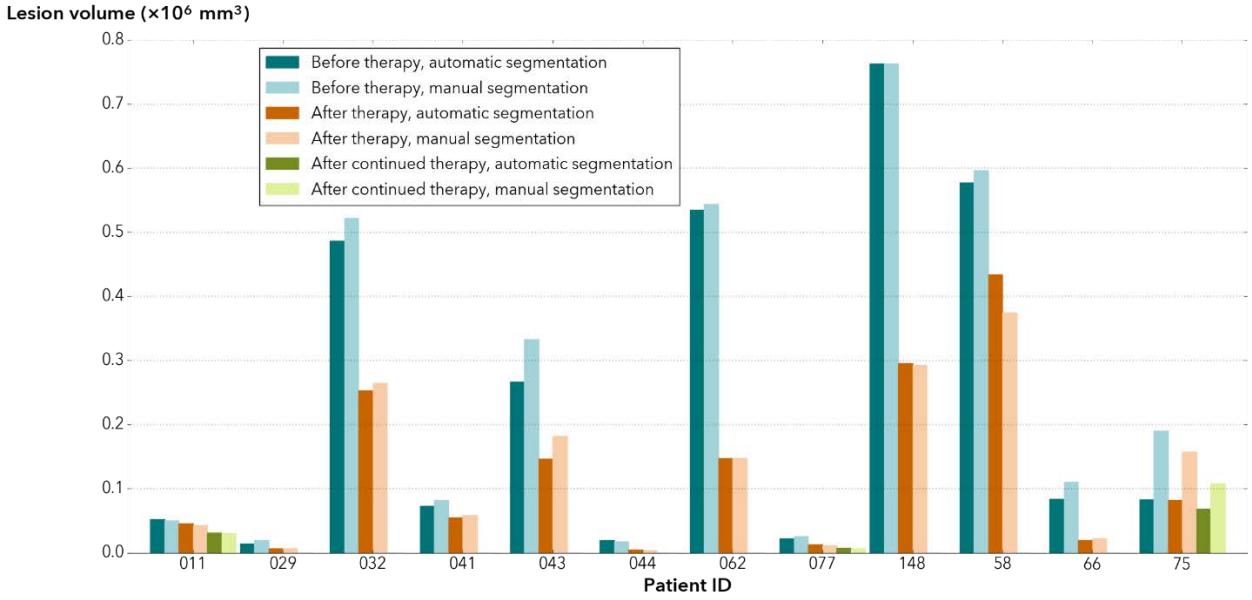


Figure 7. Lesion Volume from Model-Predicted Segmentations

The volumes predicted by the model are plotted against the volumes annotated by the radiologist and used as ground truth. The model-predicted results are displayed in a darker color and the ground truth results are in a lighter color. The model-predicted lesion volumes follow the same trends throughout treatments as the ground truth volumes and therefore verify the effectiveness of the segmentation model. The automatic segmentation volume averaged a 7.713% decrease in comparison to ground truth.

The segmentation predictions are then computed for contrast between the liver and lesions. Contrast is calculated by comparing the mean pixel values within the lesion segmentation to those within the liver segmentation. Figure 8 displays the liver-lesion contrast for each image based on the automatic segmentation from the model. The lesion pixel values are another metric ignored by RECIST that is captured through the automatic segmentation pipeline.

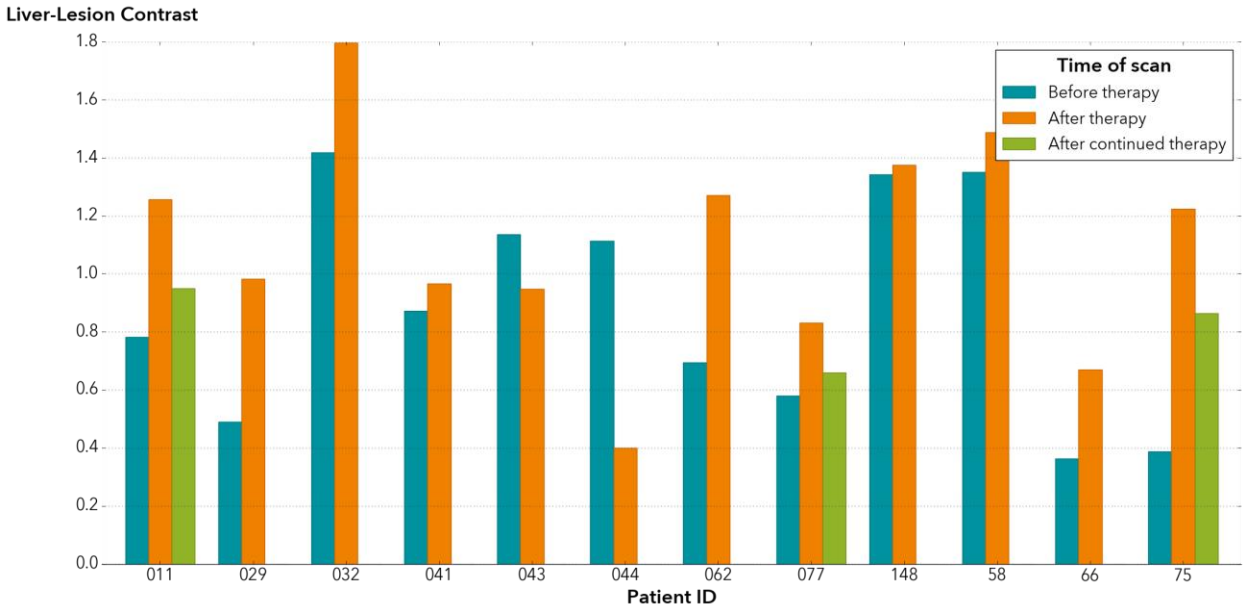


Figure 8. Liver-Lesion Contrast

LCTSC MORPHOMETRY: AN EXAMPLE USE CASE

MOTIVATION

Patients with multiple sclerosis (MS) often develop lesions on their spinal cord and suffer from loss of volume within their spinal cord. This loss of volume can be an important indicator of long-term disability from MS (Andelova et al. 2019). Segmentation of the spinal cord and lesions can provide measures of damage, which are key criteria for the diagnosis and monitoring of patients with MS (Gros, 2018). Automating this contouring process eliminates variability between radiologists. In this example, we apply a similar methodology to construct an automatic pipeline for spinal cord segmentation. The Jupyter notebook for this use-case is available for download here. This notebook gives users the opportunity to run the automatic spinal cord segmentation pipeline using a data set that is publicly available for download and use.

DATA ACQUISITION AND PREPROCESSING

The data used in this experiment is from Lung CT Segmentation Challenge (LCTSC) data set available at the Cancer Imaging Archive (Yang et al. 2017) and consists of 60 patients. The organs-at-risk (OARs) that are included in this challenge consist of annotations for the esophagus, heart, left and right lungs, and spinal cord. The DICOM-RT files contain contours for each of these organs and are displayed in different colors within the image.

The first step of pre-processing the images for spinal cord segmentation is to use the `roi2mask` step to filter out the organs that are not the spinal cord within the DICOM-RT files. This is executed by specifying the color of the spinal cord within the new parameter `roidisplaycolor`:

```
s.biomedimage.processbiomedimages(images=dict(table=imrt),
    steps=[dict(stepparameters=dict(steptype='roi2mask',
          roi2maskparameters=dict(roi2masktype='dicomrt_specific',
                                  roicontoursequence='_ROIContourSequence_,
                                  correctionsensitivity=.25,
                                  pixelintensity=255,
                                  outputbackground=0,
```

```
                                roidisplaycolor=colors[0]))),
        dict(stepParameters=dict(stepType='rescale',
                rescaleparameters=dict(rescaleType="channeltype_8u")))],
    casout=dict(name= 'masks_all', replace=True),
    copyvars=['_id_', 'color', 'RTID', '_label_'])
```

Figure 9 exhibits the contours for a patient before and after this filtering process. With the remaining contours containing only those for the spinal cord, segmentation masks are created for model training.

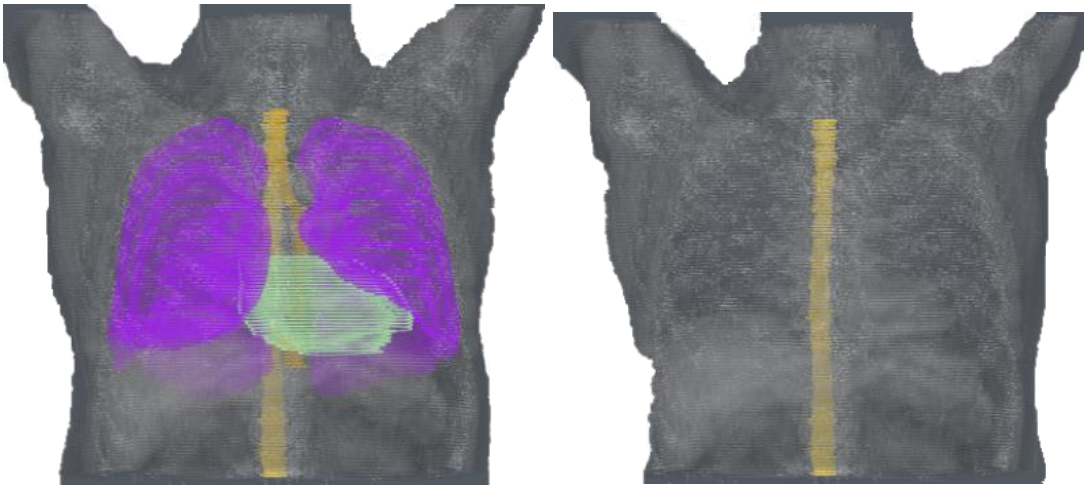


Figure 9. DICOM-RT Contours Before and After Filtering by Color

The image data is divided into training, test, and validation sets. The data consists of 60 patients total where 36 images (2106 slices) are used for training, 7 images (403 slices) are used for validation and 17 images (634 slices) are in the test set. Comparable to the CRLM data, the 3-D images are exported into 2-D slices for training that have a resolution of 512x512.

SEGMENTATION METHODOLOGY

A U-Net is trained for spinal cord segmentation given the segmentation masks created using `roi2mask`. This network is not being cascaded, as the U-Net is trained directly for spinal cord segmentation. The model is built using the U-Net DLPy API:

```
model = UNet(s,
             n_classes=2,
             width=512,
             height=512,
             n_channels=1,
             bn_after_convolutions=False)
```

This default model almost exactly resembles the models trained for liver and lesion segmentation with the main exception being in the kernel size of the last convolutional layer and the lack of activation function in the segmentation layer. This model has 34513282 parameters and uses an Adam solver with a learning rate of 0.0001. This model is trained from scratch for spinal cord segmentation over 50 epochs.

RESULTS

The test set is scored for spinal cord segmentation and these results are imported back into 3-D. The segmentation results for one patient are built and plotted on the original CT scan using Mayavi software. This display is shown in Figure 10, where the red portion of the

image indicates the region that is predicted by the model and the green region indicates the ground truth spinal cord ROI.

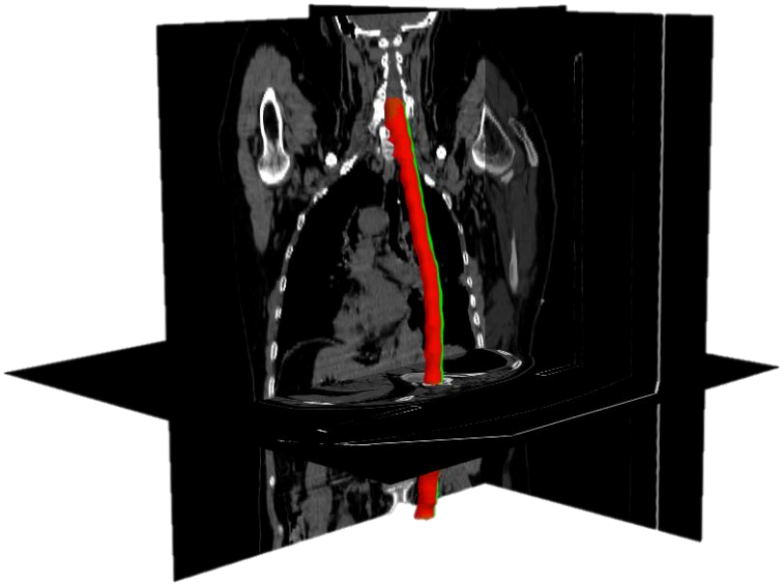


Figure 10. Spinal Cord Segmentation Results

This test set is evaluated using the DICE score coefficient against the ground truth spinal cord contours. The DICE coefficient averaged 71.349% on the test set and 69.547% on the validation set. The results are then quantified using quantifyBioMedImages and the volumes are plotted against the original spinal cord volumes annotated by the radiologist. These predicted volumes along with their ground truth comparison are displayed in Figure 11.

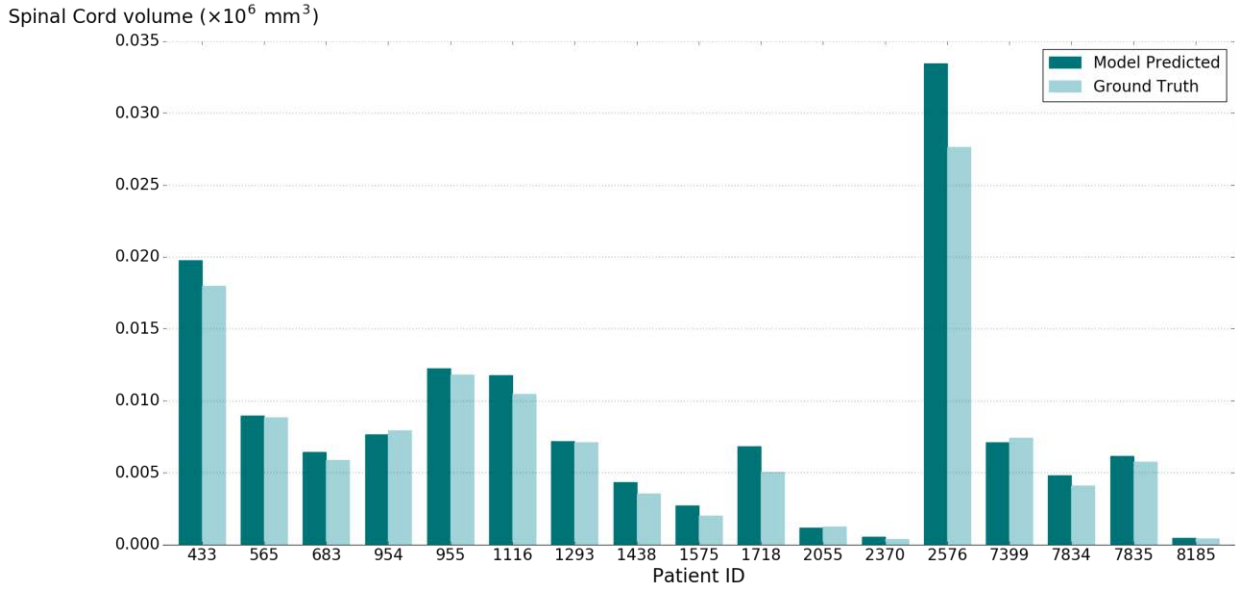


Figure 11. Spinal Cord Volume from Model-Predicted Segmentation Results

The predicted model segmentation averaged a 11.1586% decrease in volume when compared to the ground truth segmentation.

DISCUSSION

The ideas presented in this paper showcase the automation of segmentation and morphometric analysis for biomedical images in SAS® Viya®. This medical contouring application can aid radiologists in the clinic for detection and diagnosis of disease, as well as help track the health of the patient throughout the course of a disease. This automatic process reduces the burden and fatigue placed on medical professionals to perform rigorous manual contouring. By reducing the number of arduous tasks that medical professionals must face, we therefore reduce the risk of human error in the clinic. In addition, visualization and quantification of model predictions can help with personalized medicine for patients within the clinic. These derived measures can aid medical professionals with important decisions about a patient's long-term treatment.

The two use case examples show how automatic segmentation can assist clinicians in deriving important biomarkers. In the CRLM case, we overcame limitations of RECIST by capturing the total volume of the lesions rather than relying on their 1-D representation. A visual analysis of the predicted regions shows that in the before therapy case, the model tends to underpredict the total volume of the lesion. This is most likely due to the fact that before chemotherapy, lesions lack definitive boundaries. After chemotherapy, lesions tend to shrink, darken, and have more defined boundaries. As a result, the model has difficulty capturing the full extension of these pre-chemotherapy lesions within its prediction.

The second example using LCTSC data demonstrates a pipeline that users can run with data that is publicly available. Once again, we show how the automatic segmentation method allows users to derive important clinical biomarkers with little manual effort. Loss of spinal cord volume can be a strong predictor of long-term disability in patients with MS. Therefore, tracking the spinal cord volume loss over a period of time can lead to significant insights about a patient's long-term health within the clinic. It should be noted that the spinal cord images are clipped to a range of slices for each patient. The dramatic difference in volumes between patients corresponds to the number of slices for each patient. Therefore, the patient volumes should not be compared against each other. In contrast to the CRLM case, this model tends to over-predict the spinal cord regions. The predicted spinal cord region from the model often extends outside the ground truth region and, in some cases, misclassifies small regions outside of the target area.

CONCLUSION

New biomedical image analysis features in SAS® Viya® 3.5 provide tools for data preparation, image segmentation, visualization, and quantification. If you wish to download and run the pipeline for spinal cord segmentation, please follow the link provided here. Through the demonstrated segmentation and morphometric analysis pipeline, users can create an efficient detection method for important structures within CT scans. These methods improve efficiency and accuracy of biomedical structure identification, reducing burden and fatigue of medical professionals. Once the models are trained on data, the automatic segmentation can replace or assist manual segmentation tasks by medical professionals in the clinic. The segmentations are evaluated by the DICE coefficient and are shown to be competitive with state-of-the-art methods. Volumetric and pixel analysis are used to track disease progression over time and provide substantial assistance to clinical assessments. In future work, we'll be focusing on the quantification of other important biomarkers, mainly from the IBSI standard (Zwanenburg, 2016), which will involve the expansion of quantifyBioMedImages.

REFERENCES

Andelova, M., et al. 2019. "Additive Effect of Spinal Cord Volume, Diffuse and Focal Cord Pathology on Disability in Multiple Sclerosis." Front. Neurol.

Bilic, P., et al. 2019. ‘‘The Liver Tumor Segmentation Benchmark (LiTS),” arXiv:1901.04056. Available https://arxiv.org/abs/1901.04056

Christ, P. F., et al. 2016. “Automatic Liver and Lesion Segmentation in CT Using Cascaded Fully Convolutional Neural Networks and 3D Conditional Random Fields.” In Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016, ed. Ourselin, S., et al, 415-423. Cham: Springer.

Gros, C., et al., 2018. “Automatic segmentation of the spinal cord and intramedullary multiple sclerosis lesions with convolutional neural networks.” Neuroimage, 184:901–915.

Eisenhauer E.A., et al. 2009. “New response evaluation criteria in solid tumours: revised RECIST guideline (version 1.1).” European J Cancer, 45(2):228-247.

Huiskens J., et al. 2015. Treatment strategies in colorectal cancer patients with initially unresectable liver-only metastases, a study protocol of the randomised phase 3 CAIRO5 study of the Dutch Colorectal Cancer Group (DCCG).” BMC Cancer, 15:365.

Ramachandran, P., and G. Varoquaux. 2011. “Mayavi: 3D Visualization of Scientific Data.” IEEE, 13(2):40-51.

Ronneberger, O., et al. 2015. “U-Net: Convolutional Networks for Biomedical Image Segmentation.” MICCAI, 9351: 234–241

Vadakkumpadan, F., and J. Huiskens. 2019. “Medical Image Analytics in SAS® Viya® with Applications in the Treatment of Colorectal Cancer Spread to the Liver” 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/2019/3341-2019.pdf.

Vadakkumpadan, F., and S. Sethi. 2018. “Biomedical Image Analytics Using SAS® Viya®.” 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-forumproceedings/2018/1961-2018.pdf.

Yang, Jinzhong; Sharp, Greg; Veeraraghavan, Harini ; van Elmpt, Wouter ; Dekker, Andre; Lustberg, Tim; Gooding, Mark. (2017). Data from Lung CT Segmentation Challenge. The Cancer Imaging Archive. http://doi.org/10.7937/K9/TCIA.2017.3r3fvz08

Yoon, S. H., et al. 2016. “Observer variability in RECIST-based tumour burden measurements: a meta-analysis.”Eur J Cancer, 53:5-15.

Zwanenburg, A. 2016. “Image biomarker standardisation initiative”, 123. EP-1677.

ACKNOWLEDGMENTS

We thank Dr. Geert Kazemier at AUMC for providing us with the image data, and Dr. Nina Wesdorp and Sam at AUMC for annotating the images.

RECOMMENDED READING

- *SAS® Visual Data Mining and Machine Learning 8.5: Programming Guide*

CONTACT INFORMATION

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

Courtney Ambrozic
SAS Institute, Inc.
+1 919 531 2125
Courtney.ambrozic@sas.com

Fijoy Vadakkumpadan
SAS Institute, Inc.
+1 919 531 1943
fijoy.vadakkumpadan@sas.com

Joost Huiskens
SAS Institute, Inc.
+31 35 6996 831
joost.huiskens@sas.com

Paper 4694-2020

Deploying Computer Vision by Combining Deep Learning Action Sets with Open Source Technology

Jonny McElhinney and Duncan Bain, ScottishPower Energy Retail Ltd;
Haidar Altaie, SAS Institute Inc., UK

ABSTRACT

Whilst early computer vision dates back as far as 1927, it has gained momentum in the last years due to the developments in the fields of deep learning and artificial intelligence. With the desire for applying computer vision in ever more general and flexible contexts, the challenge arises how we can push image processing models to production in a robust way.

This paper focuses on doing Automatic Meter Reading (AMR) using customer-submitted photos of their meters.

The challenges in this context are two-fold. First, we need to localise the box containing the digits, and then we must classify each digit with the correct label, 0-9. Many approaches are available but both of these challenges can be addressed by combining the Deep Learning action set in SAS® Visual Data Mining and Machine Learning (VDMML) with DLPy and Keras.

However, when applying these models in a real-world business context, an additional challenge arises, which is how to deploy and keep track of these models in a consistent way. This paper shows how SAS® and open source tools can be used together to provide a consistent approach both to creating as well as managing and deploying models.

INTRODUCTION

Computer vision has the potential to be one of the key enabling technologies for the 21st century. The abundance of image data that is continuously being captured by billions of high resolution cameras has provided a diverse range of applications and research areas for computer vision. These include facial expression recognition, medical image analysis, self-driving cars, upscaling historical black and white footage, Optical Character Recognition (OCR), and many more.

The prevalence and technological advancements of smart phones in recent years provides a portable platform for deploying computer vision applications. Low power consumption and fast processing time are key requirements for any task carried out on handheld embedded systems. For this reason, despite their high accuracy, deep learning models are not always the most desirable solution due to their size, computational power requirement, and slow inference times when compared with traditional image processing techniques. However, for a task as challenging as natural scene text classification using customer-submitted photos, detecting and classifying regions of text correctly becomes incredibly difficult.

The quality of customer-submitted photographs varies massively from person to person, camera to camera, and meter to meter. In the UK, gas meters are typically housed outside of the premises and are susceptible to superficial damage over time and photographs can be taken in many different natural light conditions. Electrical meters are often hidden inside in dark cupboards and therefore the camera flash can be required. Many meter cases are made of white plastic and therefore reflect a lot of light if the flash is used. There is also a split between analog and digital counter displays. All these variables make deep learning a necessity for a robust system that can accommodate all sorts of input images.

The deep learning approach adopted for this task is to create a pipeline of three Convolutional Neural Networks (CNNs) to process a customer-submitted image and return a meter reading to be entered into the customer database:

1. Detect and extract counter region from customer photo
2. Detect all digits within counter region
3. Classify all digits (from 0-9) and construct output reading

This paper proposes an end-to-end AMR system which can be deployed to mobile devices using the ONNX (Open Neural Network Exchange) framework. ONNX allows models to be trained with one framework, converted to ONNX, and deployed with another. ONNX presents a realistic solution to the task of deploying computer vision models in production. It also enables the combination of models trained using Keras and models trained within the SAS® Viya® environment.

PROBLEM OVERVIEW AND BUSINESS VALUE

In the UK, smart meter rollout is taking longer than expected and is unlikely to reach one hundred percent coverage in the near term. All the while, customers without smart meters need to submit readings to guarantee accurate billing. This can be challenging for customers with vision or mobility difficulties and for many customers can lead to disputes in cases with long term consecutive estimates or incorrectly submitted prior readings. This is particularly a problem in the UK as electricity meters are usually located inside the premises and so can only be read by field staff if the customer is home.

SMART METER PENETRATION

Following the Energy Act (2008), the UK government mandated the replacement of all 47 million residential gas and electricity meters by smart meters [1]. Data from the Department for Business, Energy & Industrial Strategy (BEIS) shows that only around 30% of domestic meters in operation are smart as of Q3 2019 [2]. The smart meter penetration rate, shown in Figure 1, can be broken down to 31.3% for electricity meters and 28.3% for gas meters.

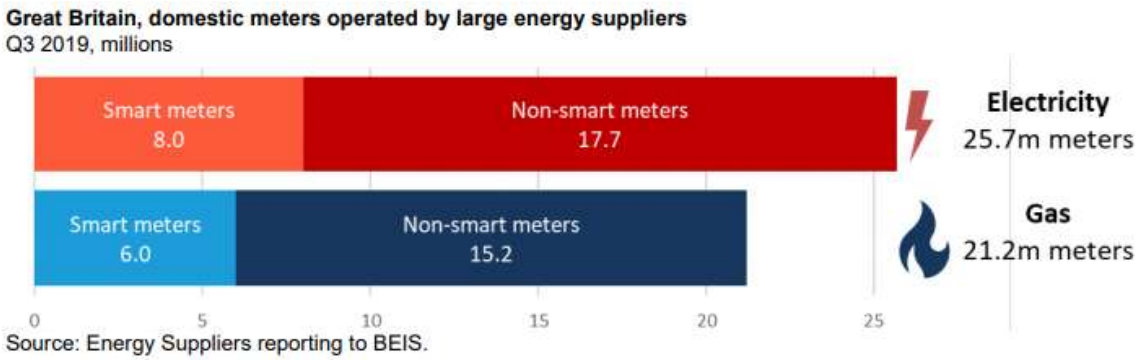


Figure 1. UK Domestic Meters Operated by Large Energy Suppliers

The number of smart meters in operation will continue to steadily rise however a large percentage of the UK population will remain on traditional meters for the foreseeable future. Some of the contributing factors for this are; unsuitable DCC network coverage for rural areas, physical impracticalities of installing smart meters inside some older housing stock, and the customers' right to refuse a smart meter being fitted.

CUSTOMER IMPACT

Natural language processing of chat contacts shows that:

- in weekly bigram word-clouds, "meter read" is consistently in top 5 topics discussed.
- hexagrams show meter reading related web-chat accounts for almost 30% of customer contact through that channel and that many of those chats are about the customer being unable to submit a reading for one reason or another.

If customers fail to supply regular self reads:

- estimates will begin to deviate markedly from historic consumption profiles if circumstances change in the premises (i.e. occupancy change, lifestyle change).
- incorrect estimates lead to inaccurate billing which drives both customer contact and complaints.

Photographic reads can make the process simpler for all customers who don't yet have a smart meter. Electronically submitting reads that can be checked against the last verified reading to ensure consistency and identify metering or reading errors.

Additionally, this process would make disputed read resolution much simpler as the evidence of the correct read as well as other information about the meter such as the serial number and device type would be captured in the image.

Together these use cases have the potential to remove tens of thousands of contacts per week across all inbound channels, reducing queue times and allowing customer service agents to concentrate on value adding tasks.

There are other benefits that can be realized in parallel with the reduction in calls in terms of app penetration, customer satisfaction, sales opportunity and the halo effect generated by bringing a leading-edge service to market.

Given the default size of high resolution smartphone images and latency over poor connections, all the processing needs to be done on the device at the edge, with an opt in to submit the photo for quality assurance and dispute resolution. This gives the customer the power to control their data and how it is utilized.

SETTING UP THE PROJECT

SOFTWARE REQUIREMENTS

In order to support the functionalities of the computer vision action sets through SAS Deep Learning features required for this project, we used SAS® Visual Data Mining and Machine Learning 8.4 in SAS® Viya® 3.4. This enabled us to load data, transform data, compute statistics, perform analytics and create output to save image data and associated metadata at scale. Additionally, SAS Visual Data Mining and Machine Learning takes advantage of SAS Cloud Analytic Services (CAS) to perform what are referred to as CAS actions. Each action is configured by specifying a set of input parameters. Running a CAS action processes the action's parameters and data, which creates an action result. Throughout this project we leveraged the "deepLearn" CAS action set [3]. This action set consists of several actions that support the end-to-end preprocessing, developing and deploying deep neural network models.

One of the key components in this process was the flexibility to use both SAS built models, and previously created models on Keras through transfer learning. Thus, we leveraged DLPy, SAS Deep Learning with Python [4]. DLPy is an open-source package, available in the Python library that data scientists can download to apply SAS Deep Learning algorithms and features available in SAS Viya to image data. DLPy is a toolset in a Python/Keras-style shell, accessed via Jupyter Notebook, for the SAS scripting language and the SAS deep learning actions from SAS Visual Data Mining and Machine Learning, with the look and feel of Python following the Keras APIs, with a touch of PyTorch flavour. DLPy also includes high-level APIs for predefined network architectures which have been implemented within this project.

Another essential package to make this possible is the SWAT (SAS Scripting Wrapper for Analytics Transfer) package [5]. The SAS SWAT package is a Python interface to the SAS Cloud Analytic Services (CAS) engine. Using SWAT, you can execute CAS actions, then pull down the summarized data to further process on the client side in Python (also available for R and Lua languages). The SWAT packages feeds into DLPy and is a required to run it.

In summary, SAS Viya® supports computer vision through 3 key interfaces:

1) SAS® Cloud Analytic Services (CAS) – The engine behind SAS® Viya® which is used to perform the CAS actions for analytical and deep learning transformations.
2) SAS Scripting Wrapper for Analytics Transfer (SWAT) – The Wrapper that allows us to program in a Python interface to call the CAS engine.
3) SAS Deep Learning with Python (DLPy) – The Python interface used to apply SAS Deep Learning algorithms in a Keras-type format. The full library is available on GitHub, with examples templates and videos.

ENVIRONMENT

As Deep learning models tend to be computationally intensive. The requirement for a GPU (Graphical Processing Unit) is often essential for training and scoring the models, especially when the models are complex. Unfortunately, GPUs could easily become expensive to run for a longer period, and the requirements to switch from GPUs to CPUs (Central Processing Unit) is vital. Illustrated in Figure 2, we were comfortable to build, test and score the model interchangeably between CPUs and GPUs.

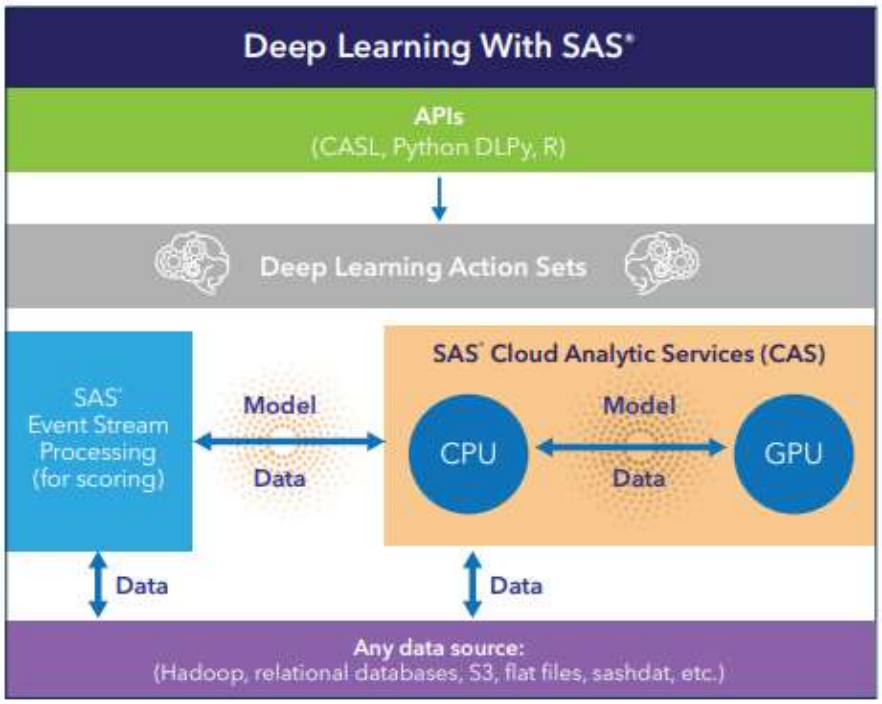


Figure 2. The SAS Deep Learning Platform for storing and modelling on AWS

We were able to deploy the SAS® Viya® stack, alongside the appropriate Python libraries onto an AWS environment, with both CPUs and GPUs. Specifically running it on Amazon Elastic Compute Cloud (Amazon EC2). The advantage was that the instance provided a web service which was secure and resizable. The specific instance type was one of the accelerated computing instances when using the GPUs, this specifically relevant for high performance computing for Machine/Deep Learning. Instance p3.8xlarge, which was chosen for this project, included 4 NVIDIA TESLA V100 GPU's with 244 GB of memory when the model needed heavy training and scoring [6]. Alternatively, CPUs were able to be used when the model was being developed, and GPUs were not required in this elastic, resizable environment. Whilst using CPUs, we were then able to leverage the CAS engine, which run on an in parallel, in-memory server, we were still able to train and score models with a sub-sample of the data.

AUTOMATIC METER READING

The problem of classifying a meter reading from a customer-submitted photo can be considered to be OCR for natural scene text. This area of work differs to widely studied challenges within the scope of OCR due to the complexity and diversity of input images.

Figure 3 shows several examples of UK gas and electricity meters. The variety in meter styles makes a traditional image processing approach very difficult without creating multiple different pipelines to accommodate all styles of meters. Even then, there is a reliance on customers to take high quality photographs in good lighting from similar distances and angles to ensure reliable performance. These criteria are not conducive to a good customer experience and therefore a more flexible solution using deep learning is the most realistic solution.

Figure 3. UK Gas (Top) and Electricity (Bottom) Meters

DOMAIN CHALLENGES

There are many challenges for an AMR system. First is the number of unwanted textual blocks (barcode, serial number, branding, units of measure, telephone number, miscellaneous text) surrounding the Region of Interest (ROI), the counter. The first step in the proposed AMR pipeline is to extract the counter from the input image for further processing. It is vital that the whole reading is included in the extraction process so that an error here does not propagate through the system, for example the leading digit being omitted. A previously adopted approach is to automatically pad all counter detections by 10-20% in width and height to ensure all digits are included.

After isolating the counter within the input image, extracting the reading is not as simple as classifying all digits within the display and submitting. Readings can contain insignificant decimal digits that are not required. For example, the top left meter in Fig 3. displays a reading of 09309.554 but a customer would be expected to enter 09309 when manually submitting their reading. The proposed system must be able to discard the appropriate digit(s) if any remain after the counter detection process. One way of addressing this is to use historical customer readings as a reference of the number of expected significant digits, in most cases 4 or 5.

A challenge unique to AMR within the natural scene text recognition domain is the presence of digits in a transitional state, or 'scrolling' digits, on analog displays. Utility meters increment as energy is consumed so it is possible to capture a photograph of the reading in transition, as shown in Figure 4. A widely adopted protocol is to classify a scrolling digit as the lower of the two digits until the next digit is entirely visible, except for 9 to 0 which should be classified as 9 until the end of the transition.

The lower registers on the right-hand side increment more frequently as each unit of energy is counted. This means that a misclassified digit at the end of the reading will produce an incorrect reading that is much closer to the ground truth value compared to a misclassified digit in one of the higher registers. Fig. 4 shows the challenge of 4/5 of the digits in transition and so a misclassification error on one of the higher registers would lead to a predicted reading that is much more inaccurate. For example, the ground truth reading in Fig. 4 is 12999 but one digit being misclassified can be the difference between a predicted reading of 12990 and 13999.

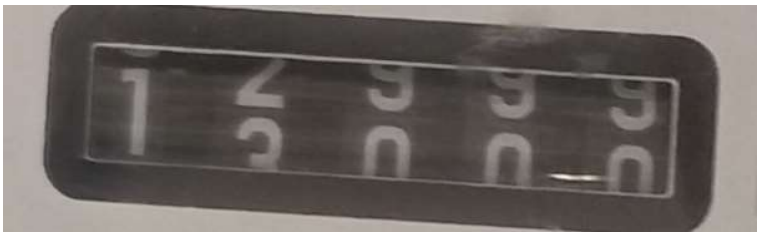


Figure 4. Scrolling Digits

The nature of customer-submitted images also presents a high degree of difficulty. Meters can be over or under exposed by difficult lighting conditions and motion blur can render digits unreadable. Artifacts such as dirt, condensation, or superficial damage can occlude some or all of the digits. A minimum requirement for an image presented to an AMR system is that a human operator would also be able to transcribe the reading. Figure 5 shows images that do not meet this requirement due to occluding artifacts or poor lighting.

There is no standardization across meter designs and as a result many training samples of each different meter type are necessary to guarantee a robust system. Meter displays can appear in different positions with variations in size, font, and number of digits. Analog displays typically contain high contrast white digits on a black background, however digital screens can have a smaller contrast of black digits on a grey background. Digital displays can use either 7 or 16 segment displays for each digit and some displays have backlights of different colors.

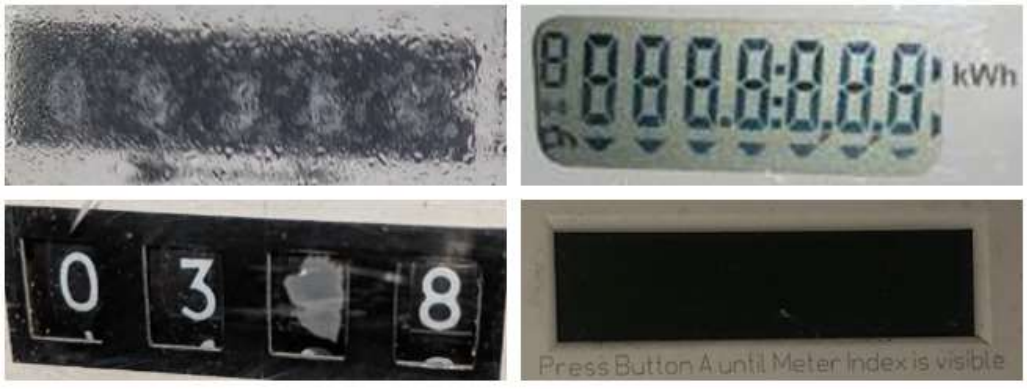


Figure 5. Low Quality Meter Images

ALTERNATE APPROACHES

Several different approaches were researched and tested before selecting the 3-stage CNN pipeline. This included traditional image processing in OpenCV using Canny edge detection and contour analysis for counter detection and digit segmentation followed by template matching for digit classification. Some correct reading predictions were obtained on clean input images, but it was clear that a deep learning approach would be the best solution to accommodate the variety of image quality and meter styles in customer-submitted images.

The first deep learning approach was to use off-the-shelf algorithms for the tasks of natural scene text detection and then classification. This can be done using open source technologies such as the Efficient and Accurate Scene Text (EAST) detector [7] to localize areas regions of text and then classifying the text using Google's open source Tesseract engine. Results highlighted that whilst off-the-shelf algorithms are straight forward to implement, the system output is not much better than the traditional image processing approach and has its own pitfalls. For example, algorithms are trained for generic robust reading tasks such as identifying text in public spaces on signs. The nature of text and the environment is very different to the challenge of AMR and therefore the reading can often be missed in the text detection stage. The inconsistency of the model is shown in Figure 6.

Figure 6. Example EAST Detections

If the reading is found and a classification is made, a lot of post-processing is required to extract it due to the residual text and numbers printed around the counter display. Digits are often misclassified due to the variety of fonts and the system fails when presented with scrolling digits.

Google offers a paid-for Vision API to perform OCR among other tasks on input images. It performs better in both text detection and classification, but it still requires images to be high quality with very little motion blur or occluding artifacts around the reading itself. If the digits are not visible with high contrast the reading will not be detected and it also struggles to classify scrolling digits.

Conclusions from testing off-the-shelf OCR models for AMR aligned with the findings of Dr. Adrian Rosebrock [8]:

> "The best accuracy will come from training custom character classifiers on specific sets of fonts that appear in actual real-world images … There is no such thing as a true 'off-the-shelf' OCR system that will give you perfect results."

METHODOLOGY

The pipeline used for our AMR system combines object detection and image classification CNNs to locate and recognize a customer's meter reading. Object detection represents the task of locating the bounding box co-ordinates and category of objects in a given image. Image classification produces a single highest confidence prediction of the contents of an image from a pre-defined set of possible classes.

Step 1 of the AMR pipeline uses a single class object detector that is trained to detect counter displays from a customer image. The assumption of one single counter per image is used to extract the counter and discard the irrelevant image data from further processing. Horizontal and vertical padding is employed to reduce the potential for an incomplete reading to be taken through the rest of the pipeline.

Step 2 uses another single class object detector operating on the extracted counter image to locate all digits visible on the counter display. This approach was adopted instead of a multi-digit recognition system due to the potential variation in reading lengths. Digits are extracted in isolation for individual classification.

Step 3 uses a 10-class image classification model trained on individual digits 0-9. Digits are classified from left to right and combined to build the output reading which can then be entered into the customer database. Step 3 also contains logic checks to ensure the predicted reading is plausible. This is done using model confidence scores from all 3 steps as well as historical reading data if available.

Figure 7 visualizes each step for processing an image with the AMR pipeline.

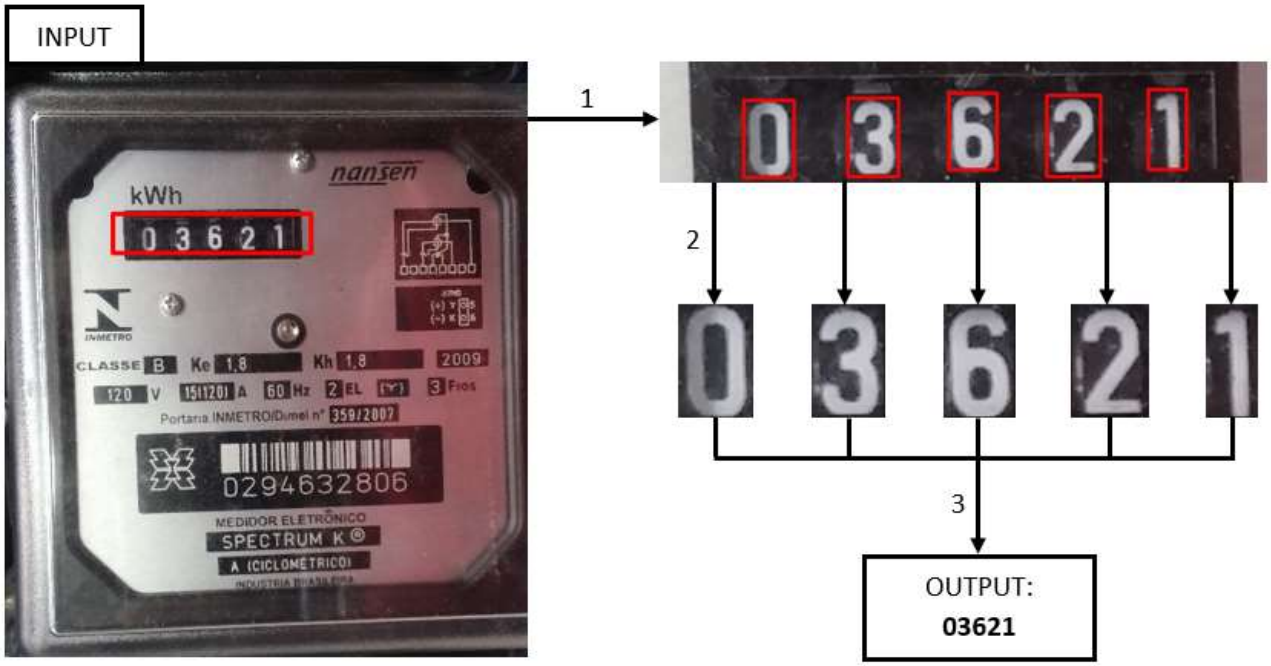


Figure 7. System Flow Diagram

DATASET

In order to train each of the three deep learning models, lots of labelled images of energy meters are required. There is no dataset of ScottishPower customer images so as proof of concept, a publicly available dataset was used. The UFPR-AMR dataset produced by Laroca et al. [9] contains 2000 labelled images of Brazilian electricity meters and is the largest publicly available meter dataset. The dataset is for research purposes only and will not be used in production.

Each image has a corresponding text file containing the reading value and pixel co-ordinates of the counter display as well as each individual digit in the format [x, y, w, h], shown in Figure 8. The training data for each of the three models can be extracted from these labels.


```
camera: LG G3 D855
reading: 08687
position: 770 1575 687 174
        digit 1: 804 1608 69 111
        digit 2: 940 1612 73 108
        digit 3: 1079 1600 69 115
        digit 4: 1216 1600 75 114
        digit 5: 1358 1599 67 113
```

Figure 8. UFPR-AMR Dataset Example

YOLO – YOU ONLY LOOK ONCE

Two of the state of the art algorithms for object detection are Mask R-CNN (Region based CNN) [10] and YOLO [11]. Mask R-CNN is the latest algorithm in the R-CNN family and carries out region proposal on an image before detecting the objects present within the proposed regions. It also allows for segmentation of the object within its' bounding box. The R-CNN family have high accuracy but a large memory requirement and slow inference time.

YOLO processes an image in a single stage by splitting it into a grid of cells. Each cell directly predicts a bounding box and predicted class. The result comes from all bounding box candidates consolidated into a final prediction by a post-processing step. YOLO models are best suited for real-time applications thanks to their fast inference time but are considered to be slightly less accurate.

Early project work was carried out in Google Colaboratory (Colab) to make use of Google's free Tesla K80 GPU for training the models. Both Mask R-CNN and YOLO frameworks were trained and tested using Keras and Tensorflow. Colab struggled to train the Mask R-CNN model due to the high memory requirement compared to YOLO but both frameworks showed impressive performance for our different object detection tasks. The inference time in Colab is almost 10x faster for the YOLO models, making it the preferred framework for the AMR pipeline.

DLPy supports both the Faster R-CNN and YOLO frameworks within SAS® Viya. It has a Keras-like Python interface to build and train deep learning model architectures using the SAS deep learning action set under the hood. Data can be prepared in native Python and loaded into CAS before training. The SWAT package makes a connection to CAS from a Jupyter kernel and lets us execute CAS actions and process results using Python.

SAS Viya supports converting many pre-trained Keras models into SAS®-compatible models but for this project the two YOLO models for detecting the counters and digits respectively were built from the ground up using SWAT and DLPy.

Data Preparation for CAS

After loading the UFPR-AMR dataset into SAS® Viya, the dataset must be split into training and testing sets. The manipulation of the individual text files, shown in Fig. 8, can be handled in Python before they are loaded into CAS as a CAS table. The images are handled separately in an ImageTable before being combined into a single table named `trainSet`.

For the first model, the counter detection model, each image has exactly one labelled counter. For the second model, the digit detection model, each image has exactly five labelled digits. Pandas dataframes can be created to store all labelled object information for each training image.

YOLO models take a 416x416 pixel image for input meaning that the training images must be resized and therefore the pixel-level co-ordinates will no longer be accurate. The best way to work around this is to convert the [x, y, w, h] format labels to YOLO format [x-centre, y-centre, width, height] normalized by the image height/width (between 0 and 1). In normalized format, the labels are aspect ratio independent and remain accurate when the dimensions of the image are changed.

The training labels for the counter detection model (row 3, Fig. 8) can be simply converted to YOLO format, loaded into a Pandas dataframe, and then uploaded to a CAS table. The table `YOLO_LABELS` is shown in Figure 9.

Selected Rows from Table YOLO_LABELS

	_filename_0	_nObjects_	_Object0_	_Object0_x	_Object0_y	_Object0_width	_Object0_height
0	meter0001.jpg	1.0	counter	0.468376	0.448077	0.185470	0.042788
1	meter0002.jpg	1.0	counter	0.520726	0.442188	0.120085	0.024760
2	meter0003.jpg	1.0	counter	0.450641	0.400481	0.143162	0.033654
3	meter0004.jpg	1.0	counter	0.464957	0.502163	0.116239	0.025000
4	meter0005.jpg	1.0	counter	0.505342	0.437019	0.144872	0.035577

Figure 9. Pre-Processed Labels CAS Table for Counter Detection

The digit detection model labels require more manipulation. This is because the input for this model is the output of the counter detection model, so the input images are the cropped counter displays. This means that the training images must be cropped from the original meter images and the labelled co-ordinates recalculated for the cropped images. The crops were taken using row 3 of the original labels and a small amount of padding was added to ensure the training images resembled the output of the counter detection model. The newly calculated pixel-level labels are visualized in Figure 10.

```
image height, width: (210, 924)

x_centre        y_centre        width   height
143              95              89      147
311              100             102     152
468              106             91      153
639              107             93      155
803              102             83      144

width, height, and centrepoint of each digit:
```

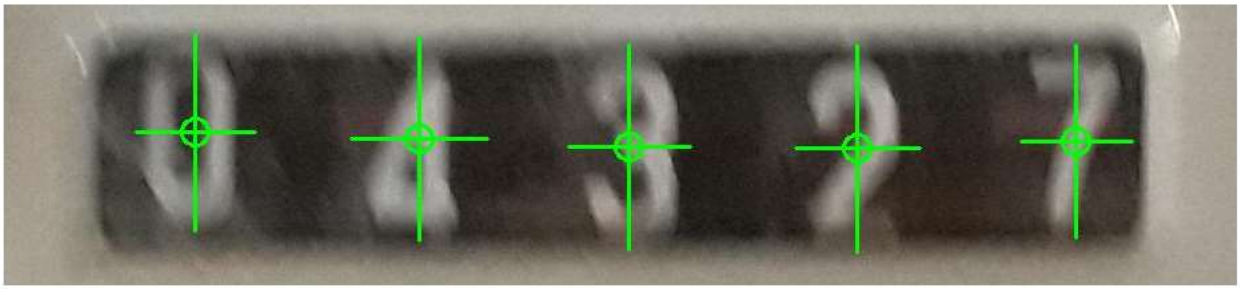


Figure 10. Digit Detector Model Training Image

The new labels can then be converted into normalized YOLO format and stored in a Pandas dataframe before being uploaded to a CAS table. The table contains columns `_Object0_`, `_Object0_x`, `_Object0_y`, `_Object0_width`, and `_Object0_height` for all five objects (digits) in each image. The column `_filename_0` must be included in each label table because it is the key that is also present in the tables that store the training images before they are combined to make each `trainSet`.

The images for both counter/digit detection models are prepared for CAS by resizing them to 416x416 pixels and uploading to a SAS-compatible ImageTable. Figure 11 shows the prepared `RESIZED_IMAGES` table containing the `_filename_0` key.

Selected Rows from Table RESIZED_IMAGES

	image	_label_	_filename_0	_id_
0	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...		meter0001.jpg	1
1	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...		meter0002.jpg	2
2	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...		meter0003.jpg	3
3	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...		meter0004.jpg	4
4	b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00...		meter0005.jpg	5

Figure 11. RESIZED_IMAGES ImageTable

In Figure 12, the SWAT `dljoin` action is used to combine the `YOLO_LABELS` table with the `RESIZED_IMAGES` table to make the `trainSet` table used to train the object detection model. This process is carried out twice to create a `trainSet` for both counter and digit detection models.

Join by Reference to CAS Table Name

```
In [18]: s.dljoin(table='Resized_Images',
                  annotation='YOLO_labels',
                  id='_filename_0',
                  casout={'name':'trainSet',
                          'replace': True,
                          'replication': 0}
                  )
```

Out[18]: **§ OutputCasTables**

	casLib	Name	Rows	Columns	casTable
0	Public	trainSet	1600	10	CASTable('trainSet', caslib='Public')

elapsed 0.122s · user 0.179s · sys 0.271s · mem 214MB

Figure 12. Using the DLJoin Action to Combine Training Data

Tiny YOLOv2 in DLPy

The Tiny YOLO detector is used in the AMR pipeline since it is better suited for portability than the full YOLO architecture. Tiny YOLO contains fewer convolutional layers and therefore less parameters making it smaller in size. There is a trade-off between model size and performance that is necessary to make the system deployable on mobile devices.

YOLO uses anchor boxes that are generated from the training data to make a prediction of bounding boxes for each grid cell of an input image – in this case the 416x416 pixel input image is split into a 13x13 grid of 32x32 pixels. DLPy contains a `get_anchors` function which creates a list of pairs of anchors from the `trainSet` using K-means clustering. Anchor boxes have particular height-width ratios that the model uses to predict target objects in an image. The five pairs of anchors generated for this task are all quite similar due to the consistency in shape of counters and digits in meter images.

The model architecture, or DAG (Directed Acylic Graph), can be built using DLPy's built-in `Tiny_YoloV2` function or built manually layer-by-layer using CAS actions. The input layer accepts an image of 416x416x3 which is then passed through many convolutional layers, batch normalization, and pooling layers before producing an output feature map with a shape of 13x13x30. The number of filters in the output layer for YOLO models is calculated as follows:

$$nFilters = (C + 5) * A \quad (1)$$

Where C = number of output classes (1), and A = number of anchor boxes (5) to produce 30 filters for each 13x13 cell that the image is split into.

Each grid cell contains 5 potential bounding boxes (since there are 5 pairs of anchors) and each of these bounding boxes has 6 elements. The first 4 elements are the YOLO format co-ordinates followed by objectness (confidence value that an object exists within the bounding box) and target class probability (either a counter for model 1, or a digit for model 2). In total, the 6 elements describing each of the 5 bounding boxes make up the 30 elements contained in each cell.

Training Tiny YOLOv2

Training any deep learning model from scratch takes a long time and may not produce the best results. A common approach for object detection networks is to apply transfer learning by using pretrained weights that are trained on large datasets such as ImageNet. These weights are transferred into our detection networks and trained on our `trainSet` data. This drastically reduces training time and tends to produce better results.

Figure 13 shows the training process for the counter detection model trained for 70 epochs. The total training time is under 8 minutes using 4 SAS Viya GPUs. DLPy's `model.fit` function uses the `dltrain` action to train the detector models and over time the loss can be seen to decrease whilst the Intersection over Union (IOU) metric can be seen gradually increasing throughout the training process.

```
yolo_model_1.set_weights('TINY-YOLOV2_WEIGHTS')

yolo_model_1.fit(data='trainSet',
                 optimizer=optimizer,
                 data_specs=data_specs,
                 n_threads=1,
                 record_seed=13309,
                 force_equal_padding=True,
                 gpu=1)
```

```
NOTE: Model weights attached successfully!
NOTE: Training based on existing weights.
WARNING: Using dataSpecs settings. Additional input or target option settings will be ignored.
NOTE: Using sasviya.viya.sas: 4 out of 4 available GPU devices.
NOTE:  Synchronous mode is enabled.
NOTE:  The total number of parameters is 11029392.
NOTE:  The approximate memory cost is 299.00 MB.
NOTE:  Loading weights cost       0.32 (s).
NOTE:  Initializing each layer cost      12.77 (s).
NOTE:  The total number of threads on each worker is 1.
NOTE:  The total mini-batch size per thread on each worker is 10.
NOTE:  The maximum mini-batch size across all workers for the synchronous mode is 10.
NOTE:  Number of levels for the target variable:      1
NOTE:  Levels for the target variable:
NOTE:  Level      0: counter
NOTE:  Epoch Learning Rate        Loss        IOU   Time(s)
NOTE:  0         0.001           8.252     0.3625     7.12
NOTE:  1         0.001           3.399     0.4342     6.50
NOTE:  2         0.001           2.892     0.5169     6.55
NOTE:  3         0.001           2.389     0.5953     6.77
...
NOTE:  66       0.0007         0.06766     0.8308     6.48
NOTE:  67       0.0007         0.03696     0.8495     6.47
NOTE:  68       0.0007         0.04868     0.8559     6.44
NOTE:  69       0.0007         0.04313     0.8588     6.68
NOTE:  The optimization reached the maximum number of epochs.
NOTE:  The total time is     456.93 (s).
```

Figure 13. Model Training Process

Testing Tiny YOLO Output and Deploying

Before the models can be strung together to build the first parts of the AMR pipeline, they must be tested in isolation to verify that the model training has been successful. Unseen test images and their labels can be loaded into a CAS table similar to `trainSet` and passed to the model to be scored in batch using the `model.predict` function. This function uses the `dlscore` action and provides an average IOU score for the whole test set calculated using the ground truth labels.

After making predictions on a batch of images, the quickest way to visualize model performance is to use DLPy's built-in `display_object_detections` function. It decodes the model output and draws the predicted bounding boxes on inference images for quick verification. The annotated images show the predicted objects as well as the model's confidence score. Some results from both counter and digit detection models are shown in Figure 14. For the final AMR system, counter predictions will be padded by 10% in width and height to ensure that the leading and trailing digits are included for further processing. The digit predictions are not padded before being passed to the classifier.

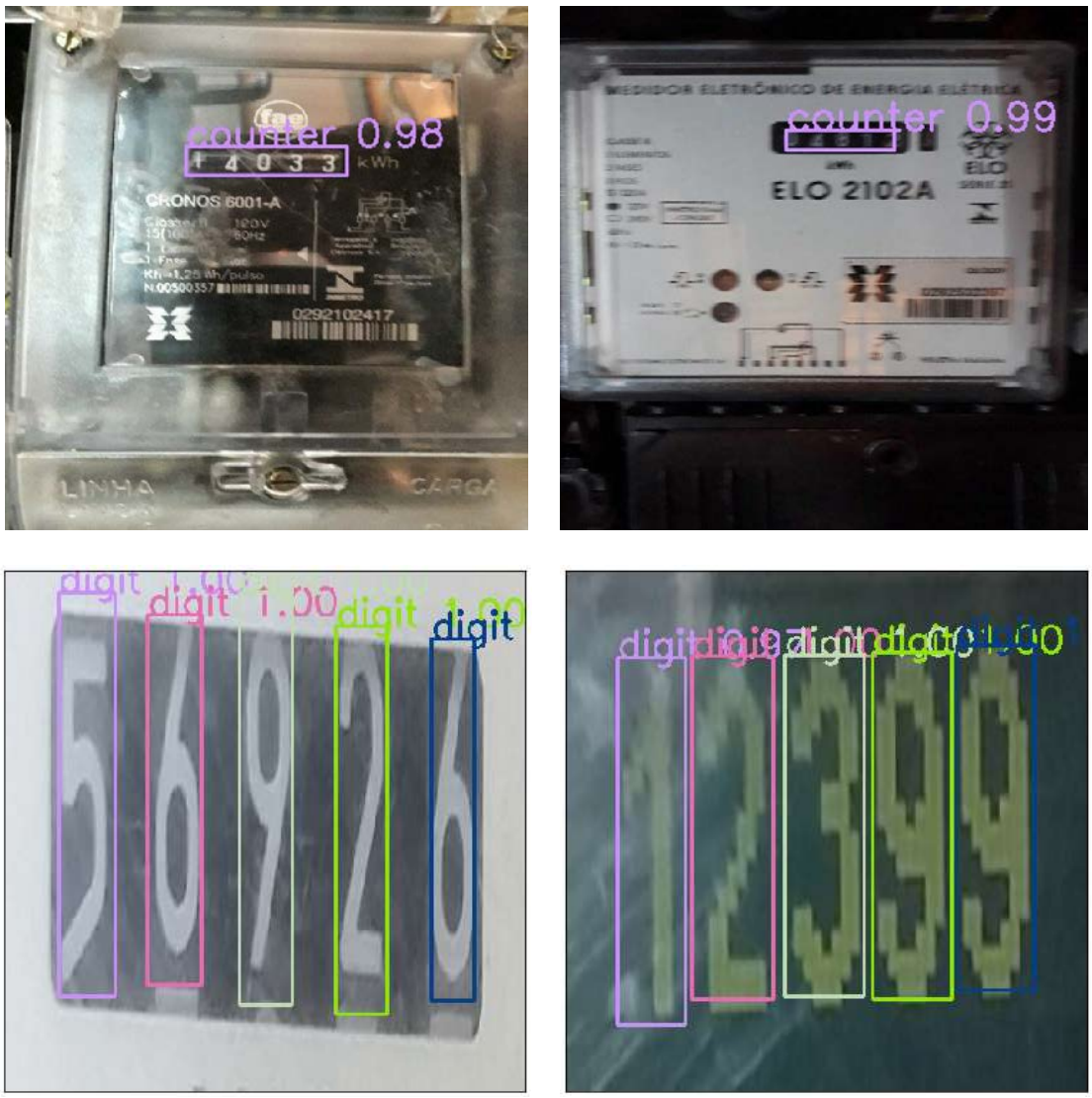


Figure 14. Verifying Tiny YOLO Model Outputs

After verifying that the models have been trained successfully, they can be 'deployed' or saved from Jupyter into the SAS Viya environment. DLPy supports deploying the YOLO models as either SASHDAT, ONNX, or ASTORE (Analytic Store) format using the `model.deploy` function. The deployment format depends on the intention of use, SASHDAT allows models to be loaded quickly and easily into the Jupyter environment from CAS using the `model.from_sashdat` function. The model can be saved in ASTORE format if it is to be scored using SAS Event Stream Processing® (ESP), or ONNX can be used for mobile deployment or scoring using the ONNX runtime library.

DIGIT CLASSIFICATION AND SYSTEM OUTPUT

The final stage in the pipeline is the digit classification model. A Wide Residual Network (ResNet) [12] that was originally used for Google's Street View House Number (SVHN) dataset was retrained for the AMR digit classification task. The ResNet takes the outputs from the digit detector model and classifies 32x32 pixel digits as one of ten possible classes. The predictions are sorted from left to right and an output reading is produced.

It was trained as a Keras model using Google Colab. Keras models can be imported into SAS Viya as HDF5 files or as ONNX files[1]. DLPy provides `Model.from_keras_model` and `Model.from_onnx_model` functions to load models into SAS Viya. Once the model has been converted into SAS-compatible format, it can be deployed as SASHDAT, ONNX, or ASTORE as above. This means we have built the end-to-end AMR system in the Viya environment using YOLO models that were built and trained using CAS combined with a classification model trained externally in Keras and integrated into a SAS-compatible format.

The system output for the UFPR-AMR prototype doesn't need much pruning because all images are known to have 5 digit reading outputs. After training on customer images and productionizing, sanity checks can be put in place using expected number of digits and checks for false-positive detections using both YOLO and ResNet confidence scores for each digit.

END-TO-END TESTING

Once all three models are saved in SAS Viya, we can test the performance of the end-to-end solution by loading them as SASHDAT files and testing them from the Jupyter interface. This allows us to recreate the process visualized in Figure 7 and make predictions on individual images from the UFPR-AMR test images. An input image is passed through each of the three models and an output reading prediction is produced. Confidence scores from each model are available for debugging purposes, as shown in Figure 15.

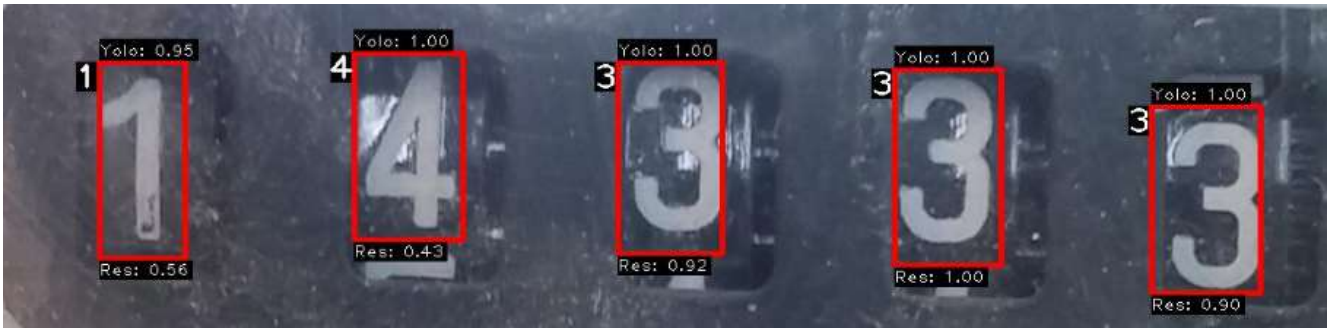


Figure 15. Digit Detection and Classification Confidences

[1] Our Wide ResNet model could only be imported into Viya in ONNX format using a workaround to deal with transpose operations.

Viya makes it possible to use CAS actions for model predictions in tandem with image manipulations such as cropping, resizing, or annotating in Python using open source tools such as OpenCV. This means we can create mockups of what a customer would see when they take a photo of their own meter. An example system output is annotated on an input image shown in Figure 15.

Once the logic is in place to handle one image end-to-end, it can be bundled into executable python scripts to make predictions on individual images or in batch using CAS from the Jupyter notebook environment. This allows us to simulate the system as a callable service such as a RESTful API.

Figure 16. System Output Displayed on Input Image

DEPLOYMENT ON MOBILE APP

Once all 3 models were created, we can deploy each of the models to an ONNX format in a simple command, using the following script:

```
model1.deploy(path="/opt/sas/viya/config/data/cas/default/public/",
              output_format='onnx')

model2.deploy(path="/opt/sas/viya/config/data/cas/default/public/",
              output_format='onnx')

model3.deploy(path="/opt/sas/viya/config/data/cas/default/public/",
              output_format='onnx')
```

As previously discussed in the paper, the ONNX format allows us to set up for model deployment, as SAS is a member of the ONNX community, you can train a model in SAS then export it to ONNX, to leverage the capability of easily moving models between different frameworks for deployment, as we will demonstrate here. Similarly, you can import an ONNX model into SAS in an identical fashion.

Converting these 3 Deep Learning Models into a singular mobile application was approached by using the ONNX format of the models and converting them into TensorFlow using the ONNX-TensorFlow package [13]. Using this, The TensorFlow Lite converter takes a TensorFlow model and generates a TensorFlow Lite file (.tflite), The TensorFlow Lite file is then deployed to a client device (in this case, the mobile app) and run locally using the TensorFlow Lite interpreter. Following this, we will have all 3 models (Counter Detection, Digit Detection, Digit Classification) converted into '.tflite' formats, which will run via the Java API for Tensorflow Lite.

The TensorFlow Lite converter, which converts TensorFlow models into an efficient form, and can introduce optimizations to improve binary size and performance (Similarly to the SAS ASTORE format). Its key advantage exists that it allows the Deep Learning models, which are usually largely sized, to easily and smoothly run at the edge, instead of sending data back and forth from a server. This method massively increases process performance, specifically on servers such as mobiles, tablets or any IoT devices.

This process was taken as it's simpler to deploy onto an Android app using Android Studio. A similar process could've been taken using CORE ML to deploy the models on iOS, using onnxruntime [14]. Although, it is worth mentioning that deployment into iOS using TensorFlow Lite is possible.

Finally, being able to deploy such models on the edge has massive potential, specifically on a mobile phone app. There exists a capability to intelligently analyze the meter image data while still controlling it on the client-side device with the potential to still open up a controlled stream of information back to the cloud to retrain the model based on appropriate new images, and then deploy improved models in inference on the edge. This application will also provide flexibility for the customer, as the potential exists to run offline and upload the data when a connection exists whilst maintaining a great performing model with a quick response time.

CONCLUSION

The AMR prototype and deployment process outlined in this paper provides a template for taking our meter reading solution into production using the SAS Viya platform. It supports the whole lifecycle of the project, from data preparation and model building to unit testing and producing the models that can be combined in a single application for mobile deployment. Viya allows developers to build and train models in Python using DLPy's high level of abstraction to utilize the SAS deep learning action set under the hood. It also supports models trained externally in Python using open source tools such as Keras.

The next step in putting this system in the hands of ScottishPower customers is to acquire and label customer-submitted images of meters. After that, the steps put in place to train models using the UFPR-AMR dataset can be replicated with the new training data. By productionizing this computer vision application using SAS Viya, we can greatly improve the customer experience in meter reading submissions. This in turn should reduce customer contacts regarding reading disputes and incorrect bills. The system is the first of its kind using an end-to-end solution for meter reading submissions in the highly competitive UK energy retail market.

REFERENCES

[1] Department of Energy & Climate Change, UK Government. 2010. "GB-Wide Smart Meter Roll Out for the Domestic Sector. Impact Assessment." Available at https://www.ofgem.gov.uk/ofgem-publications/63551/decc-impact-assessment-domesticpdf.

[2] Department for Business, Energy & Industrial Strategy, UK Government. 2019. "Smart Meter Statistics in Great Britain: Quarterly Report to end September 2019." Available at https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/848325/2019_Q3_Smart_Meters_Statistics_Report.pdf.

[3] SAS Software. 2020. "SAS® Visual Data Mining and Machine Learning 8.5: Deep Learning Programming Guide. Deep Learning Action Set: Syntax" https://go.documentation.sas.com/?docsetId=casdlpg&docsetTarget=cas-deeplearn-TblOfActions.htm&docsetVersion=8.5&locale=en

[4] SAS Software. 2020. "Python-DLPy." https://github.com/sassoftware/python-dlpy

[5] SAS Software. 2020. "Python-SWAT." https://github.com/sassoftware/python-swat

[6] Amazon Web Services. 2020. "Amazon EC2 Instance Types." https://aws.amazon.com/ec2/instance-types/

[7] Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., and Liang, J. 2017. "EAST: an efficient and accurate scene text detector." *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 5551-5560). Available at http://openaccess.thecvf.com/content_cvpr_2017/papers/Zhou_EAST_An_Efficient_CVPR_2017_paper.pdf

[8] Rosebrock, A. 2017. "Using Tesseract OCR with Python." Available at https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/

[9] Laroca, R., Barroso, V., Diniz, M. A., Gonçalves, G. R., Schwartz, W. R., Menotti, D. 2019. "Convolutional Neural Networks for Automatic Meter Reading." *Journal of Electronic Imaging, vol. 28, pp. 1-14.* Available at https://arxiv.org/pdf/1902.09600.pdf

[10] He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017. "Mask R-CNN". *Proceedings of the IEEE international conference on computer vision* (pp. 2961-2969). Available at http://openaccess.thecvf.com/content_ICCV_2017/papers/He_Mask_R-CNN_ICCV_2017_paper.pdf

[11] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788). Available at https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

[12] Zegoruyko, S. and Komodakis, N., 2016. "Wide Residual Networks". *arXiv preprint arXiv:1605.07146*. Available at https://arxiv.org/pdf/1605.07146v1.pdf

[13] Open Neural Network Exchange. 2020. 'Tensorflow Backend for ONNX.' https://github.com/onnx/onnx-tensorflow

[14] Open Neural Network Exchange. 2020. 'ONNX to Core ML Converter.' https://github.com/onnx/onnx-coreml

ACKNOWLEDGMENTS

We would like to acknowledge the continued involvement of the SAS team; Jennifer Major, Scott Bowler, Emma McDonald, Matteo Landro, Nick Heather, Matthew Stainer and Prashant Chamarty, thank you to Alexander Koller for setting up the environment and thank you to Alex Ge, Anthony Kan for supporting the deployment onto the mobile app. We would also like to acknowledge the SAS DLPy, R&D and Product Management teams for their support throughout the development period.

This project wouldn't have been possible without the support of the Data Lab and MBN Solutions liaising with ScottishPower for their MSc student placement programme. Jessica Walkenhorst, formerly of ScottishPower, was the supervisor of this project and played an important role in its success. We would finally like to acknowledge Gail Miller, Cara Tulley, and Monica Murphy at ScottishPower for the parts they played in creating this project.

CONTACT INFORMATION

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

Jonny McElhinney
ScottishPower Energy Retail Ltd
j.mcelhinney@scottishpower.com

Duncan Bain
ScottishPower Energy Retail Ltd
duncan.bain@scottishpower.com

Haidar Altaie
SAS UK & Ireland
Haidar.Altaie@SAS.com

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.

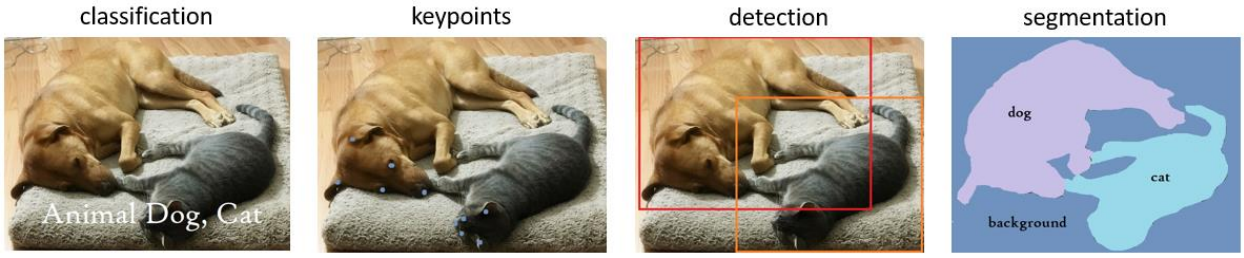


Figure 1. Basic Tasks of Computer Vision

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.

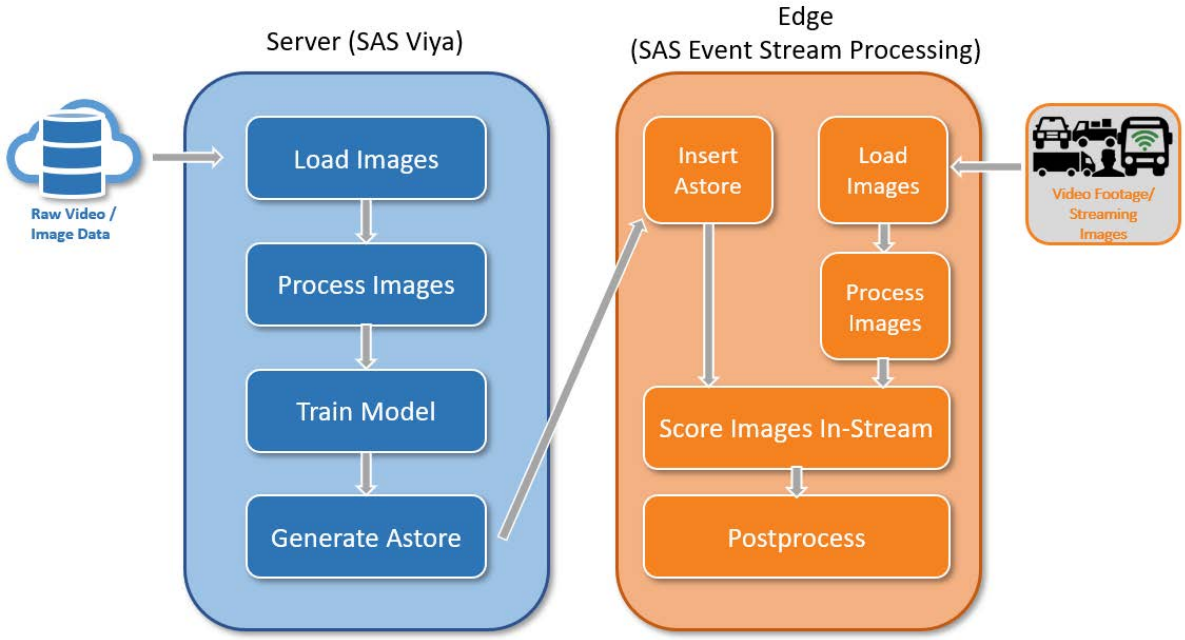


Figure 2. Computer Vision Architecture

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.

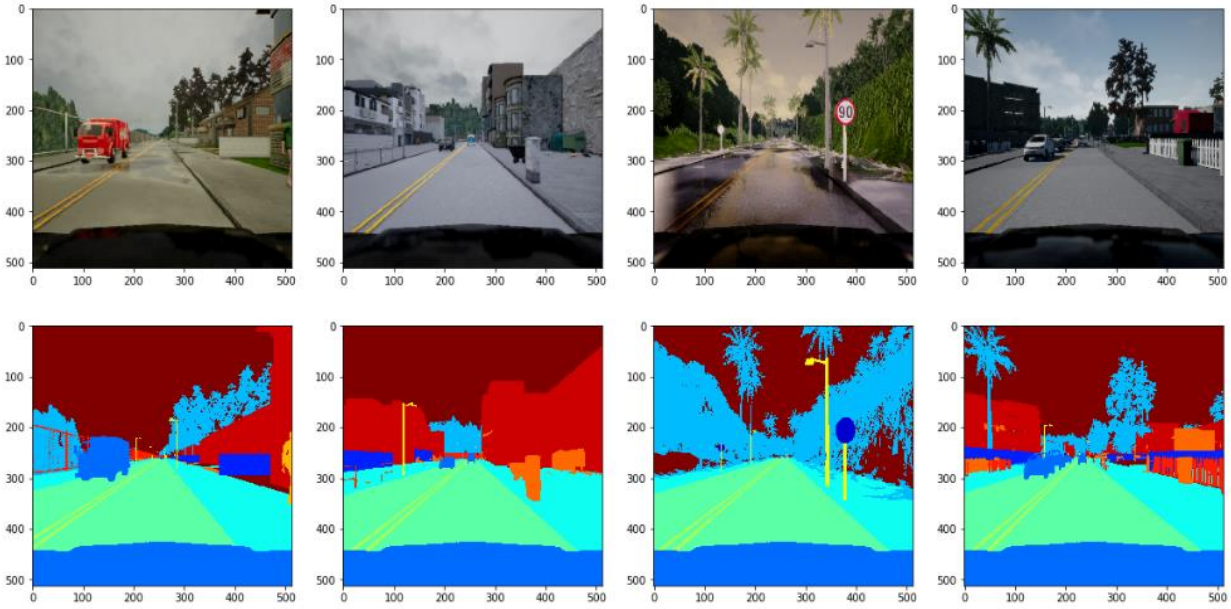


Figure 3. Training Data Visualization (raw images in the top row and ground truth masks in the bottom row)

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).

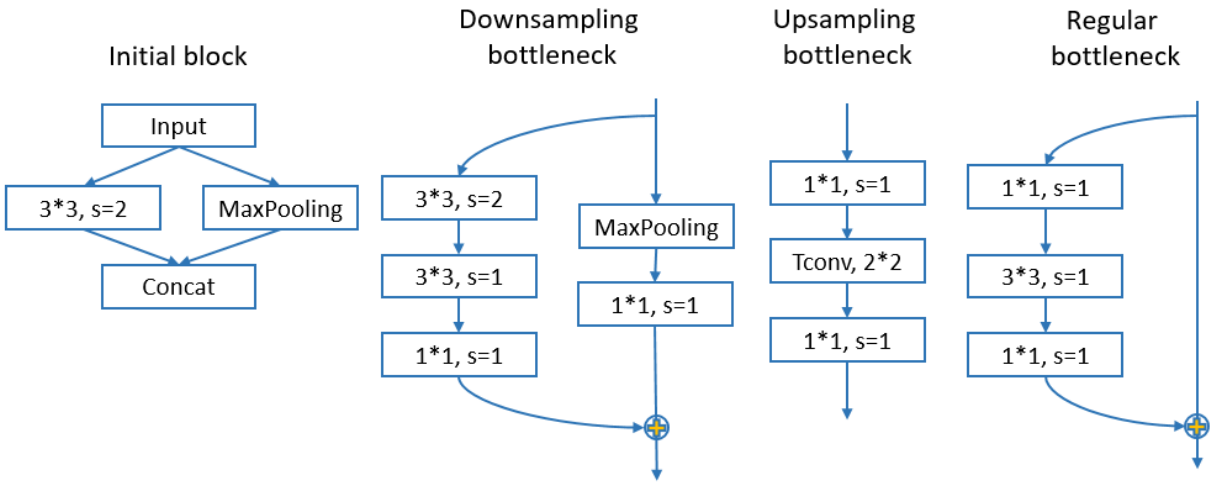


Figure 4. Diagram of Each Module

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.

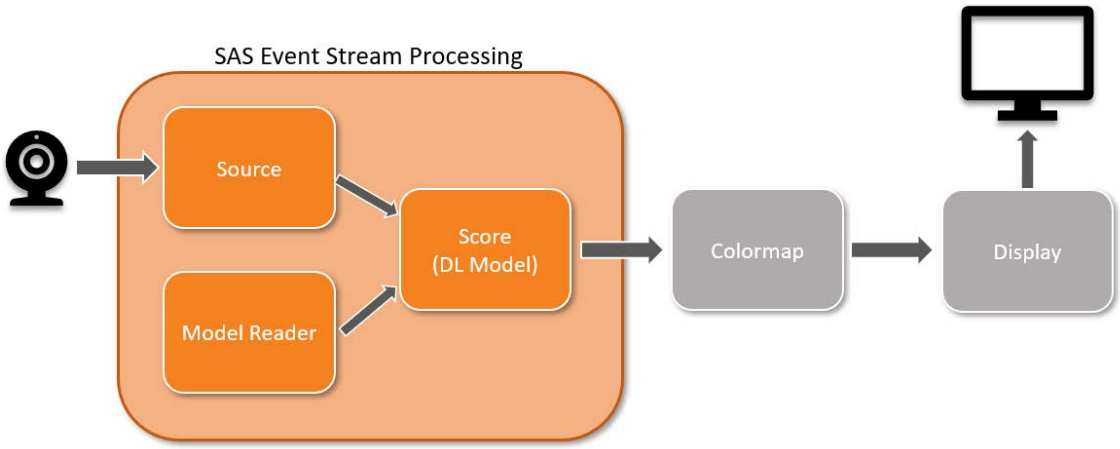


Figure 5. Scoring Flow on Edge Device

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.

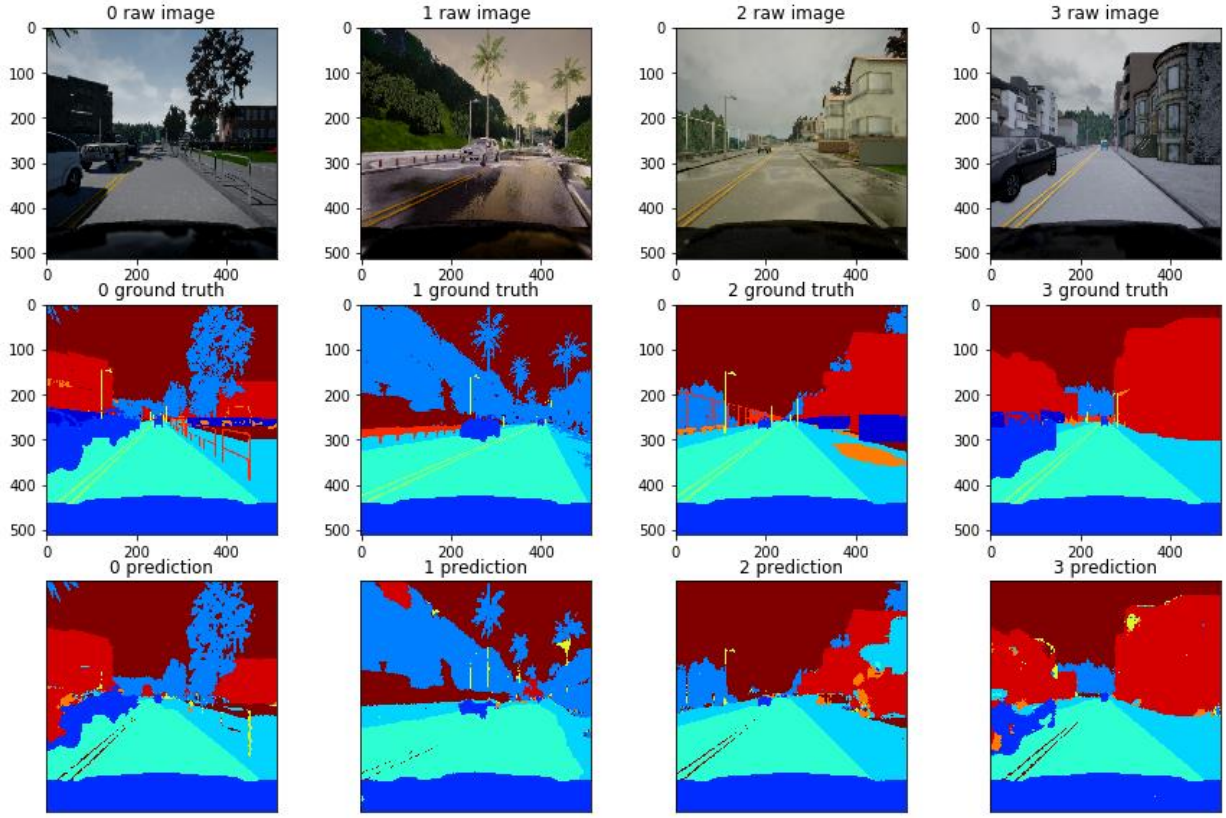


Figure 6. Prediction Using Test Images (input images in the top row, ground truth masks in the middle row, and predicted images in the bottom row)

	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.

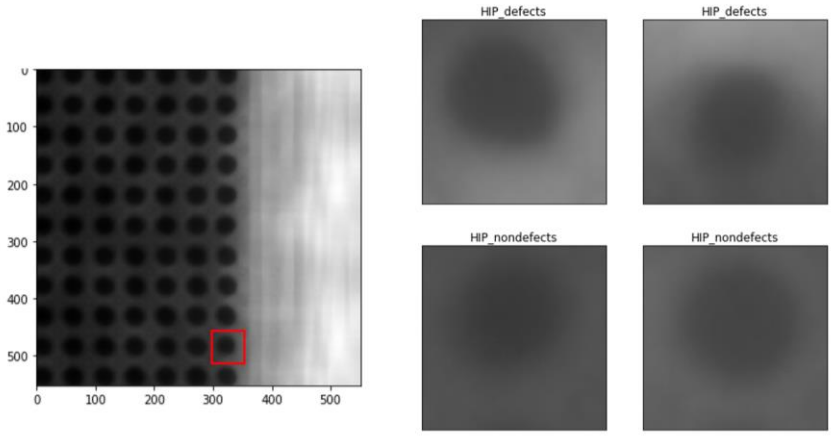


Figure 8 (Left) Defective Joint Shown in the Enclosed Red Box. (Right) Cropped Training Samples That Consist of Two Defective and Two Nondefective Joints

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.

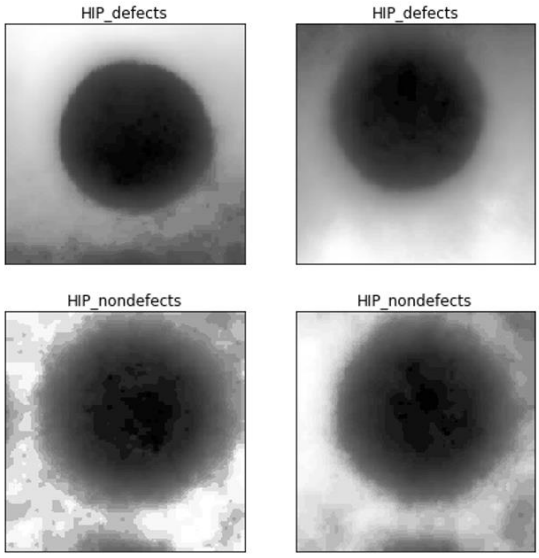


Figure 9. Histogram Equalization Applied to the Raw Image Data

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.

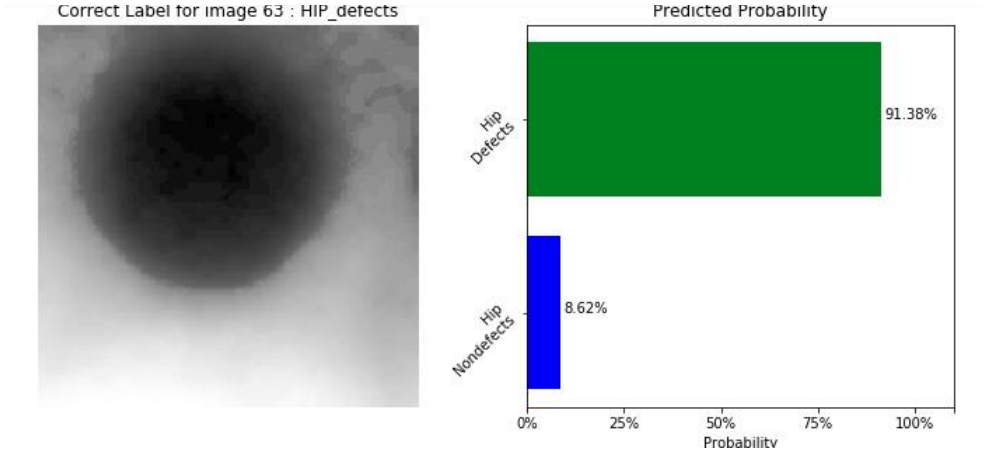


Figure 10. Scoring Hold-Out Data Using a Trained VGG-16 Model That Shows Correct Classification of a HiP Defect

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.

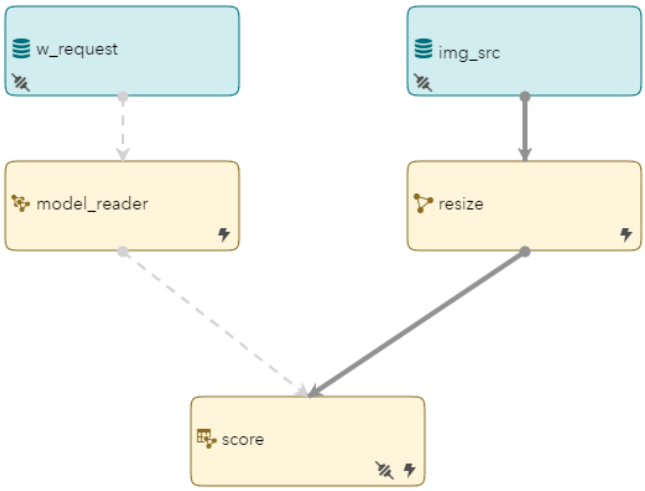


Figure 11. Process Flow Showing Input of an Analytic Store and Scoring New Data Using SAS Event Stream Processing Studio

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.

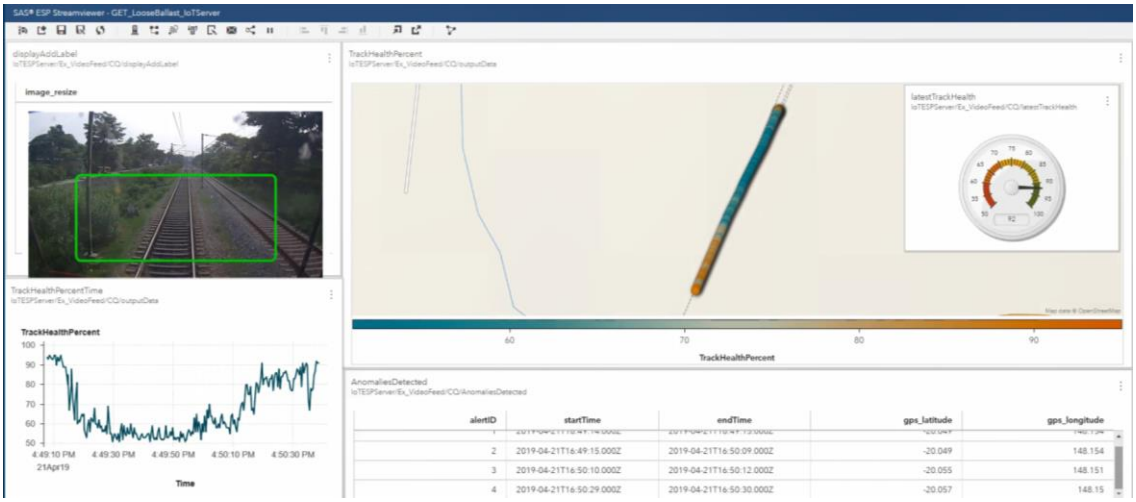
Typically, trains have a camera mounted to the front that can capture high-definition video of the railway tracks. Being able to run SAS Event Stream Processing on an edge device (a camera that has enough computing power) and store the raw video feeds along with additional information such as geolocation, 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

Paper SAS3278-2019

How to Use Deep Learning with Your Internet of Things (IoT) Digital Twin

Brad Klenz, SAS Institute Inc.

ABSTRACT

With the Internet of Things (IoT), a digital twin is created to have a virtual representation of a remote device or system. The digital twin shows you the device's operating condition, no matter where it is physically located. IoT devices have a number of sensors installed on them, as well as sensors for the environment around them. Analytics can bring this sensor data together to create a true real-time digital twin. A previous paper showed how streaming analytics are used for device state estimation and anomaly detection. This paper explains how deep learning can be added to your digital twin for more understanding. Image and video analytics are used to capture operating conditions that are missed by regular sensors. Recurrent neural networks (RNN) add temporal data analysis and pattern detection in real-time data streams that are prevalent in digital twins. With these deep learning capabilities, your digital twin provides a new level of insight for your remote devices.

INTRODUCTION

Deep learning is becoming more prevalent, with some typical use cases emerging. Natural language processing (NLP) is used with personal assistants and chatbots. Computer vision is used for object recognition in autonomous driving and detecting tumors on medical images. Facial recognition is used for access control. Reinforcement learning is being researched in game applications.

Now research is emerging on industrial IoT applications that will help augment existing applications of digital twins. Digital twins are a virtual representation of a physical asset or device, frequently in a remote location.[1] With IoT, data is collected from sensors on a device, on neighboring devices, the environment around a device, and whatever interacts with the device. The speed is real time, and connectivity allows us to span distances instantly in many cases. Advances in streaming analytics now enable us to process this real-time data using machine learning and artificial intelligence.

To add more value to your digital twin, deep learning can be added for specific use cases. These use cases are fairly targeted based on typical sources of data found in digital twin applications. Further in the future, artificial general intelligence (AGI) will be able to use the entirety of data from your digital twin for more general AI application.

Here are some of the deep learning techniques shown in this paper:

- Computer vision – Using images and video to provide insight not easily available with existing sensors. Cameras can sometimes be installed much more easily than other sensors. Cameras can also be retrofitted to existing assets passively, where adding sensors can more invasive.
- Recurrent neural networks (RNN) – Sensor data frequently captures measurement over time, and we are looking for issues that develop over time. RNNs can provide complex pattern recognition as well as specialized forecasting.

- Reinforcement learning (RL) – Your digital twin is frequently a twin of a physical asset that is being controlled for optimal operation. The output of the physical asset is typically captured. Using RL, we can learn how the controls of the asset can be set to learn how to achieve optimal output.

REAL-TIME APPLICATION OF DEEP LEARNING IN YOUR DIGITAL TWIN

Deep learning is very compute intensive. The deep learning models are trained on large databases and are almost always done offline. It's not unusual to take hours or days to train a model. Once the model is trained, the application of the model through inferencing is less compute intensive, but still requires more compute resources than is typical for digital twin applications. For some applications, near real-time or slightly delayed results are sufficient. For example, in the computer vision defect detection described below, it might be acceptable to hold a production batch while the defect detection is performed. In other cases, real-time inferencing is needed. Inferencing can be done in the cloud or data center where sufficient resources are readily available. For edge inferencing, edge gateways are now becoming available with sufficient compute power, but you must plan for this specialized need.[2,3]

COMPUTER VISION

Popular uses of computer vision include facial recognition and object detection. To see where computer vision can help a digital twin, look for applications that would require visual inspection. Here are some examples:

- Defect detection in semiconductor manufacturing – In semiconductor manufacturing of wafers and dies, many tests cannot be run until the packaging phase. With computer vision it is practical to take images of the wafer earlier in the production process and inspect it for issues. The inspection can be used to find defects and determine the number and location. This will allow an earlier determination of final yield from the wafer. With previous labeled images of diagnosed defects, it is also possible to classify the defect types using computer vision. This will help augment a larger root cause analysis for process improvement.

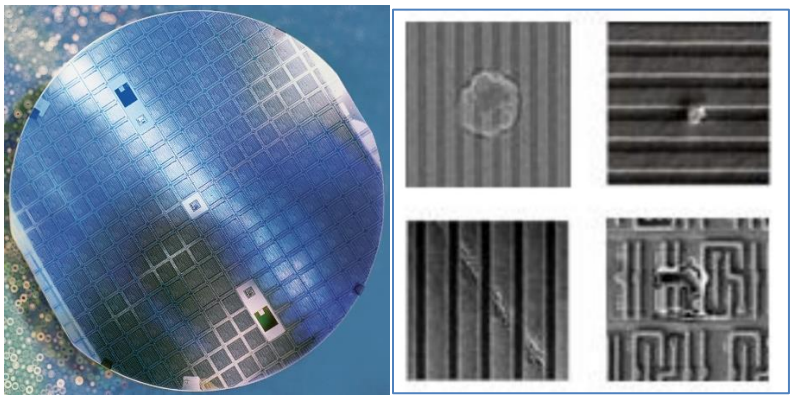


Figure 1 Silicon Wafer and Wafer Defects

- Defect detection in discrete parts – With visual inspection of discrete parts, you can more easily catch a number of production defects. These are various issues in production quality. For example, in aerospace and automotive parts production, you can determine incomplete finishes or casting issues.

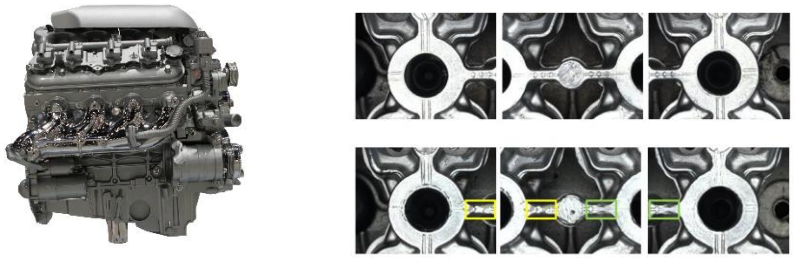


Figure 2 Automotive Engine and Casting Defects

- Infrared patterns for heat buildup in power substations – Using specialized cameras, you can capture images for different spectrums. Infrared cameras can capture a more complete picture of temperature deviations and patterns than what would be possible with individual temperature sensors. This allows for new applications such as monitoring power substations for components getting ready to fail.

IMPLEMENTING A COMPUTER VISION MODEL

The process for implementing a computer vision model is as follows:

- If possible, fix the camera to a stable mount point so that all images will be taken from the same angle and with the same proportions. This vastly simplifies the model training as compared to general object recognition models, which must capture objects from many angles. The fixed camera location also simplifies the process of determining the location of defects on the piece.
- Another option is to initially create a model that finds easily identified features on the piece. For the power substation example, you could have general instructions on how to point the camera at a transformer in the substation. An object recognition model could identify the bushings on the top of the transformer. This would provide reference points to scale the images with images captured at similar angles. This is similar to how facial recognition models determine the various key points on a face.[4,5]
- In the case where issues develop or occur over time, a video can create a large number of images, both of known good cases and defect cases.
- You will use the images to create a classification model using Convolutional neural networks (CNN). Depending on how well labeled your data is, you can create models of various complexity.
    - If you primarily have a collection of known good images, you can create a binary classification model that identifies images with a high likelihood of known good or suspected anomaly images. The power transformer is an example of this.
    - If you have images that have been labeled with known defect types, you can create a more complex classification model that identifies the various defects. The discrete parts are an example of this. There might be previous images labeled with an incorrect bearing insertion, and other images labeled with incorrect part milling.
    - If you have good location identification, you can also break down the images and find the locations of portions of the image with defects. The

semiconductor wafer is an example here. This would allow you to quantify the expected yield based on the proportion of the wafer with defects.

- When you have trained the model, you can determine at what latency you can infer and test new images being captured. Determine if you need to stream image-by-image and get immediate results. Alternatively, you might be able to capture a batch of images and process in batch. Also determine if the inferencing can be done in the cloud or server, or if an edge gateway is needed.
    - The power transformer example might be a good candidate for edge inferencing. Since the transformers are located in remote locations with reduced or sporadic network bandwidth, an edge device could process the images at the substation. In addition, only rarely will there be an issue requiring attention. Processing at the edge would eliminate the need for a large amount of image transfer for rare events.
    - The discrete parts application would be a good example of real-time inferencing in the cloud or server. The factory installation will allow high quality network connectivity. The low latency would allow defective parts to be identified immediately and be removed before being used in subsequent assemblies.
    - The semiconductor example is a batch process that would fit well with batch inferencing. Since each step in the process is costly, it would be worth the time needed to hold the batch until it could be verified in full detail.

Computer vision is a technique that will have many applications for digital twins.[6,7,8]

RECURRENT NEURAL NETWORKS

Recurrent neural networks (RNNs) are a special class of deep learning neural networks designed for sequence or temporal data. An area where RNNs are popular is for natural language processing. In this application, large text and document databases are modeled to discover common word sequences in context. The model can then be used to generate new messages for the appropriate context. This is seen in chatbot applications. Within IoT and digital twins, there are many examples of such sequence and temporal data. Many sensors are collecting data over time. The sequence or pattern of the measurements over time can be used to understand interesting characteristics of the digital twin asset. One example is measuring energy circuits in a smart building or power grid. The pattern of the energy use on a circuit can capture the start or end of an asset operation, such as a motor start, which signals an operation change in the digital twin asset. Another use of RNNs is for forecasting unusual time series data. An example is forecasting the energy output from a solar farm.

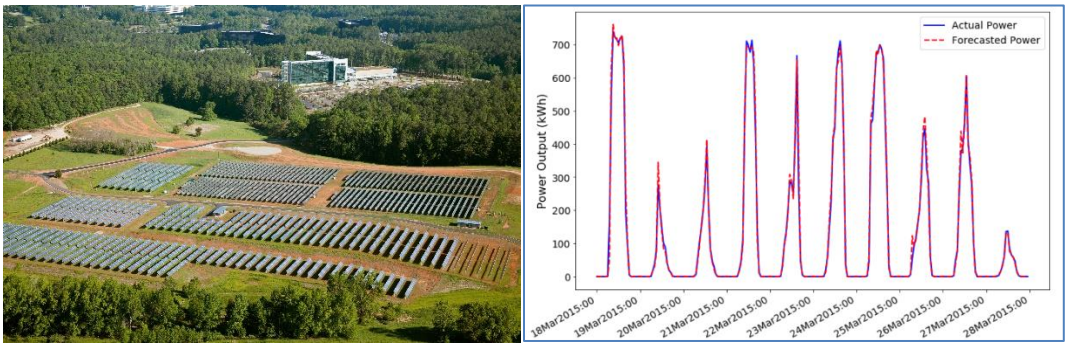


Figure 3 Solar Farm and Power Output Chart

In this case, there is a cyclical component that could be forecasted using traditional methods, but there is a less well modeled component of weather and cloud cover. With the large amount of data available from the solar farm and nearby solar farms, a deep learning RNN can capture the more sporadic aspects of the energy output.[9]

The process for training an RNN is different if you are working with sequence data versus working with temporal data.

The process for training the RNN with sequence data is as follows:

- Break the data into segments of sequential measurements. The length of the segment is determined by the time interval of the data and the expected duration of the precursor to an event. For the energy circuit example in smart buildings, the data is collected at 5-second intervals, and we use the previous minute of data.
- Create a target variable for the events of interest and use it to label the sequences where the event occurs. For our example, we are using motor starts and identifying weak motor starts indicating capacitor failure.
- Train the RNN. Note that RNN training has a feature for bidirectional model fitting. That is useful for natural language processing applications where words can be in a different sequence, but still indicate the same event, like a positive product review. Since our measurement data is always moving forward in time, we do not need to use bidirectional model fitting.
- The trained model can then be deployed for inferencing. In most cases, the model inferencing function will be sufficiently fast to be used on the real-time measurement stream, either in the cloud, server, or edge device.

The second type of RNN is used to forecast. The example in this case is to forecast the energy output of a solar farm for short time periods in the future (1 hour). The key in this case is to create a set of lagged variables for the predictors and the response variable. The response variable is the energy produced. The steps for training this RNN are as follows:

- Take the historical input database and create lagged variables for the predictors and response variable. The number of lags is determined by the time interval of the measurement data and the expected correlation of previous measurements on the forecast time horizon. For the solar farm example, we are producing 1-hour-ahead forecasts, and the data over the last few hours is sufficient to capture the primary effects for the forecast. Note that there are a large variety of conditions possible throughout the year and previous observed weather, even though the forecast horizon is fairly short. Since we have a large amount of historical data of the various conditions, the use of an RNN is appropriate for this problem.
- When creating the lags, you should evaluate your data for missing values and consistent time intervals. The ideal case is data collected at consistent intervals with few missing values. If the data has a large number of missing values or inconsistent collection intervals, you can use the TIMEDATA procedure to improve the data for training.
- Since training and evaluating the RNN model is dependent on the sequence, partitioning the data requires more care than typical random partitioning. In this case, we need to preserve the sequence of the data for use in the model creation steps (training, validation, test). The easiest way to do this is to partition the data based on the time variable. Use the earliest historical data for the training data set. Then use the next time partition for the validation data set. Finally, use the most recent data for the testing data set. This is sufficient if the performance of the asset has been consistent over the historical data sample. If there have been periods of

degraded performance, it is best to eliminate that data from the data sets used to create the model.

- Train the RNN. Note that some predictors might be estimates that you can capture from other sources. A major factor in solar farm energy forecasting is weather conditions such as cloud cover. In this case, you will want to include these predictors from a source that can provide actual and estimated values.
- For the solar farm example, we are doing short term forecasting to be used in energy load and generation balancing. Longer term demand planning is done through a separate, more traditional, forecasting process.
- You can use RNNs for one-step-ahead forecasting, where the forecast interval matches, or is less than, the desired forecast interval. This will yield the most accurate forecast. In some cases, you might need a multistep forecast to forecast future time periods based on the near term forecast estimates. These forecasts are typically less accurate but can be tested to determine if they have sufficient accuracy.

RNNs are a valuable deep learning technique for IoT digital twins.[10]

REINFORCEMENT LEARNING

Reinforcement learning (RL) is a subfield of machine learning and deals with sequential decision-making in a stochastic environment. In any RL problem, there is at least one agent and an environment. The agent observes the state of the environment and takes and executes a decision. Environment returns a reward and a new state in response to the action. With the new state, the agent takes and executes another action, environment returns reward and new state, and this procedure continues iteratively. RL algorithms are designed to train an agent through this interaction with the environment, and the goal is maximizing the summation of rewards.

RL has recently got a lot of attention due to its successes in computer games and robotic applications.[11,12] Beside the simple RL applications, there are still few real-world applications of RL to increase efficiency. We studied and did some research to extend an RL algorithm for controlling the heating, ventilation, and air conditioning (HVAC) systems. HVAC includes all the components that are supposed to maintain a certain comfort level in the building. Buildings consume 30% to 40% of all consumed energy in the world, so that any improvement could result in a huge saving in energy consumption and CO2 release.[13] Advances of the new technologies in recent years have improved the efficiency of most components in the HVAC systems. Nevertheless, still there are several directions to reduce the energy consumption by controlling different decisions on these systems.

There are two general categories of HVAC system, single and multizone. The single zone problem refers to an area that uses an HVAC system (for example, a heater or an AC system that is installed in an office), where the main control decision is the temperature set-point or just the binary action of turning the device on or off. In multizone systems, a central HVAC system supports several zones (for example, offices, hallways, conference rooms), and with a given set-point for each zone, the system needs to maintain a certain comfort level in each zone, while there are many possible control decisions.

We considered a multizone system and selected the amount of air flow as the main control decision. Using the obtained data from a building at SAS in Cary, NC, we trained an environment and used it to train an RL algorithm when there are 10 zones in the system with a set-point of 72 with ±3 allowance. Figure 4 shows the results of 50 cases with different initial temperatures. The upper figure is the temperature and the lower figure is taken actions over 150 minutes, in which every three minutes a decision is taken. We

compared this result to the commonly used rule-based algorithm (in which the system is turned on/off at 69/75) and RL obtained 47% improvement on combination of obtained comfort and energy consumption.

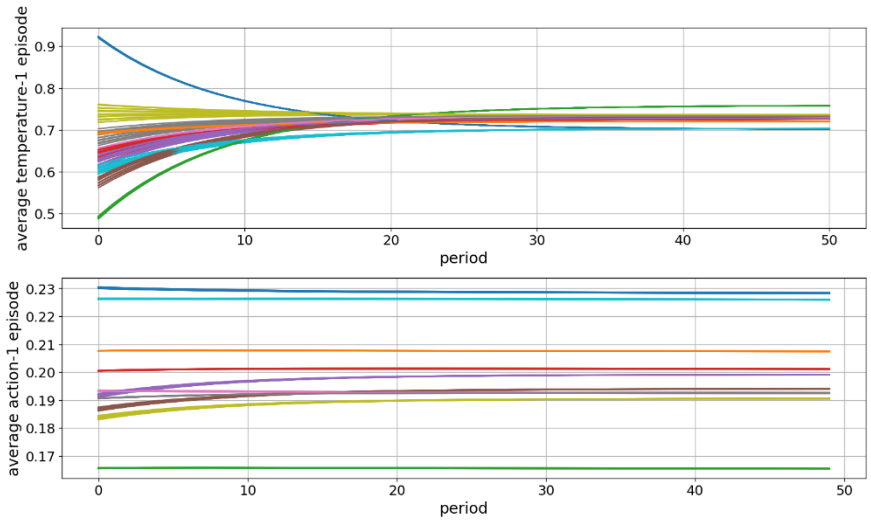


Figure 4 The Average Temperature and Average Action of the RL Algorithm

HYPERPARAMETER TUNING

For all deep learning methods, hyperparameter tuning is an important step. Hyperparameter settings are often dependent on the domain knowledge of the application. Research into the specific application can yield a set of parameter settings to be tested. In some cases, a set of parameter settings has been established as best practices. In other cases, research will be needed to determine the best settings.

One feature in SAS® Visual Data Mining and Machine Learning is hyperparameter autotune. This feature will take a range of potential parameter settings and perform an optimal search for the best performing settings. This will greatly help cases where research is needed on the parameter settings.[14,15]

CONCLUSION

Use of deep learning is growing into new application areas, and IoT digital twins are an application that will benefit from this growth. The applications for deep learning are different for digital twins than the typical applications seen today, like autonomous driving, facial recognition, and natural language processing. Instead, computer vision will be used to perform visual inspection for defect detection and identification. Recurrent neural networks will be used on the time series data streams that are prevalent in IoT to find complex pattern sequences and forecast where effects of environmental variables can be discovered with the large amount of data in IoT. Reinforcement learning will find many applications in IoT as assets in rich environments can be controlled by the algorithm, and results can be measured to determine the benefits achieved.

NOTES

1. Klenz, Brad. 2018. “How to Use Streaming Analytics to Create a Real-Time Digital Twin.” *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/2018/2004-2018.pdf
2. Williams, David. 2019. “NVIDIA Graphics Processing Units Accelerating SAS® Analytics.” *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/2019/3618-2019.pdf
3. McGrath, Dylan. 2018. “New Architectures Bringing AI to the Edge.” Available: https://www.eetimes.com/document.asp?doc_id=1333920
4. Long, Xindian. 2018. “Understanding Object Detection in Deep Learning.” Available: https://blogs.sas.com/content/subconsciousmusings/2018/11/19/understanding-object-detection-in-deep-learning/
5. Long, Xindian. 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
6. Sethi, Saratendu. 2018. “Computer Vision Using SAS® Deep Learning.” Available: https://www.sas.com/offices/pdf/ax-2018-milan/sas-ax18-sethi.pdf
7. Gong, Julia. 2019. “Using Deep Learning for Tumor Segmentation in Medical Images.” Available: https://blogs.sas.com/content/subconsciousmusings/2019/02/15/using-deep-learning-for-tumor-segmentation-in-medical-images/
8. Gong, Julia. 2019. “Constructing the Front of the Computer Vision Pipeline.” Available: https://blogs.sas.com/content/subconsciousmusings/2019/02/28/constructing-the-front-of-the-computer-vision-pipeline/
9. Kahler, Susan. 2018. “Using Deep Learning to Forecast Solar Energy.” Available: https://blogs.sas.com/content/subconsciousmusings/2018/07/05/deep-learning-forecasts-solar-power/
10. Qi, Yui. 2018. “Recurrent Neural Networks: An Essential Tool for Machine Learning.” Available: https://blogs.sas.com/content/subconsciousmusings/2018/06/07/recurrent-neural-networks-an-essential-tool-for-machine-learning/
11. Hao, Karen. 2019. “The Rise of Reinforcement Learning” in “We Analyzed 16,625 Papers to Figure Out Where AI Is Headed Next.” Available: https://www.technologyreview.com/s/612768/we-analyzed-16625-papers-to-figure-out-where-ai-is-headed-next/
12. Burda, Yuri. 2018. “Reinforcement Learning with Prediction-Based Rewards.” Available: https://blog.openai.com/reinforcement-learning-with-prediction-based-rewards/
13. U.S. Department of Energy, 2008, “Energy Efficiency Trends in Residential and Commercial Buildings” Available: https://www1.eere.energy.gov/buildings/publications/pdfs/corporate/bt_stateindustry.pdf
14. Koch, Patrick, et al. 2018. “Autotune: A Derivative-Free Optimization Framework for Hyperparameter Tuning.” Available: https://www.kdd.org/kdd2018/accepted-papers/view/autotune-a-derivative-free-optimization-framework-for-hyperparameter-tuning
15. Koch, Patrick, Brett Wujek, and Oleg Golovidov. 2018. “Managing the Expense of Hyperparameter Autotuning.” *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/2018/1941-2018.pdf

ACKNOWLEDGMENTS

The author wishes to acknowledge Afshin Oroojlooy for the section on Reinforcement Learning. His work will be very beneficial to the advancement of digital twins.

CONTACT INFORMATION

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

Brad Klenz
SAS Institute Inc.
Brad.Klenz@sas.com