SAS® GLOBAL FORUM 2021

Paper 1132-2021

Automated Vehicle Odometer Reading Using SAS® for AI & Computer Vision

Tyler Rosacker, FCAS, State Farm; Kedar Prabhudesai, Hardi Desai, Neela Vengateshwaran, and Varunraj Valsaraj, SAS Institute Inc.

ABSTRACT

Artificial intelligence (AI) is gaining increasing popularity with its ability to automate mundane, repetitive tasks that would otherwise require hours of manual work. In this paper we discuss one such application using SAS AI and computer vision to automatically extract vehicle odometer readings from dashboard images. Reading vehicle odometers can be useful for verifying how far the vehicle has driven, and better understanding the risk the vehicle entails from an insurability perspective. However, with potentially thousands of images received daily, it would not be practical for a human to conduct a review of each image. To overcome this, we have built a system using the You Only Look Once (YOLO) deep learning architecture, to automatically extract the odometer reading from a vehicle dashboard image. Our system comprises of two YOLO V2 models – one to detect and isolate the odometer region from the dashboard image and another to detect individual digits in the odometer region to construct a string of digits corresponding to the odometer reading. Our models are trained using a combination of open-source and closed-source datasets to build a robust system that can be used for automated odometer extraction from a wide variety of vehicle dashboard images. Our odometer detection model achieved an accuracy of 88.3% and the digit detection model achieved an overall error rate of 17.5%. This paper discusses the methodology behind this approach along with detection results.

INTRODUCTION

The automobile insurance industry is one of the biggest industries in the United States measured by market size and revenue and expected to continue to grow in the forthcoming years. The biggest boon to the industry is the availability of abundant data that insurance companies collect from customers to meet their expectations to deliver prompt services in the event of an accident. This presents a great opportunity for artificial intelligence systems to be used for automating mundane, repetitive tasks that would otherwise require hours of manual work.

Auto insurance companies determine the price of an insurance premium using rating variables – which are characteristics of policyholders used to estimate the cost of risks associated with the insurance. For example, a vehicle that covers more miles per year has a higher risk from an insurance perspective compared to one that covers fewer miles per year. Thus, one of the earliest rating variables used for predicting auto insurance losses was the mileage of the vehicle being insured. This variable has a ton of appeal as a rating variable due to its high predictive power and the intuitively causal relationship with insurance loss. It also has a very objective measurement in the form of the vehicle odometer.

While mileage may be a high impact rating variable for pricing auto insurance; the actual process of capturing of accurate mileage data can be difficult. Insurance companies have historically asked their policyholders to estimate the driving they will do, but, as you can imagine, those estimates are neither accurate nor granular. There is alternatively the option

to purchase this data from original manufacturers (OEMs) or from third party data vendors e.g. Carfax, AutoCheck etc. However, these options cover a limited portion of vehicles on the road, as well as have a monetary cost. In more recent years insurers have moved to using vehicle telematics devices to capture mileage as well as other vehicle behavior data. Even so, an insurer may have concerns about the quality or reliability of the data provided by these telematics devices.

One potentially attractive way an insurer could get accurate mileage data is to have policyholders provide their odometer readings through a smart phone app. Over time, these mileage reads can be used to calculate, or project forward the mileage the vehicle will drive. Having such a system in place, the insurance company would certainly have an interest in verifying that the odometer reading provided by the policyholder is accurate. One option for verification is to have the policyholder provide a photo of their dashboard, which can then be compared against the odometer reading provided by the policyholder for additional verification of the data provided. This photo verification process is tedious, since a large insurance company might have millions of photos that would need verification each month. In this work we demonstrate a system using SAS AI and computer vision which can be used to automate the verification of vehicle odometer photos. This system uses computer vision models to automatically extract a string of digits corresponding to the odometer reading which can then be compared with the odometer reading submitted by the policyholder to verify mileage data.

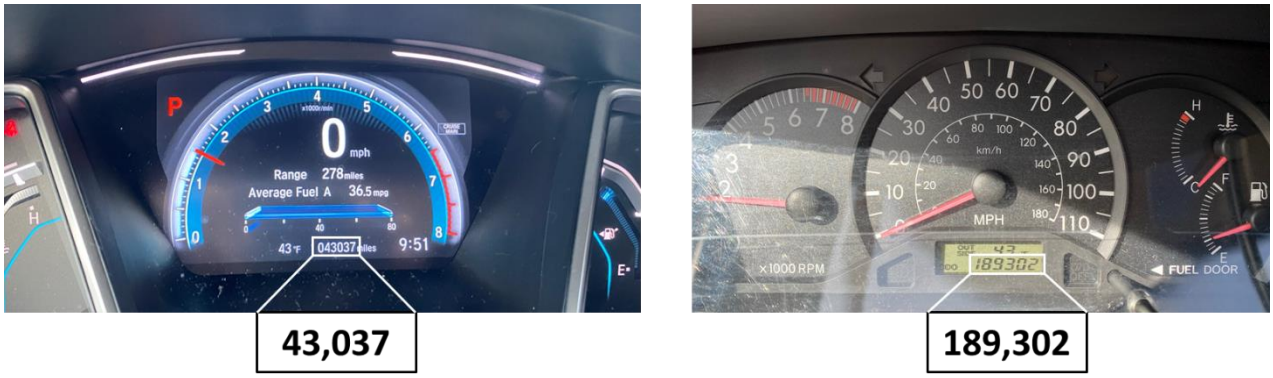


Figure 1. The goal of this project is to automatically extract odometer readings (shown inset) from dashboard images. The system comprises of two computer vision models – one to isolate the odometer region and another to detect digits.

METHODOLOGY

We have built a pipeline to automatically extract the odometer reading from photo of the dashboard as shown in Figure 1. As we observe the images captured contains various components of vehicle dashboard including but limited to speedometer, trip mileage and overall mileage. In this solution we have focused on building the solution to accurately read the overall mileage of the vehicle. The major steps involved in building this solution are illustrated in Figure 2. At its core are two computer vision models – one to detect the odometer from the dashboard image and another to recognize digits in the odometer region. For this project we used the "Image", "DeepLearn" and "aStore" action sets in SAS Cloud Analytics Services (CAS); the SAS® Deep Learning Python Interface (DLPy); and the OpenCV toolkit in Python. In this section we start with a description of the data followed by the details about the methodology we used to design the system.

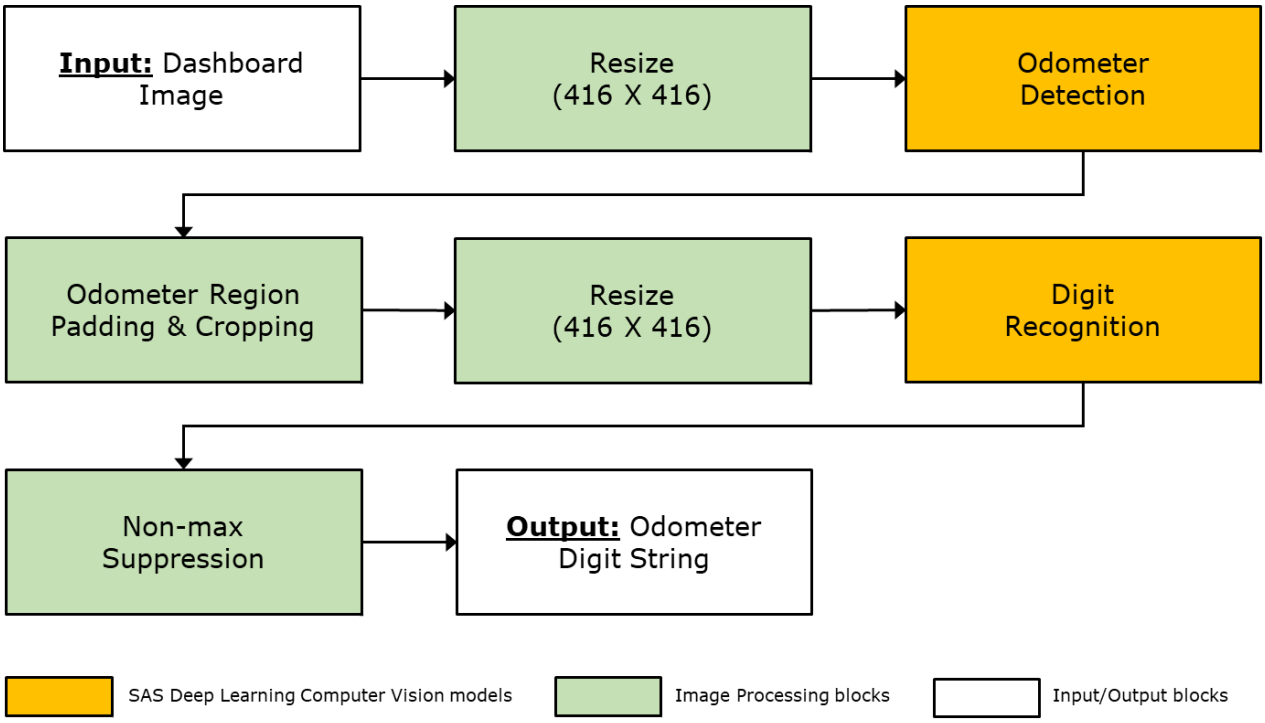


Figure 2. Flowchart showing the main components of the system. Orange blocks denote SAS Deep Learning Computer Vision models: Tiny-YOLO-V2 for odometer detection and YOLO-V2 for digit recognition.

DATA DESCRIPTION & CHALLENGES

The data for this research was provided by State Farm and included 5,000 dashboard photos used for model training as well as an additional 1,000 photos held out for verifying performance of the entire pipeline. These photos were generally of high quality; however, a limited subset of these images were blurry, dark, and not considered as valid dashboard photos. If the human labeler was able to confidently label the odometer reading from the dashboard images, we have included the image in our dataset. However about 0.1% of photos were removed where that was not possible.

The valid photos used for training the models had a high degree of variability. It included relatively recent vehicle model years with digital odometers as well as older vehicles with mechanical odometers. It should also be noted that with a huge variety vehicle make and model, we observed a huge variation the dashboard layout and design too.

The dashboard image was typically clicked on a smart phone by the policy holder on the State Farm app. This brought in a lot of variability with respect to how zoomed is the odometer reading, lighting conditions as well as the location of the odometer within an image. The odometer generally was located near the center of the photo, but the proportion of the photo that the odometer took up had a wide range between less than 0.1% of the photo, and upwards of 30%+ of the photo. This is of significance as many computer vision architectures struggle with this type of scale difference. As we modelled the solution for this use, we have accounted all the above described situations.

DATA PRE-PROCESSING AND LABELLING

The data provided for this project was not labeled as per the requirement for our proposed solution. To start this project, the 5,000 images were uploaded and manually labeled in an open-source labelling tool, Computer Vision Annotation Tool (CVAT). A sample annotated image is shown in Figure 3.

First, we labeled a tightly bound region around the odometer region. This label included metadata about if the odometer was "mechanical" or "digital" to help the modeling process pick up distinctions between these two very different looking types of odometers. Next, we labeled each digit found within the odometer as well as the decimal points included for numbers like "4,502.1". This again included a tight bounding box coordinates around the digit as well as label indicating the value of the digit (0-9). After the images were labeled, the metadata was output from CVAT, and read into CAS as an "ImageTable" using DLPy.

Figure 3. Example of the odometer and digit labeling in CVAT. Bounding boxes around the odometer region and individual digits were annotated along with additional metadata about the odometer such as type and orientation.

ODOMETER DETECTION MODEL

As discussed previously, the first modeling task for this system was to detect the location of the odometer within the vehicle dashboard. This was generally expected to be the more difficult task due to the range of repeated symbols on a dashboard. "How would the model learn what is an odometer versus a speedometer versus a trip odometer?" was an open question at the beginning of this research.

In order to make this model as broadly applicable as possible, the inputs to the model were restricted to only the single 416 x 416 pixel image. The data set had other potential predictors such as make, model, time of day, etc. but this research limited the scope to only using the image. This makes the model more applicable to real world tasks where that information may not be as readily available, as well as allowed us to use pre-built model architectures that only take image inputs.

To solve this task, a Tiny-YOLO-V2 model was selected. This model form was selected for a few reasons. The relatively simple architecture was expected to train faster, as well as have faster computations during inference. This faster inference time may be appealing in the specific application as the model may need to run on a smartphone. An extensive hyperparameter search was conducted across ~30 hyperparameters and over 2,500 models were ultimately built. The final model was trained for 150 epochs with an AdamSolver, and a polynomial learning rate scheduler (learning rate = 0.00075, power = 1.5).

The data for this task was relatively limited in the grand scheme of computer vision problems. Approximately 5,000 photos were part of the labelled dataset. These images were split 70/15/15 as Training/Validation/Test set respectively. With this size of data as described above, many make/model/model year combinations had only 1 or 2 photos. While the dashboard looks identical across different vehicles of the same type, we wanted the model to generalize to different lighting conditions, image orientations, and odometer sizes. To that end, the "augmentImages" and "processImages" CAS actions were used to create additional variability in the set of training images. This action made it possible to mirror pixels across the image to create images with different odometer sizes. An example of these augmented images is included in Figure 4. Use of these augmented photos was grid searched like other hyperparameters, and it was clear that the augmented data enhanced the model. However, as expected the training time of the model increased linearly with the number of augmented images. One noticeable benefit from this image augmentation was that model performance improved significantly for photos with extremely small odometers compared to the full image (i.e., odometers smaller than 0.5% of overall image pixels) while not hurting the performance for images with large odometers.

Figure 4. Example of augmenting a photo using pixel mirroring to make the odometer smaller. A large portion of the image is "fake".

As an additional way to handle the limited dashboard data available for this project, pretrained weights were used for warm starting the model training. These pre-trained weights were provided from a model that focused on a totally unrelated task of detecting odometers. That unrelated task was predicting the targets of the Open Images V4 dataset, which has images with a broad set of 600 classes from a variety of topics, none of which relate to vehicle odometers. Even though these pre-trained weights seemingly should have no applicability to the task at hand, the use of random weight initialization versus these pre-trained weights was grid searched like other hyperparameters, and it was clear that the pretrained weights enhanced the model.

DIGIT RECOGNITION MODEL

The second modeling task was to recognize digits from the odometer region to generate a string of digits corresponding to the odometer reading. This was achieved by training a separate digit recognition model based on the YOLO-V2 deep learning architecture.

Like the variability in the odometer types, there was a large variability in digit types as well. Some of the factors that contributed to the variability were: digit fonts, digit angles, digit types (digital vs. analog), background color, foreground color, lighting conditions etc.

To build a digit recognition that is robust enough to account for these variabilities, we used a combination of open-source and closed-source datasets to train the digit recognition model. To start with model training, we trained the YOLO-V2 model for 50 epochs with 50,000 images from the Street View House Numbers (SVHN) dataset, using the AdamSolver optimizer with learning rate of 0.0001 ("step" learning rate policy with step size = 10). The SVHN dataset consists of images of house numbers as shown in Figure 5. After training the model using images from the SVHN dataset, we froze the model weights and continued training the model with an additional 2,700 images provided by State Farm. Recall that dashboard images were manually annotated with bounding boxes around the odometer region and around individual digits. We extracted odometer crops from dashboard images and resized them to 416 X 416 to train the digit recognition model for an additional 25 epochs ("step" learning rate policy with step size = 5). A total of 11 object classes were used for training: 10 for digits 0-9 and one class for the decimal.

Figure 5. Sample images from the open-source Street View House Numbers (SVHN) dataset. We used 50,000 images from the SVHN dataset along with 2,700 images from State Farm to train the digit recognition model.

COMBINED PIPELINE

After training the odometer detection and digit recognition models, ASTORE files were generated for each model to deploy the models and score holdout dashboard images. Each holdout image was first resized to a size of 416 X 416 and the odometer detection model was run. Since YOLO-V2 model returns normalized bounding box co-ordinates, the output was used to crop odometer region from the original non-resized dashboard image. This allowed

us to get the best resolution of the odometer crop region. Using the raw outputs from the odometer detection model, we found that the bounding boxes had a tight margin around the odometer region, which tended to trim the first or the last digit of the odometer. Hence to ensure that all the digits in the odometer region would be accounted for, we padded a few pixels around the bounding box to extract a slightly bigger odometer crop, as shown in Figure 6.

This odometer crop was then resized to a size of 416 X 416 and scored using the digit recognition model. The digit recognition model provided bounding boxes around detected digits in the odometer region along with a confidence score associated with class of each detection. We observed that in many instances the digit recognition model produced multiple detections i.e. multiple bounding boxes, around a single digit as shown in Figure 7. We performed non-max suppression to solve this. First, we computed pairwise intersection-over-unions (IoU) for each bounding box pair and used a threshold of 0.2 to detect overlapping bounding boxes. Next, for overlapping bounding boxes, we selected the box with a higher confidence score and suppressed the box with the lower confidence score. Finally, we sorted the bounding boxes based on the x co-ordinates to build a string of digits corresponding to the odometer reading.

Figure 6. (A) Sample odometer crops based on bounding box output of the odometer detection model. The model tended to produce a tight margin around the odometer region which resulted in some digit getting trimmed. (B) We padded a few pixels along both row and column dimensions to ensure that all digits were accounted for.

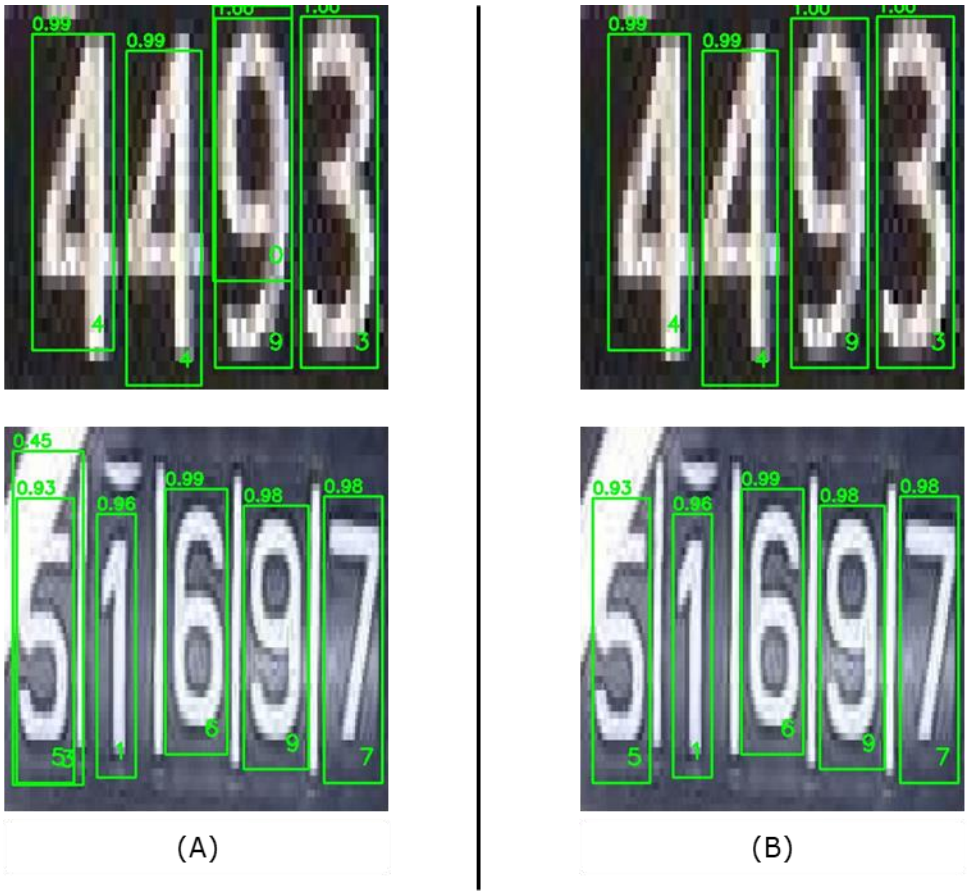


Figure 7. (A) Sample images of odometer crops overlayed with bounding boxes around digits from the digit recognition model. Notice that some digits have multiple predictions: digit nine has predictions 0, 9 (top left), digit five has predictions: 5, 3 (bottom left). (B) Output of respective images after performing non-max suppression, resulting in a single detection for each digit.

RESULTS

In this section, we discuss the performance of the odometer detection model and the digit recognition model separately and finally the performance of the entire system putting both the models together.

ODOMETER DETECTION MODEL

We evaluated the performance of the odometer detection model based upon a 40% pairwise-intersection over union (IoU) threshold between the detected odometer and the ground truth odometer bounding box. At that threshold, the precision was 88.3%, the false-positive rate was 13.3%. Included below is the precision of the final model across different IoU thresholds.

This relatively low threshold was selected because this model did not need to have a perfect bounding box around the odometer. Instead, it was important that the full odometer was included in the final image so that the Digit detection model did not have to deal with cropped digits. This bounding region was further padded as described in the "Combined Pipeline" section above which would tend to increase the union area of the bounding boxes further.

While difficult to quantify, a visual inspection of model errors was conducted. In general, most images were satisfactory with about 90% of images appearing to be good detections of the odometer.

For those images with poor odometer detections, a common cause of model confusion was tightly packed characters near each other. Figure 10 shows one such example where the model was distracted by the word "standard" seemingly due to the tightly packed characters.

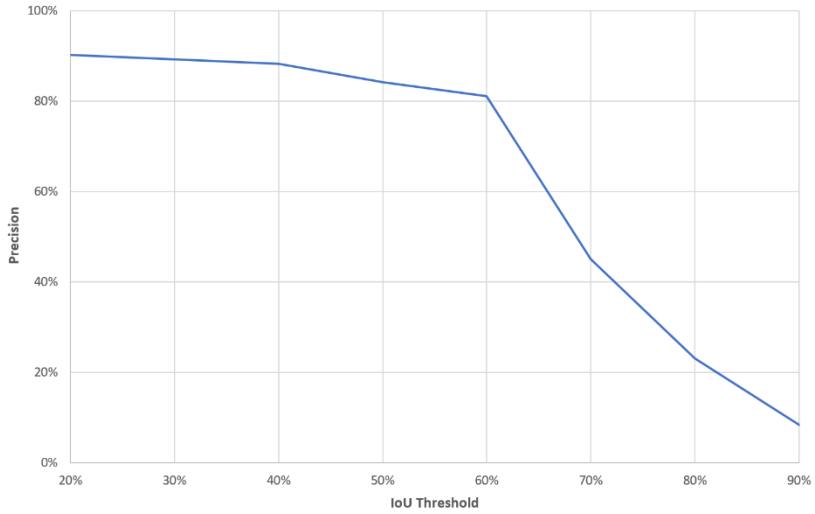


Figure 8. Precision of the Odometer Detection Model across various IoU thresholds. An IoU threshold of 40% was used for hyperparameter tuning.

Figure 9. Example of a true positive detections for the odometer detection model. The numeric value above each box shows confidence level in the detection.

Figure 10. Two examples of false positive detections for the odometer detection model. The numeric value above each box shows confidence level in the detection. In the first example, the model was more confident in the word "standard" than the actual odometer, and in the second example the model detected two odometers.

The Tiny-YOLO-V2 architecture is known for struggling with detecting objects with extremely small scales. This fact is due to the low-resolution grid (13 X 13) used for object detection. A potential enhancement on this model would be to investigate using other architectures such as the YOLO-V3 model or even two stage models like Fast RCNN. These models are known to handle small object detection better, especially if high resolution images are provided.

DIGIT RECOGNITION MODEL

We evaluated the performance of the digit recognition model using the digit error rate (DER) metric, which is analogous to the word error rate metric used in automatic speech recognition. The DER accounts for three types of errors that can occur when detecting individual digits, shown in Figure 11:

(A) Deletion type error: When the model fails to detect a digit altogether

(B) Insertion type error: When the model inserts a digit at a location where a digit does not exist

(C) Substitution type error: When the model substitutes a digit with an incorrect digit

The number of instances of each of the above errors are determined by comparing the true odometer digit string with detected digit string using a dynamic programming approach, and the sum of these instances are normalized by the length of the true odometer digit string to calculate the DER:

$$Digit\ Error\ Rate\ (DER) = 100 \times \left[\frac{N_{dels} + N_{ins} + N_{subs}}{N_{total}}\right]$$

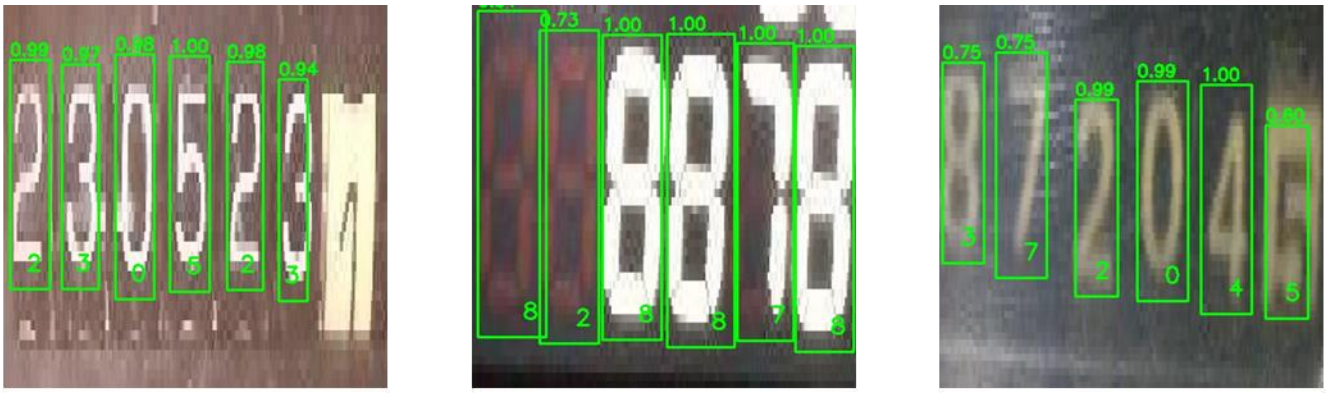
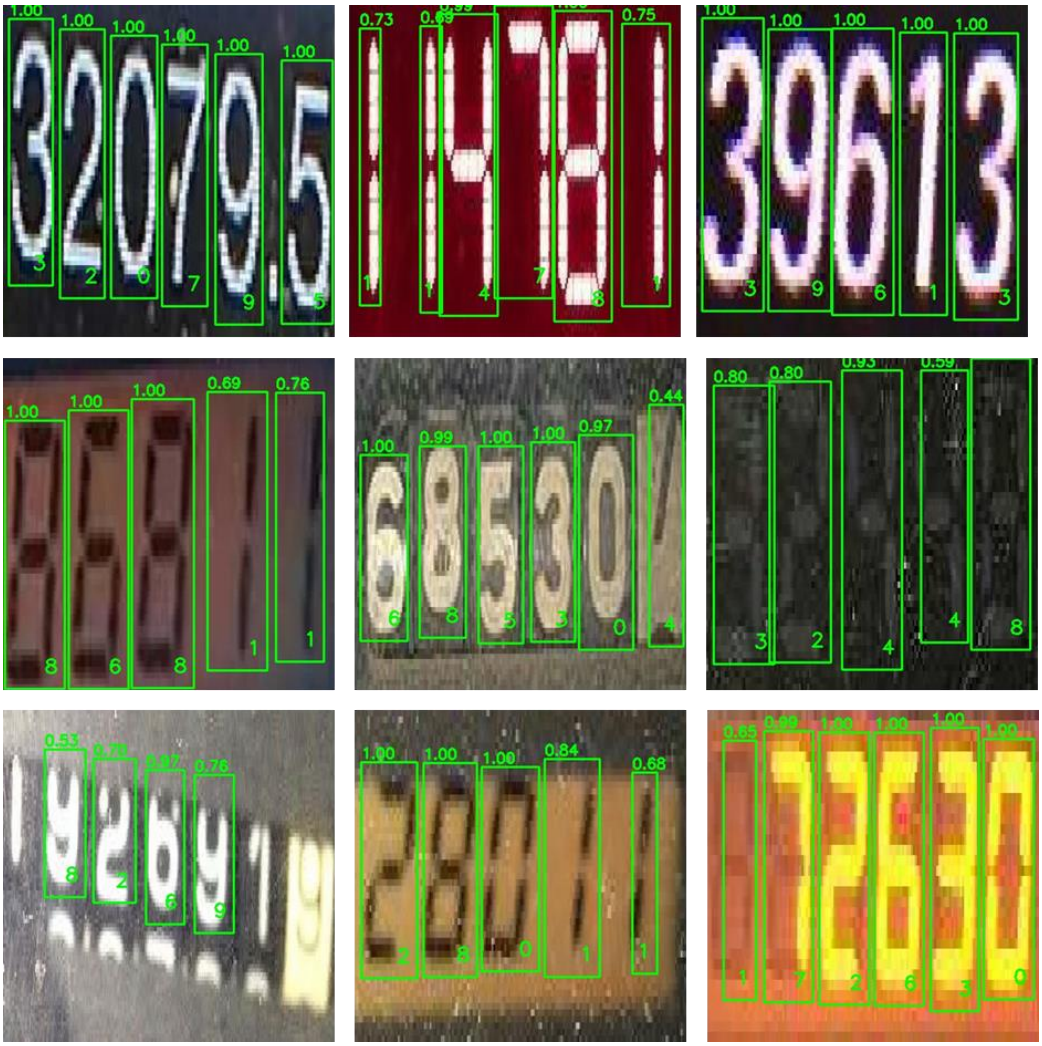


Figure 11. Examples of images with detection bounding boxes demonstrating the three types of errors used for calculating digit error rate (DER). (A) Deletion type error: digit one is missed altogether by the model, (B) Insertion type error: model has inserted digits eight, two erroneously, (C) Substitution type error: model has incorrectly detected the digit eight as three.

We evaluated the results of the digit recognition model on a holdout set of 1,243 images. Note that for these images, we did not run the odometer detection model but extracted the odometer crops from the annotations generated during the manual labeling task. Figure 12 shows some sample images superimposed with the detections generated from the digit recognition model. Note the large variability in the digit characteristics as discussed before.

We calculated the DER by comparing the output of the digit recognition model with the true odometer readings provided by human labelers. A histogram of the DER over 1,243 holdout images is provided in Figure 13. Since we are calculating the error rate, a lower value indicates better performance. We notice that a bulk of the mass of the histogram is concentrated on the DER value of zero percent. A total of 1,022 (82.2%) images had a DER of zero percent, that is they had a perfect prediction, whereas a total of 1,198 (96.3%) images had a DER less than forty percent. Based on these results, we can conclude that despite the variability in odometer digits, training the digit recognition model with help of SVHN and State Farm images, helped the model to generalize and detect digits accurately in most of the images.



320795	114781	39613
86811	685304	32448
_8269_ _	28011	172630

Figure 12. Examples of images with digit detection bounding boxes superimposed. Images are arranged in a 3X3 grid with respective odometer readings generated

using the digit recognition model shown below. Digits detected correctly are shown in green whereas incorrect or missed detections are shown in red. Note the variability in the digit types, digit fonts, digit angles, lighting conditions etc.

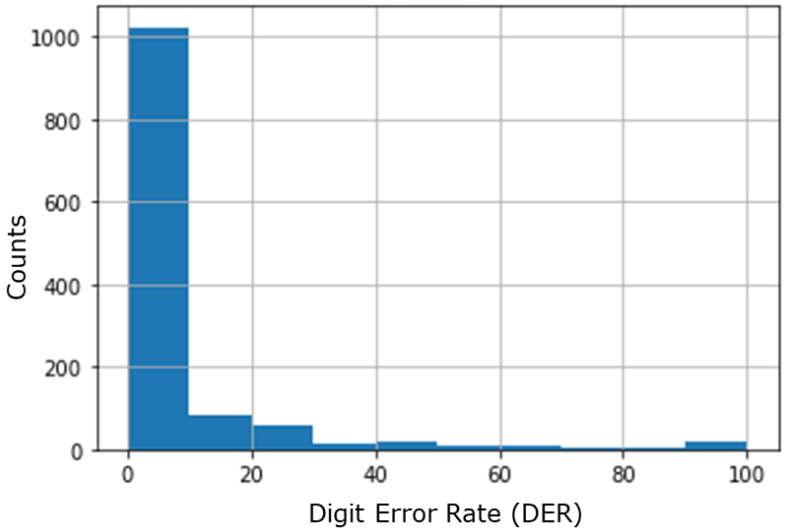


Figure 13. Histogram of Digit Error Rate (DER) over 1,243 holdout images demonstrating performance of the digit recognition model. We notice that the bulk of the distribution is concentrated on the value of zero percent DER.

COMBINED PIPELINE

We used a separate holdout set of 989 images to test the performance of the combined pipeline i.e. both the odometer detection and digit recognition models. This holdout set was separate and untouched during training or validation for both the odometer detection and digit recognition models. Histogram of DER over these 989 images is shown in Figure 14. A total of 648 (65.5%) images had a DER of zero percent indicating perfect prediction. Note that a perfect prediction indicates that both the odometer detection model accurately detected where the odometer is, and the digit recognition model accurately predicted all the digits. A total of 833 (84.2%) images had a DER of less than forty percent. A DER of hundred percent indicates that either the odometer detection model did not produce an accurate odometer crop, or the digit recognition model did not predict a single digit in the odometer accurately. A total of 95 (9.6%) images had a DER of hundred percent.

We also calculated the overall digit error rate by aggregating the errors over the entire set of 989 images. A total of 5,492 digits (N_{total}) were present in the set of images and the individual errors were as follows:

- Deletion type errors: 600 (N_{dels})
- Insertion type errors: 115 (N_{ins})
- Substitution type errors: 248 (N_{subs})

$$Overall\ DER = 100 \times \left[\frac{600 + 115 + 248}{5{,}492}\right] = 17.53\%$$

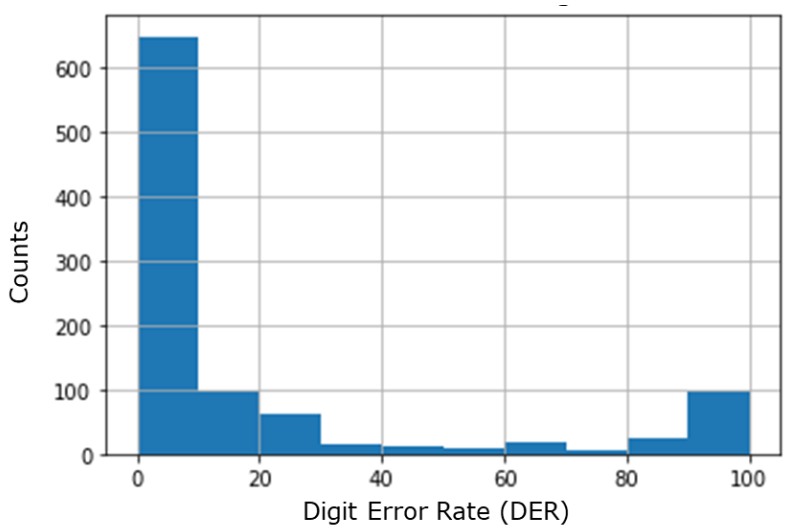


Figure 14. Histogram of Digit Error Rate (DER) over 989 holdout images demonstrating performance of the combined pipeline i.e. using both the odometer detection model and the digit recognition model. Note that these results were evaluated on a set of holdout images different from the one shown in Figure 13.

FUTURE WORK

For future work, we have a wide set of potential enhancements that may be valuable to research to improve overall system performance. While these areas of research were not deeply explored as part of this project, we think these are fairly logical extensions of this research, and good things to look into before someone might try to recreate this type of system in their production environment.

One difficulty with this research was the quality of the input images. Many images were overly blurry, dark, or taken from a weird angle/rotation. While ideally a computer vision model would generalize to these scenarios, there is a physical limit to how bad a photo can be before a model can't reasonably be expected to interpret the photo. To that end, we would like to research ways to pre-process and filter out poor quality photos before they are input into the system. This could involve an additional model run before the rest of the system that predicts if the model is of high enough quality, or a simpler approach would be to use heuristics like average light level or estimating the sharpness of the image using rise distance. The end result would be that a user of this system could go back to the person who took an image and say "could you please try again", and hopefully the average quality of images fed into the model in production would improve.

Another issue during this model build was the comparatively limited set of images (~5,000). With computer vision tasks there is a wide set of image augmentation methods that could be used, and those were briefly researched as part of this project. Additional work could be done to further this image augmentation. Of particular interest would be additional time spent trying to tweak the perspective of photos such as modifying photos to look like they were taken from more off-center angles.

Next, we liked to research how applications of multitask learning could help with allowing the odometer detection model to generalize. The current odometer detection model only knows

to detect 3 things ("digital odometer", "mechanical odometer", "empty space"). This type of detection can lead to many false positives where other parts of a vehicle dashboard will look more like an odometer than they do like empty space such as a trip odometer or a speedometer. Potentially expanding the task of this odometer detection model to have it also detect other objects on the dashboard such as a speedometer could help it generalize better to have fewer false positives on the actual task of interest.

A final item to research would be applications of meta-modeling. When the odometer detection model is run it produces potentially multiple odometer detections as well as confidence levels for each detection. Likewise, the digit detection model produces similar metrics. A promising idea for future research would be to build a model that sits at the end of the system, takes these various metrics ("how confident about the predictions are you", "how many detections were made", "for each odometer detected, how many digits would be detected for that specific odometer"), and feed them in as predictors. This model would then output an overall confidence level of the model building pipeline and could be used in production to flag potentially misleading results for human review. It could also be used in model development to debug the types of images that have poor performance currently.

CONCLUSION

The goal of this project was to demonstrate that SAS AI and computer vision can be used to automate mundane and repetitive tasks that would otherwise need hours of manual work. In this paper we described a system to automatically extract odometer readings from vehicle dashboard images using two deep learning computer vision models – one to isolate and localize the odometer region and one for digit recognition. From the results from the combined pipeline section we see that from 65.5% of images had a perfect prediction and would require no manual labor to validate the odometer readings. If we assume that the system is prone to errors with respect to a few digits, 84.2% of images had a DER of less than forty percent. Thus, manual labor would be required for the rest of the images (15.8%) where DER is greater than forty percent.

REFERENCES

Actuarial Standards Board of the American Academy of Actuaries,"Actuarial Standard of Practice No. 12, Risk Classification (for AllPractice Areas)," revised in 2005, updated for deviation language in 2011.

Karapiperis, Dimitri; Birnbaum, Birny; Brandenberg, Aaron; Castagna, Sandra; Greenberg, Allen; Harbage, Robin; and Obersteadt, Anne, Usage-Based Insurance and Vehicle Telematics: Insurance Market and Regulatory Implications, National Association of Insurance Commissioners, CIPR Study, March 2015, pp. 1-16 (excluding section on Tower Watson's DriveAbility) and 42-60.

J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," arXiv preprint, 2017. 3, 4

Nguyen, N., Do, T., Ngo, T. D., & Le, D, "An Evaluation of Deep Learning Methods for Small Object Detection". Annual Review of Anthropology, 2020, ["3189691"]

SAS Deep Learning Python Interface. Version 1.2.1-dev
https://sassoftware.github.io/python-dlpy/getting-started.html.

ACKNOWLEDGMENTS

We would like to thank members of the SAS Center of Excellence (CoE) team and the SAS pre-sales team who helped us with labeling odometers and digits which were used for modeling. We would also like to thank Conor Hagan for setting up the AWS environment and Mike James for managing the State Farm Student Exchange Program.

CONTACT INFORMATION

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

Tyler Rosacker
State Farm Mutual Insurance Company
tyler.rosacker.fhn6@statefarm.com

Kedar Prabhudesai
SAS Institute, Inc.
Kedar.Prabhudesai@sas.com

Hardi Desai
SAS Institute, Inc.
Hardi.Desai@sas.com

Neela Vengateshwaran
SAS Institute, Inc.
Neela.Vengateshwaran@sas.com

Varunraj Valsaraj
SAS Institute, Inc.
Varunraj.Valsaraj@sas.com