SAS® GLOBAL FORUM 2020 #SASGF

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

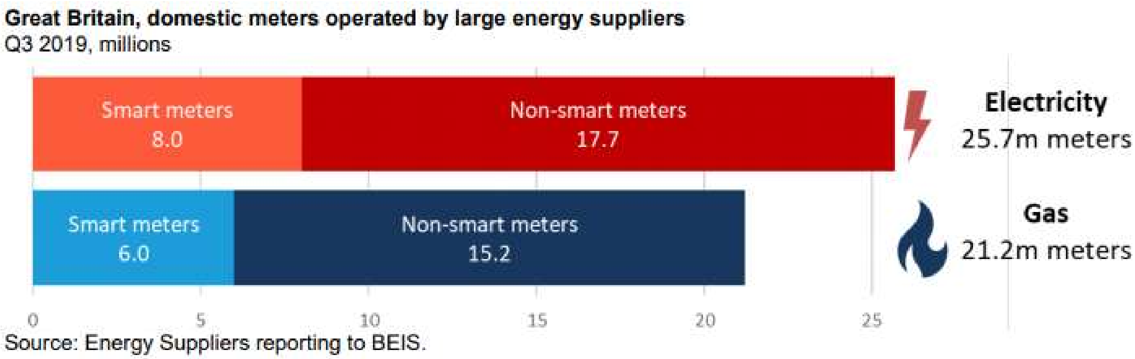
This paper proposes an end-to-end AMR system which can be deployed to mobile devices using the ONNX (Open Neural Network Exchange) framework. ONNX allows models to be trained with one framework, converted to ONNX, and deployed with another. ONNX presents a realistic solution to the task of deploying 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.


Great Britain, domestic meters operated by large energy suppliers
Q3 2019, millions

Source: Energy Suppliers reporting to BEIS.

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.

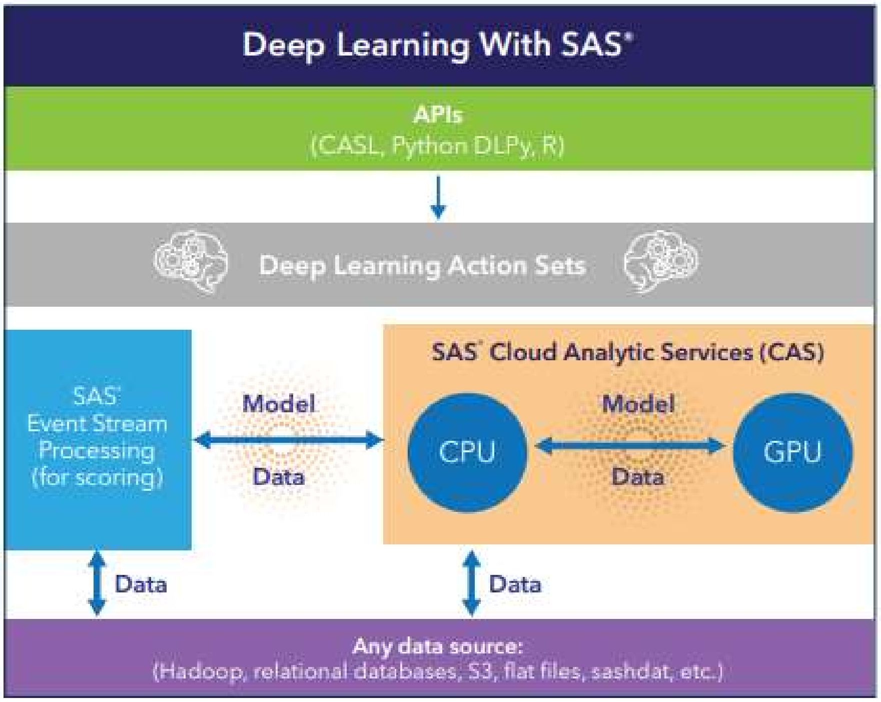

Deep Learning With SAS®
APIs
(CASL, Python DLPy, R)
Deep Learning Action Sets
SAS® Event Stream Processing (for scoring)
Model
Data
SAS® Cloud Analytic Services (CAS)
CPU
Model
Data
GPU
Data
Data
Any data source:
(Hadoop, relational databases, S3, flat files, sashdat, etc.)

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.

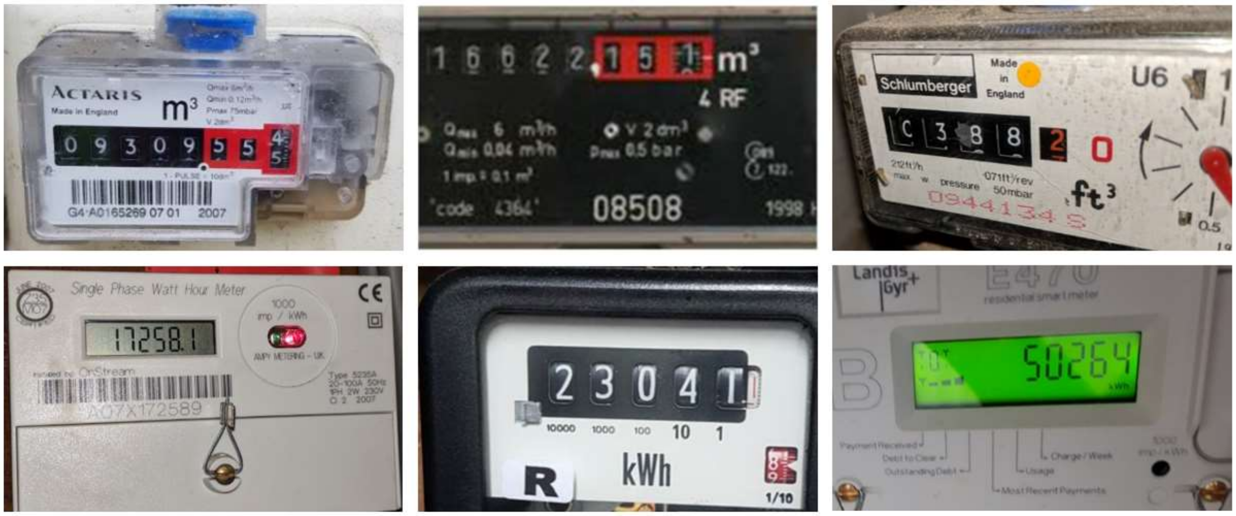


Figure 3. UK Gas (Top) and Electricity (Bottom) Meters

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.

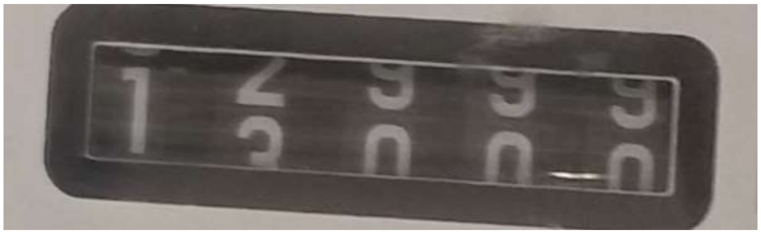


Figure 4. Scrolling Digits

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.

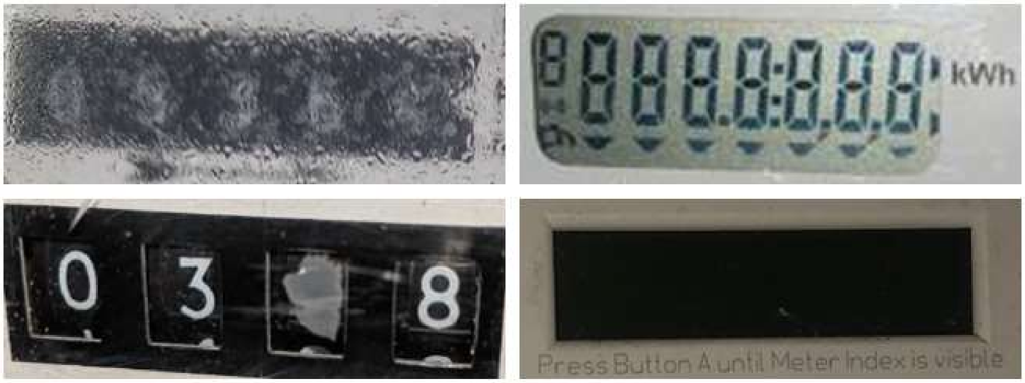


Figure 5. Low Quality Meter Images

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.

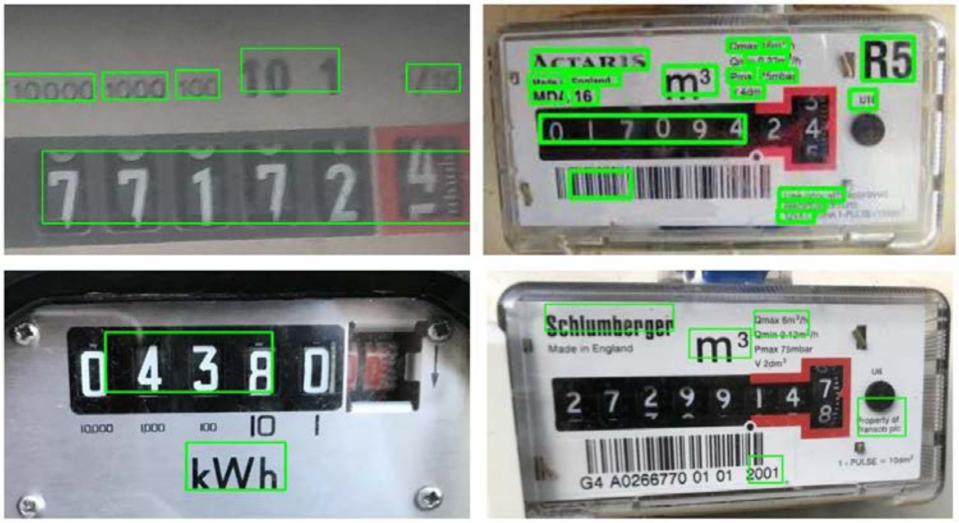


Figure 6. Example EAST Detections

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.

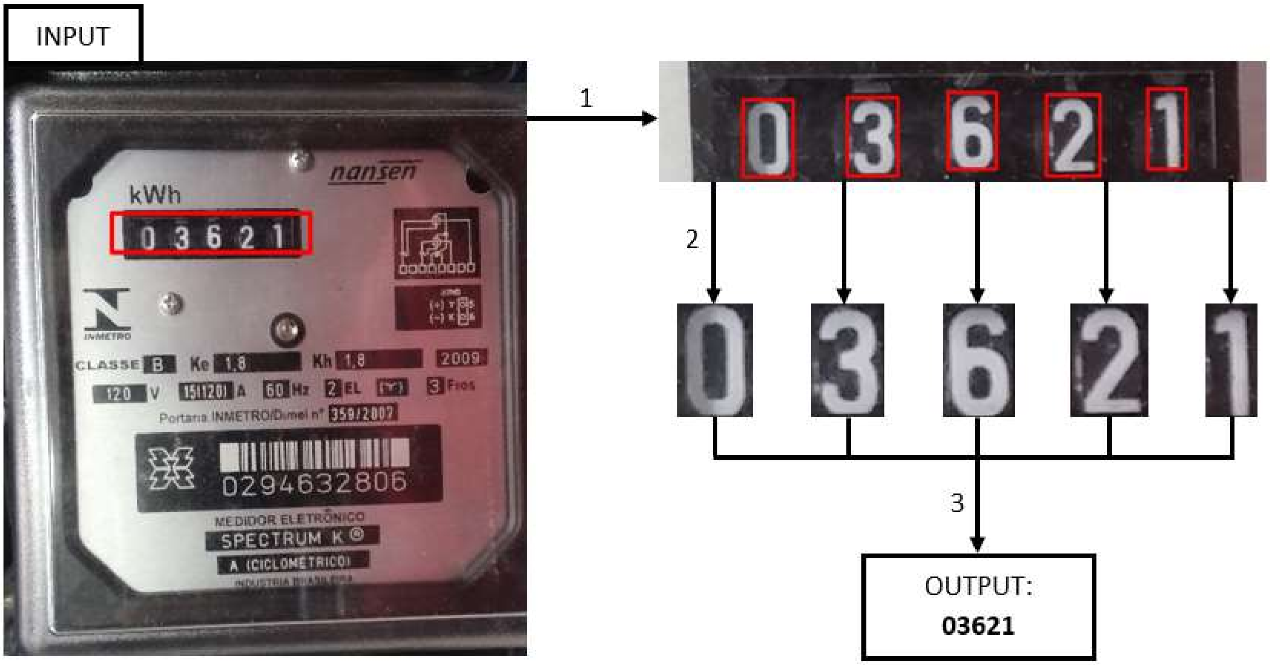
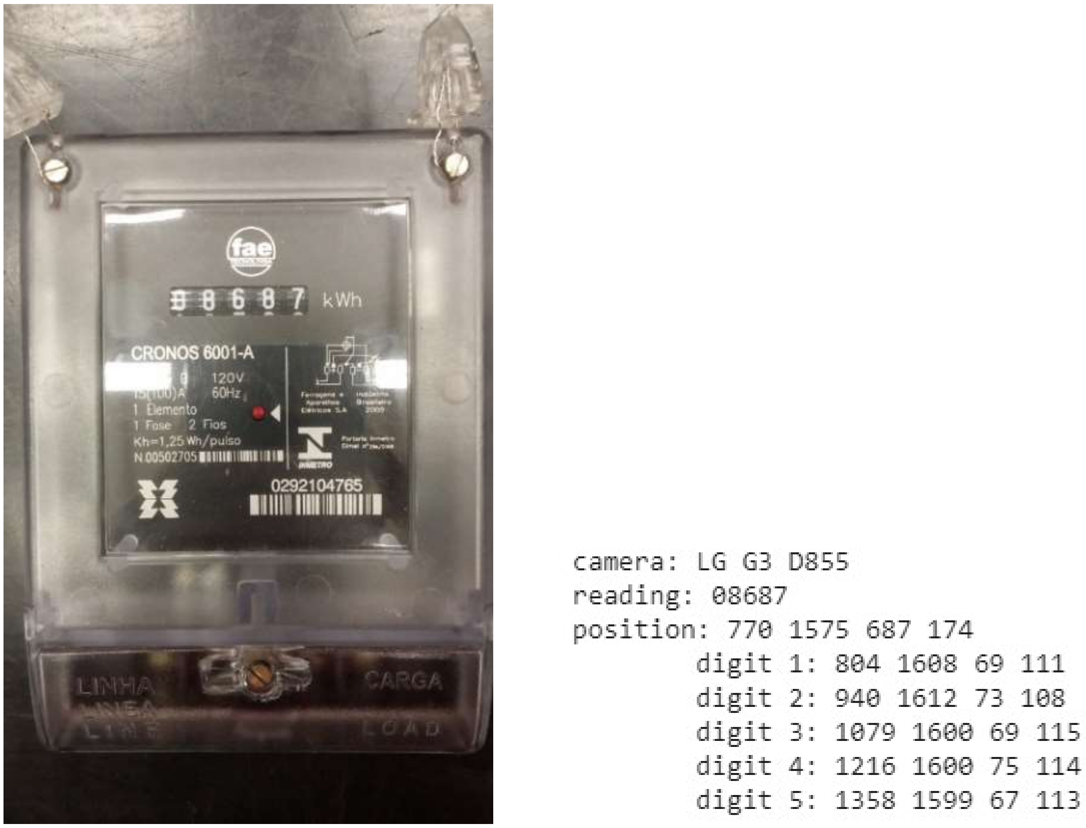


Figure 7. System Flow Diagram

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.

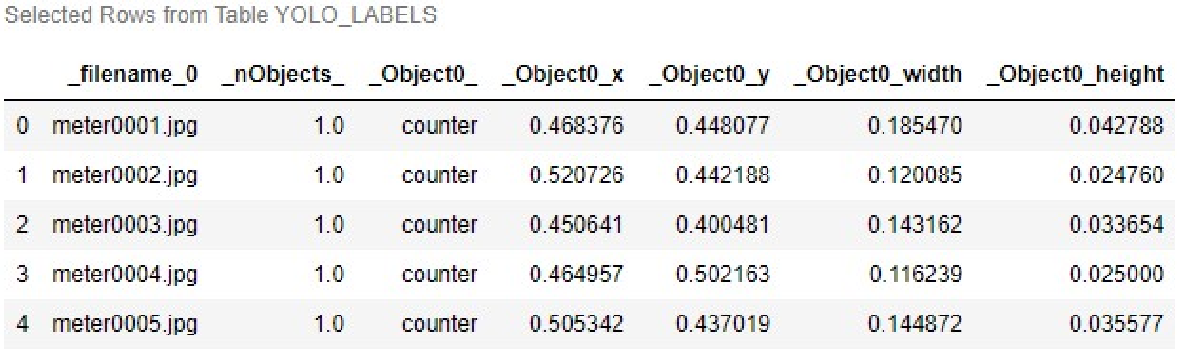
Data Preparation for CAS

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.

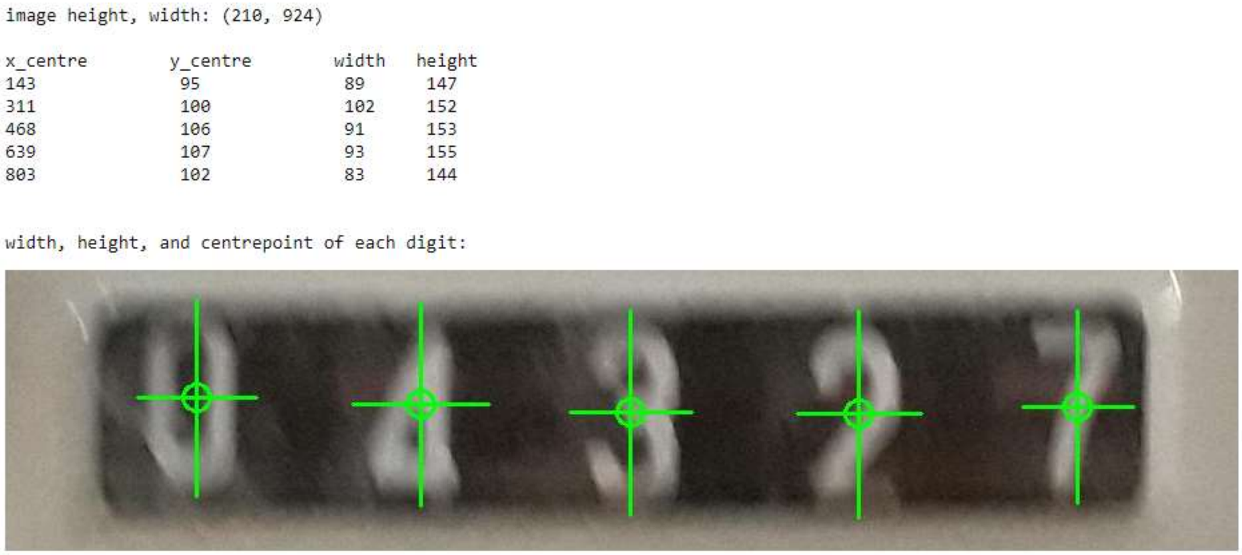


Selected Rows from Table YOLO_LABELS

	_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.

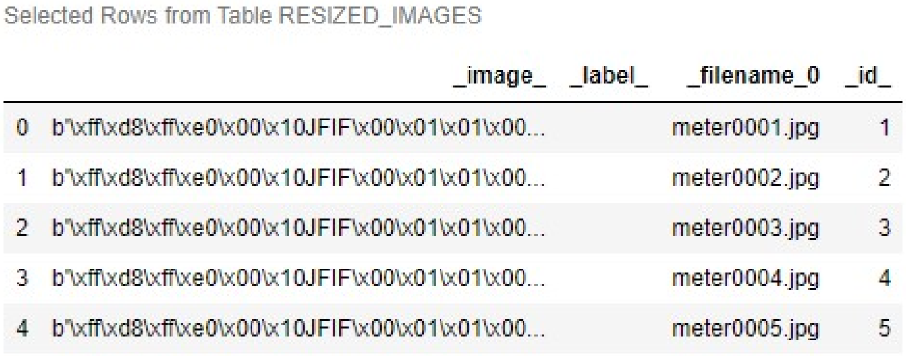


```
image height, width: (210, 924)

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.

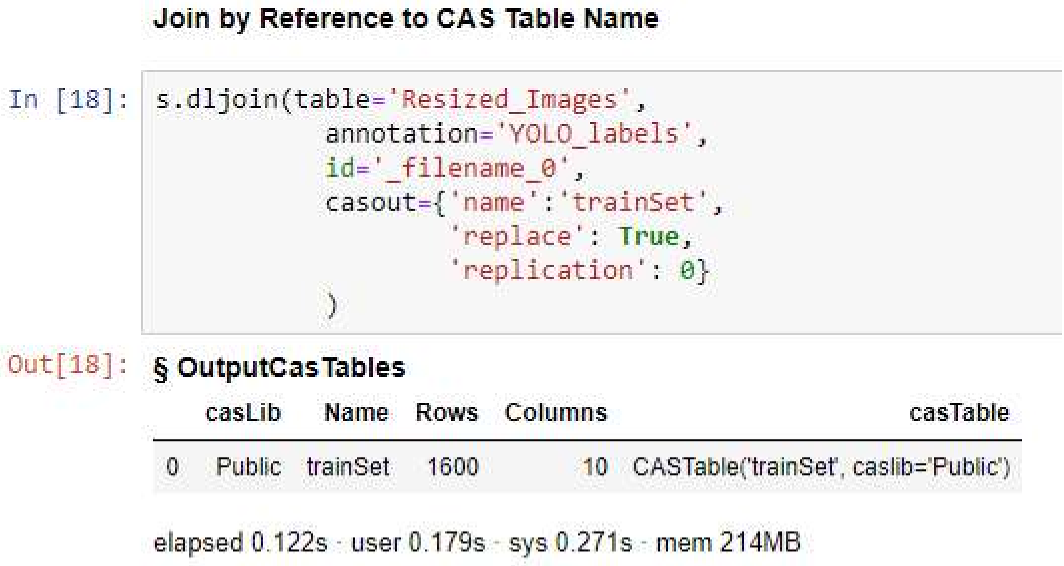


Selected Rows from Table RESIZED_IMAGES

	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:

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

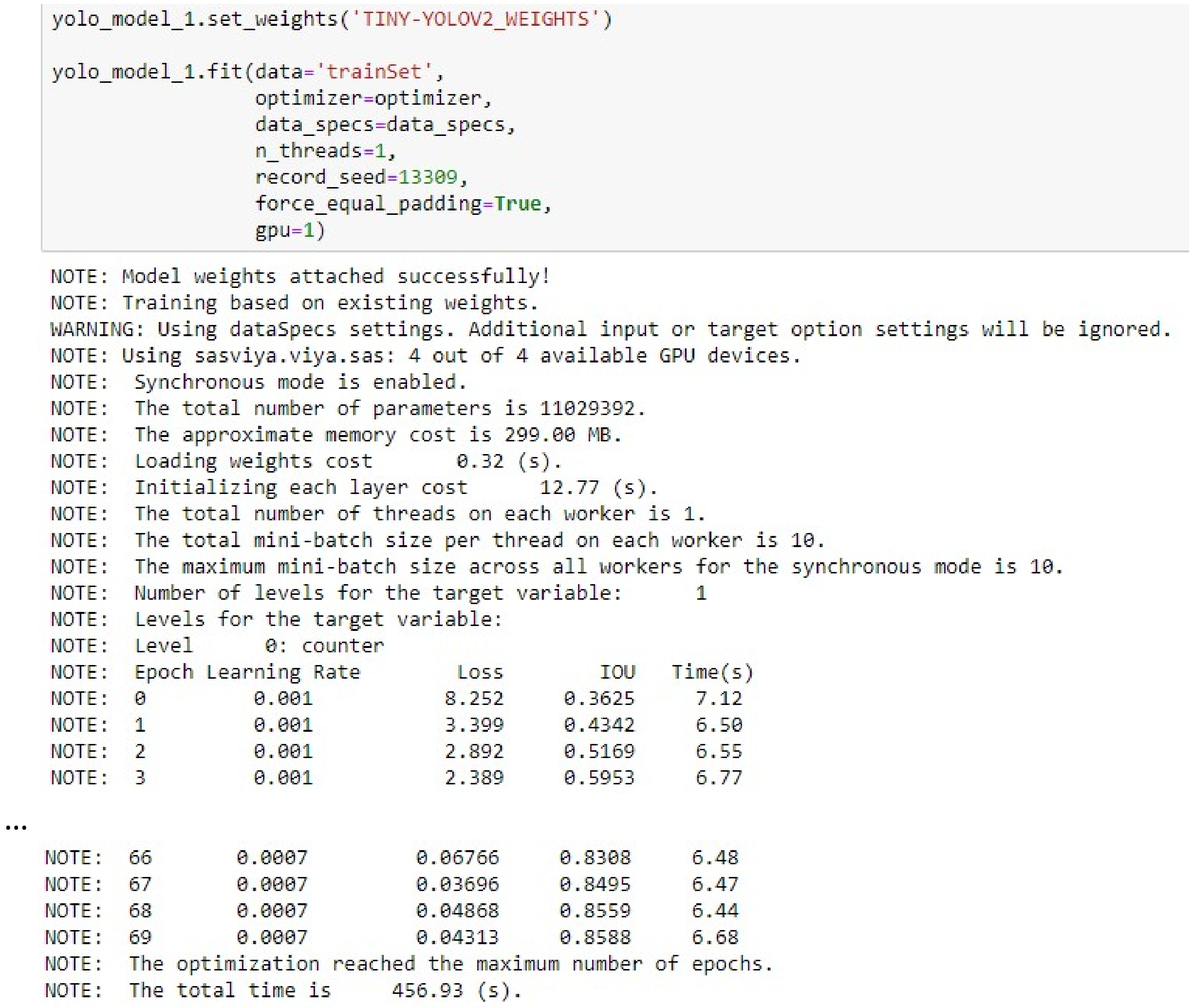
Where C = number of output classes (1), and A = number of anchor boxes (5) to produce 30 filters for each 13x13 cell that the image is split into.

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.

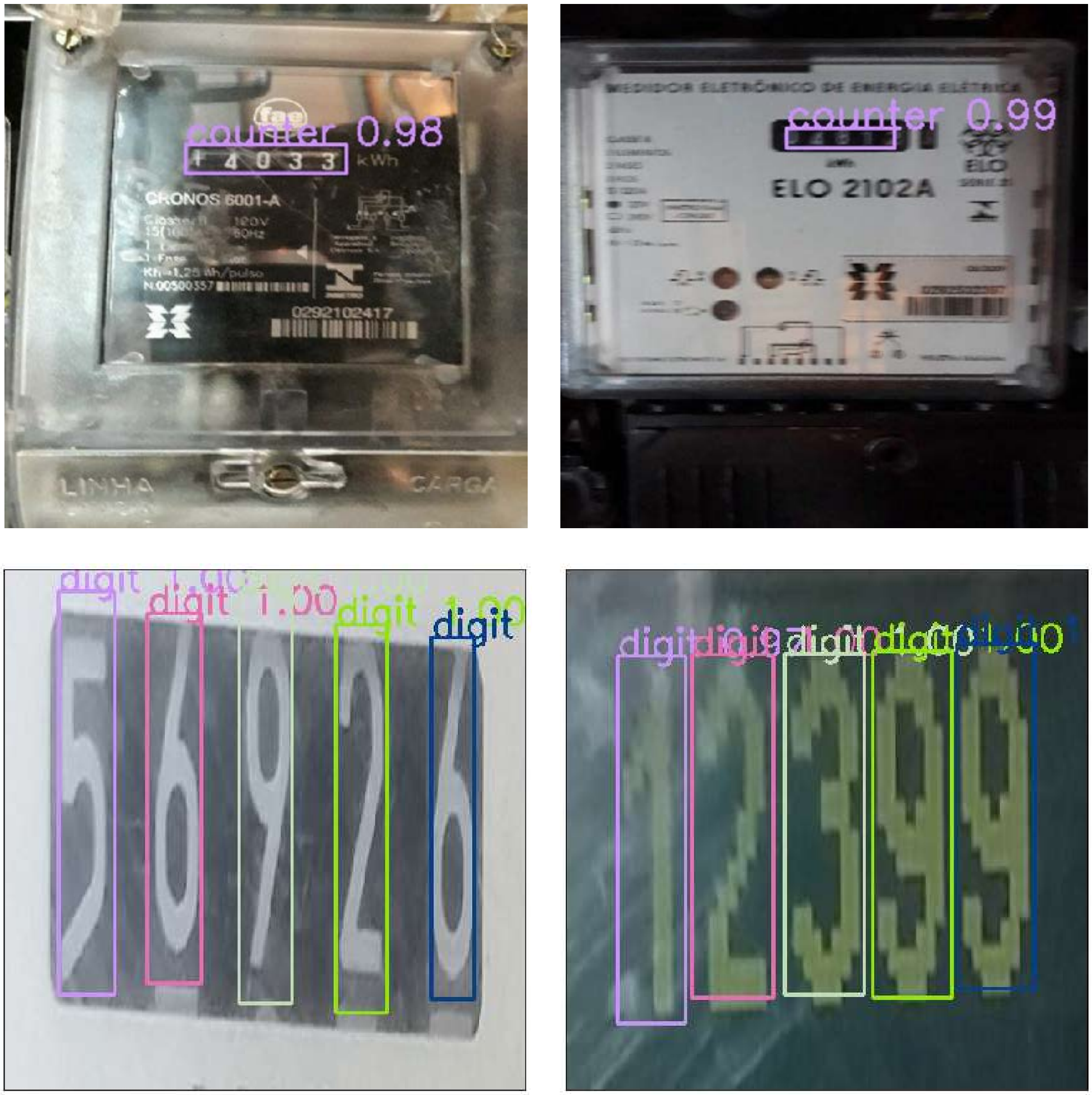


Figure 14. Verifying Tiny YOLO Model Outputs

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.

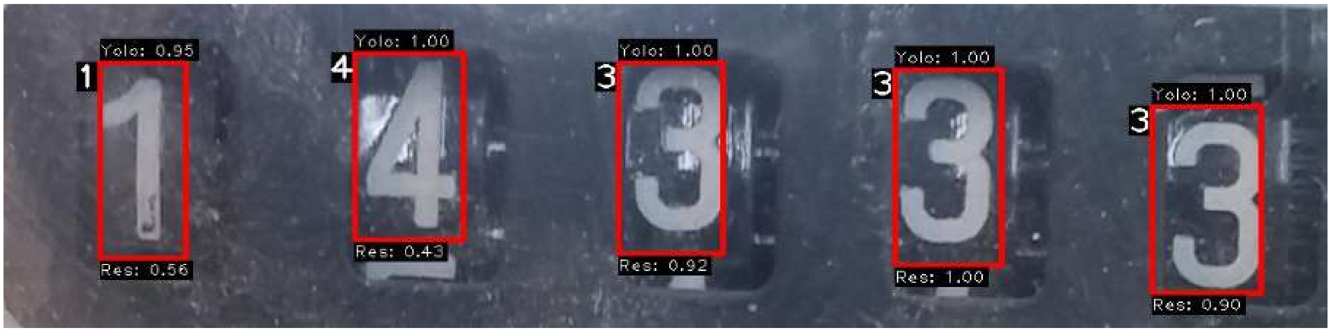


Figure 15. Digit Detection and Classification Confidences

[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.

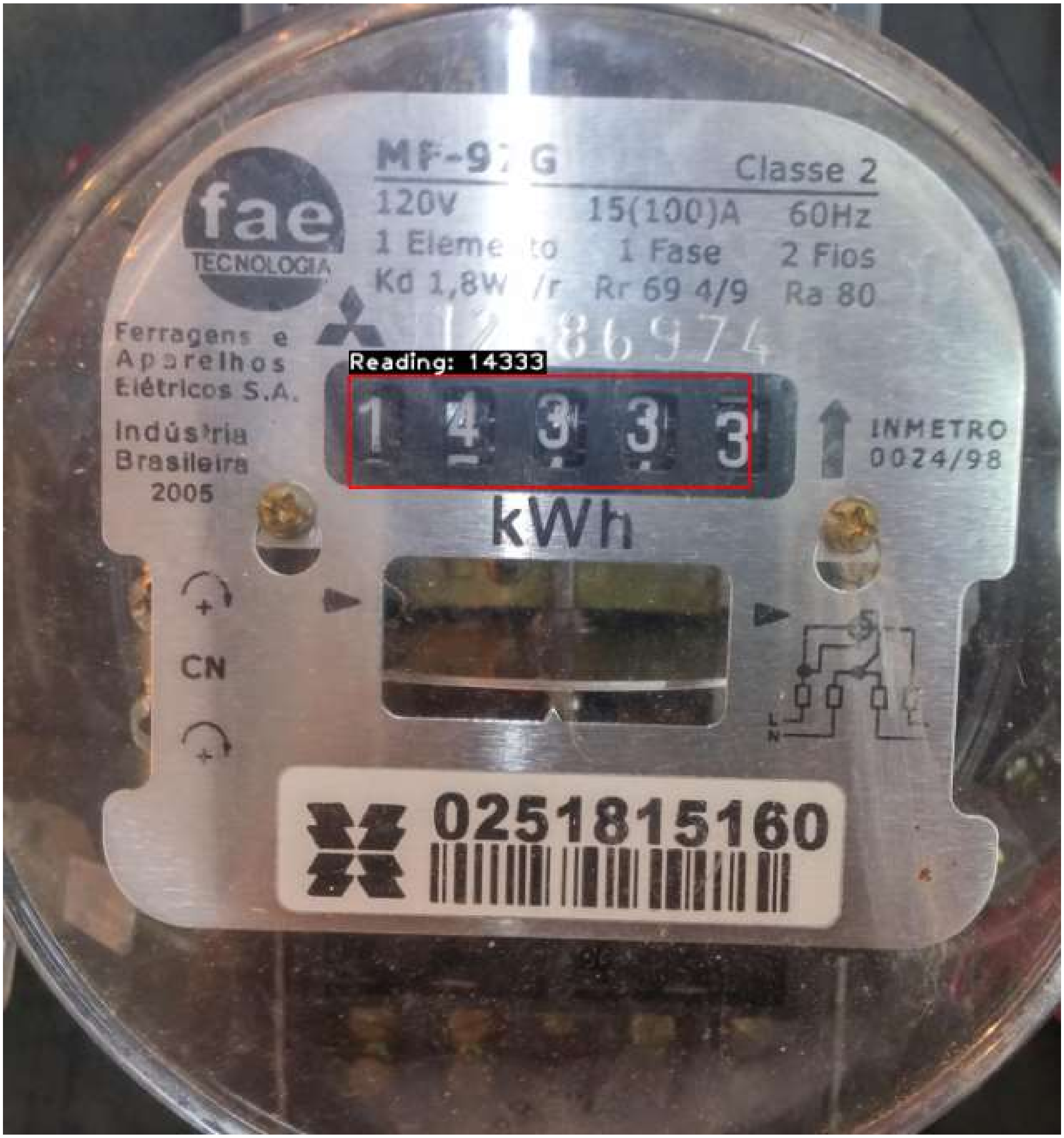


Figure 16. System Output Displayed on Input Image

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