Pantograph Pose Estimation

Jesse Decker
1355 West 26th Street
Erie, PA, 16508
jessedecker@protonmail.com

ABSTRACT

Pantographs provide an electrical conduit from an overhead electric line (OEL) to an electrical motor or battery on vehicles such as locomotives, buses, and trams [1]. They work by dynamically adjusting their position to maintain contact with the OEL and have been used in this manner since electrification in the late nineteenth century.

In this paper, I describe a process for predicting the position of eighteen keypoints on a modern electric railroad pantograph using the Mask R-CNN framework as developed by *He et al*. [2]. Pantographs are an ideal candidate for pose estimation because the components of the pantograph are occluded, or hidden from view, as the pantograph goes through its range of motion. This occlusion makes an exact measurement of the complete object impossible.

Mask R-CNN employs the Microsoft Common Objects in Context (COCO) dataset for object detection, instance segmentation, and person keypoint detection. The framework scored at the top of the results for COCO 2016 in all three tracks of the COCO suite of challenges[3]. Because pantographs are not represented in the COCO dataset, I extend the Mask R-CNN framework to work with a custom dataset.

I trained the network on three hundred images labeled with three classes to represent the points of contact between the pantograph and the OEL. Each class is labeled using bounding boxes, segmentation masks, and a skeleton of six keypoints per class. Pixel distance (PD), or the straight-line distance as measured from labeled keypoint position to predicted keypoint position, was used as a performance metric to evaluate the network. The trained network was able to achieve a mean pixel distance of 16.7 pixels for successful classifications using a validation dataset set of thirty images consisting of 90 detections with 534 labeled keypoints.

Keywords

Mask-RCNN, Pose Estimation, Pantograph, COCO

1. INTRODUCTION

A pantograph is a hinged framework attached to the roof of an electric locomotive used to provide a reliable electrical circuit from overhead electric lines to the electric motor on board the locomotive. The primary components of a pantograph are the panhead that makes contact with the electric wires and two folding mechanical arms that raise and lower the panhead. A lifting control uses inputs from several sensors to operate the arms and maintain contact between the panhead and the OEL.

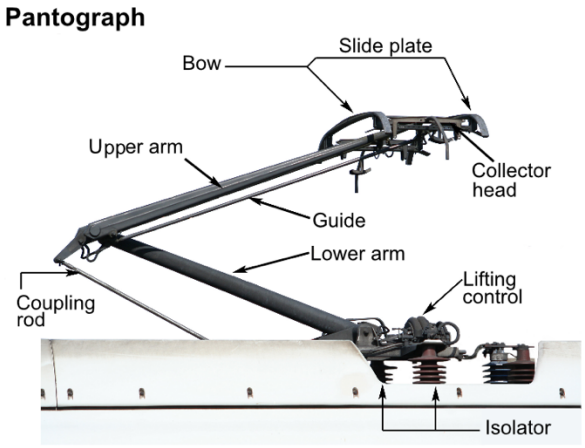

Figure 1: Pantograph diagram

The panhead consists of three components: a front and rear slide plate and a collector head in the middle. The components are fixed and do not move relative to one another.

The panhead is moved vertically to account for changes in distance between the locomotive and the overhead electric wires as it travels down the railway. The panhead can also be moved forward and backward in relation to the locomotive when the lifting mechanism articulates the mechanical arms.

During operation, the panhead can tilt left to right or front to back due to downward pressure from the OEL, vibration, wind, or other misalignments. If the lifting control detects a signal outside of the standard operating threshold, it will collapse the pantograph immediately.

Unfortunately, there are a host of potential hazards in a system which requires a direct physical connection to operate. Disengagement occurs when the panhead loses contact with the OEL. Entanglement occurs when the two become intertwined. Entanglement can be costly in terms of both cost to repair the equipment and for the loss in service time of the impacted locomotive and rail line. A typical contact pattern between the panhead and OEL is visible in Figure 2.

Figure 2: Pantograph sample image

Repair costs are a growing concern as many locomotive manufacturers shift their production from diesel over to hybrid and all-electric engines in a push for efficiency. Newer locomotives also operate at higher speeds, increasing concerns further.

Predicting failure states is a complex problem outside the scope of this paper, but we can better understand the conditions that lead to failure by analyzing how the pantograph moves in space. Pose estimation allows us to do just that by describing the position of several points of interest along the panhead on an X-Y plane. Further, pose estimation provides the ability to output these points even when parts of the panhead are not visible within the frame of the camera.

This data output by the network, a series of keypoints, could be used as input to various computer vision algorithms by railroad engineers if the location of the predicted keypoints is accurate and reliable. To evaluate the ability of the Mask R-CNN network to learn the movement of the pantograph and predict keypoints sufficient for this purpose, I built a framework for labeling images and constructing a dataset, trained the Mask R-CNN network on that dataset, and performed pose estimation analysis using PD as an evaluation metric.

2. RELEVANT WORK

Mask R-CNN is a deep neural-network-based framework for object detection and image segmentation. It extends the object detection framework Faster R-CNN by adding a layer for predicting a segmentation mask from each detection instance [4].

2.1 Faster R-CNN

Faster R-CNN utilizes a two-stage execution to make detections. The first stage, a Region Proposal Network (RPN), is responsible for proposing candidate Region of Interest (RoI) boxes. The second stage utilizes RoI pooling to extract features from these candidate RoIs for classification and regression (Figure 3).

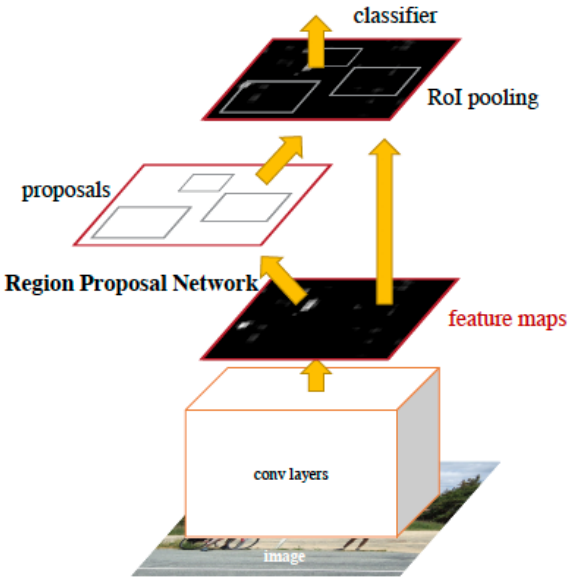

Figure 3: Faster R-CNN[4]

The RPN works by generating a set of rectangular object proposals, each with a regression score, from an input image. A small convolutional is slid across the image at set points, outputting a feature map. The center of each sliding point is called an anchor.

These anchors, along with features data, are used by a classifier to make a series of region predictions to determine the class of object within the region and regression or confidence score. Each region is evaluated by the output score, with the highest scoring classification being selected for prediction.

RoI pooling allows for accurate classification but does not ensure the pixel level alignment between the feature map and RoIs required for image segmentation. For that, we turn to Mask R-CNN.

2.2 Mask R-CNN

Mask R-CNN uses a similar two-stage architecture for detection and segmentation, with the first stage being an RPN similar to the one used in Faster R-CNN. The second stage employs a process called RoIAlign to ensure proper mapping between the extracted features and the input. The significant step forward Mask R-CNN makes over Faster-R-CNN is the addition of a layer parallel to the classification layer that is used to predict a segmentation mask within the RoI (figure 4).

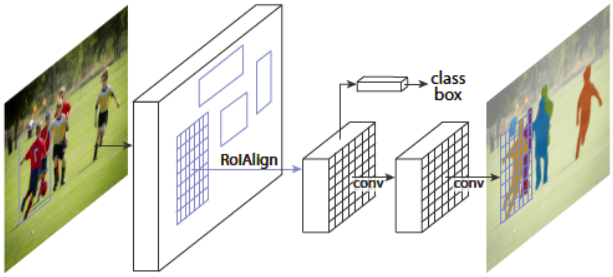

Figure 4: Mask R-CNN [2]

2.2.1 Network Architecture

The network consists of two architectures: the convolutional *backbone* used for feature extraction over an entire image, and the network *head* generating segmentation masks within an RoI.

The backbone relies on a combined ResNet and Feature Pyramid Network (FPN) first stage. The 101 layer ResNet employs five convolutional stages with strides of size 4, 8, 16, 32, and 64. The FPN uses a top-down architecture to build an in-network feature pyramid from a single-scale input.

The head is a set of custom Keras layers that take the RoI as input and outputs a single binary mask for the RoI.

Using a ResNet-FPN backbone for feature extraction with Mask R-CNN provides improved performance with regard to both accuracy and speed in both classification and regression over the previous Fast R-CNN network.

2.2.2 Segmentation Mask

The network adds a branch for predicting a segmentation mask for each RoI in parallel to the classification of the object and bounding box detection regression. The branch is a fully connected layer allowing for pixel-level mapping due to the addition of RoIAlign.

Because predicting the segmentation mask is performed separately from classification, Mask R-CNN does not use the mask to perform classification. This speeds things up to other segmentation models that rely on classification from the mask. Figure 5 shows both the bounding box and the segmentation mask drawn for each detection instance.

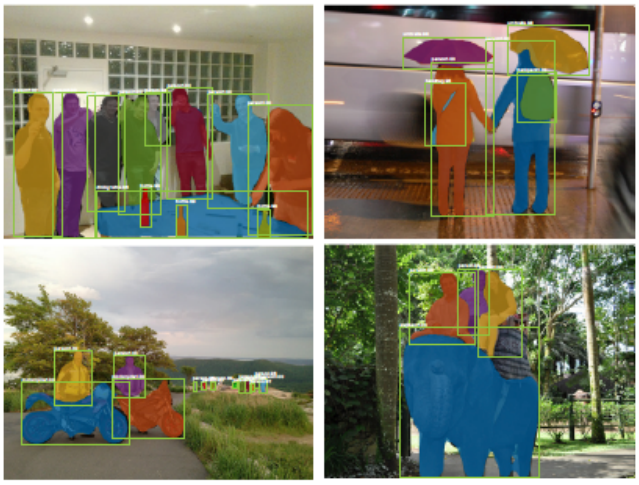

Figure 5: Mask R-CNN classification and detection[2]

2.2.3 Human Pose Estimation

Mask R-CNN extends the functionality of the pixel-level segmentation mask by modeling a keypoint's location as a one-hot mask. Contrary to the segmentation mask itself in which any point can be classified as a foreground class or background object, only a single pixel within the keypoint mask can be classified as a foreground object. The network can predict N number of one-pixel masks within the larger, single segmentation mask as a keypoint for any number of keypoints.

Keypoints are typically used to model human joints such as knee, elbow, and ankle. The network defines a skeleton for visualization as a list of start and end keypoints, i.e., the ankle is connected to the knee. Each human, or person COCO object, is represented by a series of 17 keypoints (Figure 6). The network does not contain any actual domain understanding of the human body, however, and because of this, it can be extended to work with non-person COCO objects.

Figure 6: Mask R-CNN Human Pose Estimation [2]

The keypoint head layers are slightly different from the segmentation mask layer it is based on, utilizing a stack of eight 3x3 512-d convolutional layers, followed by a deconvolving layer to produce a keypoint mask resolution of 56 by 56. This is the same resolution used to store the entire mask. The increased resolution significantly improves keypoint location accuracy.

The network was trained for pose estimation, all 135K images annotated with keypoints in the COCO trainval dataset were used. Images were resized to 800 pixels on the shortest side, and the network was trained for 90K epochs.

	AP^{kp}	AP^{kp}_{50}	AP^{kp}_{75}	AP^{kp}_{M}	AP^{kp}_{L}
CMU-Pose+++ [6]	61.8	84.9	67.5	57.1	68.2
G-RMI [31]†	62.4	84.0	68.5	**59.1**	68.1
Mask R-CNN, keypoint-only	62.7	87.0	68.4	57.4	71.1
Mask R-CNN, keypoint & mask	**63.1**	**87.3**	**68.7**	57.8	**71.4**

Figure 7: Keypoint comparison among top pose estimation models.[2]

Mask R-CNN has shown outstanding results, outperforming other segmentation networks capable of performing pose estimation (figure 7).

3. PROPOSED SOLUTION

My proposed solution for pantograph pose estimation is based on the human pose estimation process as implemented in the Mask R-CNN paper. I developed and iterated through three independent processes for creating a labeled dataset, training the network on the labeled dataset, and evaluating the network's performance using keypoint PD.

3.1 Dataset Creation

The labeled dataset was created using a three-step process of capturing an image from source video, labeling it for image segmentation, and generating annotation data from the labeled image.

3.1.1 Image Selection

All images used to train or validate the network were selected from two video files recorded on a camera located inside an operating locomotive. It is mounted in a fixed position in the front of a locomotive. The camera is directed toward the pantograph, and the camera angle remains unchanged throughout the videos. As the pantograph travels through its normal range of motion, the panhead moves up and down within the frame of the camera.

The video resolution is 1920x1080 pixels and is recorded in the RGB color mode. Each video is approximately 90 minutes long and is recorded at 60 frames-per-second totaling nearly 650,000 frames.

Each video was recorded in natural sunlight, with no other illumination provided. Image quality is, subjectively, quite good. Three issues that appeared intermittently within the dataset that could impact the performance of the network include low light conditions, background interference with catenary structures, and instances where the panhead is partially out of frame.

I first developed an algorithm to save frames of video as JPEG images. An equal number of randomly selected frames from each of the source videos are saved into train and validation folders at full resolution. The video name and frame number were combined to form a unique file name.

3.1.2 Image Labeling

Because neither the panhead nor its components are defined in the COCO dataset, I defined three classes to represent the components of the panhead: front bar, middle bar, and rear bar. Due to the vertical movement of the panhead, one bar can block, or occlude, the view of another bar with respect to the camera.

Occlusion occurs regularly in the dataset with the front bar occluding the view of some part of the middle bar and the middle bar occluding the view of some part of the rear bar. This is visible in figure 8, with the front bar in red, the middle bar in green, and the rear bar in blue.

Figure 8: Occlusion of the middle and rear bars.

For each instance of a class visible in an image, a series of color-coded annotations are drawn. A single rectangular bounding box is added to classify the instance and define the outer boundaries of the instance. One or more polygonal masks are added to specify the visible portion of the object.

A set of color-coded one pixel by one-pixel keypoints are drawn to define the location and visibility of points of interest on the instance. A keypoint can exist in one of three possible states of visibility: visible within the image, not visible within the image, and occluded within the image. Each state is distinguished with a corresponding color.

All three classes share a skeleton of six labeled keypoints: L1, L2, L3, R1, R2, and R3. Keypoints L1 and R1 represent the outermost edge of the bar in each class. Keypoints L2 and R2 represent a downward arc at the mid-point of the bar visible in each instance. Keypoints L3 and R3 represent the flat, horizontal section of the bar for each class.

Figure 9: A fully annotated image

The L3 and R3 keypoints of the rear bar are always occluded and, therefore, always estimated even when labeled manually. The position of the L2 and R2 keypoints for all three classes is challenging to determine consistently. The L1 and R1 keypoints are generally easy to identify but are subject to being cropped out of view when the panhead is at the bottom of the camera frame, and therefore labeled as not visible. This may impact results as the dataset is already limited.

All image labeling was done in Adobe Photoshop, and each annotation is added as an image layer. Once labeled, each annotation layer is exported as a bitmap image and saved alongside the original JPEG image. Bitmap images are used because they provide an anti-aliasing free means of defining edges necessary for generating pixel-level data. The fully annotated Photoshop PSD file is saved for future use as needed. Photoshop actions are used to automate much of the process.

3.1.3 Data Generation

I developed an algorithm to read the type and position of annotation encoded in each of the bitmap images. Annotation data is formatted to be consistent with the COCO data format with images, classes, and annotations being saved to JSON file.

Both automated and manual data validation processes were employed to ensure alignment between labeled image and annotation data. Each master image is opened, and all annotations are overlaid onto the image for a visual inspection. An algorithm ensures all keypoints fall within their respective segmentation mask and that image and annotation data remain in alignment as updates are made to the labeled images. After inspection and validation, all bitmap images associated with a master image are deleted to save file space and must be exported from the PSD file if needed.

The completed dataset consists of 330 images split into a training set of 300 images and a validation set of 30 images. The training set consists of 900 labeled detections with 5290 keypoints, and the validation set has 90 labeled detections with 534 keypoints.

3.2 Network Development

Before working with the Mask R-CNN network, several changes were required to the underlying codebase to allow for the training of the three custom classes and skeleton.

A pantograph class was extended from the base COCO class, and the data loading process was modified to work with the default COCO format. During this data loading process, the three labeled classes are read from the annotation file into the PantographDataset class. The list of keypoints and the list of skeleton connections are read in as class attributes.

A PantographConfig class was written to override the default configuration settings as needed. The number of classes was specified as 4, three custom classes plus the background class required for segmentation. The number of keypoints was specified as 6. The maximum dimension image setting was set to 1024.

Several changes were required to the Mask R-CNN model, utility, and visualization files as well. Throughout the files, hard-coded references were updated to correspond to the appropriate configuration variable. Several functions responsible for resizing the image, mask, and keypoints were updated to work with more modern packages. All hard-coded references to the keypoints and skeleton were updated. I incorporated my own data visualization code into the default code to allow for more control over the process and further assure the alignment of the data as it moves from data creation to model development.

3.2.1 Configuration

I employed a data inspection process to determine optimal configuration settings before training. With a small sampling of images chosen at random from both the train and validation datasets, I tested the updated PantographDataset class methods and Mask R-CNN code in the order they are run when training. The images and any accompanying annotations and data are printed to screen for visual inspection.

Each image is resized to 1024 by 1024 pixels with padding added to the top and bottom of the image to avoid cropping any part of the image. Resizing the images was required to fit multiple images in a batch using a GPU. During this process, resized bounding boxes, segmentation masks, and keypoints are computed.

The use of mini-masks further reduces the amount of memory needed by storing the information about the segmentation mask in an N by N mini mask instead of a full-sized 1024x1024 mask. This offers substantial memory savings since there are generally three positive class instance for each image. Some loss of fidelity can occur during this resizing process, mainly when dealing with smaller masks, as is the case with the middle and rear bars. This loss of fidelity could translate to a reduction in the number of training points available to the network. This loss is also noticeable when the mini-mask is resized to 1024 by 1024 pixels and displayed on the screen. I settled on a mini mask shape of 224 by 224 pixels as a compromise between fidelity and the amount of memory available.

Mask R-CNN supports one generator-based data augmentation technique by default, horizontal flip. In testing, I found this technique added variation to the training information that never occurs in this pantograph dataset. Horizontal flip was disabled before training. I considered other techniques, including equalizing the histogram of the image, but concluded this dataset would not make a good candidate for augmentation due to the limited, but particular variation within the dataset. Training the network on this dataset would benefit from a higher number of images labeled training images rather than data augmentation.

3.2.2 Training

The network was initialized with the weights file provided by the paper's author and was trained using the Google Cloud Platform with a single Nvidia Tesla V10 GPU. With a batch size of 2 images, I was able to process all of the images in each epoch in 150 steps per epoch, training both the backbone and head for a total of roughly 400 epochs. This took approximately 10 hours spread over several sessions. After training both the Resnet backbone and custom head layers, I froze the backbone and trained the head for another 50 epochs.

The final training loss was 1.6, and the validation loss was 1.8.

3.3 Network Evaluation

The trained network was used to make predictions on the labeled validation dataset. The resulting dataset, including predicted classes, bounding boxes, masks, and keypoints results for each of the 30 validation images, was saved to JSON file, then loaded into an evaluation script alongside the labeled JSON file. Pose estimation analysis was performed to compare the performance of the network using the two files.

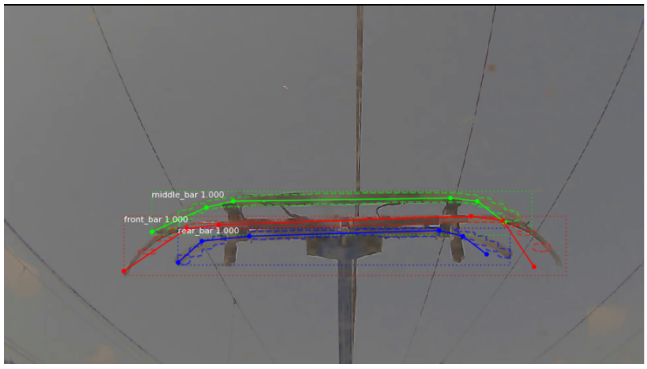

Figure 10: Detection showing misclassification of front and rear bars with segmentation masks and keypoints.

Because the classification and regression score are generated by the network's backbone, and the segmentation mask and keypoints are generated by the head layers, we can evaluate the network's ability to classify instances and its ability to ability to generate masks within the instance separately.

3.3.1 Classification

The first step in evaluation is comparing the labeled classifications to the predicted ones. Comparing figures 10 and 11, we see the network only made 83 detections, seven short of the labeled 90. This is due in part to maintaining a minimum confidence hyperparameter of .9 for detection. Lowering this threshold increased the number of detections, but at the cost of accuracy. At .9, this makes for a successful classification rate of .59.

```
Validation Dataset
------------------
Number of images:        30
Number of detections:    90
Number of keypoints:     534
```

Figure 11: Validation dataset

```
Predicted Dataset
-----------------
Number of images:        30
Number of detections:    83
Number of keypoints:     498
```

Figure 12: Predicted dataset

3.3.2 Pose Estimation Analysis

The second step is to evaluate the network's ability to predict keypoints within a segmentation mask. To do this, I used only the keypoints associated with successful classifications. As before, a hyperparameter can be adjusted to optimize the balance between the number of keypoints predicted and their accuracy. In this subset of data, there are 291 keypoints.

```
count    291.000000
mean      16.748488
std       13.112395
min        0.000000
25%        5.915000
50%       15.030000
75%       21.010000
max       52.090000
Name: PD, dtype: float64
```

Figure 13: Pixel distance summary statistics

Figure 12 shows the mean pixel distance for the predicted points was 16.7 pixels, with 50% of the keypoints being within 15 pixels. To provide a frame of reference for evaluation with regard to solving the original problem, we need to understand what is a usable distance and how many keypoints fall within that distance. The maximum usable pixel distance to build pantograph failure detection systems is 20 pixels. A distance of 5 pixels or less is required to confidently build more accurate computer vision algorithms, including 3D scene reconstruction.

```
PD<=      Count   PCGT
20        210     : 72.16494845360825
10        103     : 35.39518900343643
5         60      : 20.618556701030926
1         5       : 1.718213058419244
```

Figure 14: Keypoints by pixel distance

Figure 14 shows that 72% of the keypoints met the 20-pixel requirement, while just over 20% of the keypoints met the 5-pixel requirement. Five of the predicted keypoints were at 1 pixel or less away from their labeled position.

To further analyze the network and understand where improvements can be made, I broke the results down by class and keypoint.

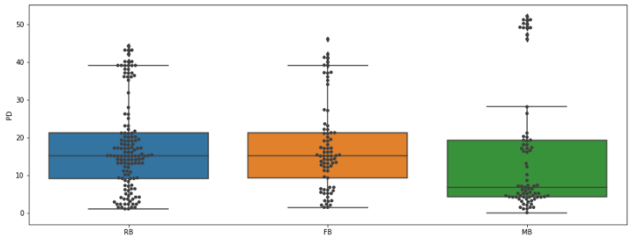

Figure 15: Pixel distance by class

Figures 15 shows a boxplot with values overlaid. From this chart, we can see the rear and front bars on the left and center performing similarly. Both have a roughly even distribution of pixel distances with a close mean as well. The middle bar, however, shows much a much lower mean of 9 but is unevenly distributed. This class has many keypoints at a pixel distance of 5 or less, but also more than ten outlier keypoints greater with a pixel distance greater than 45.

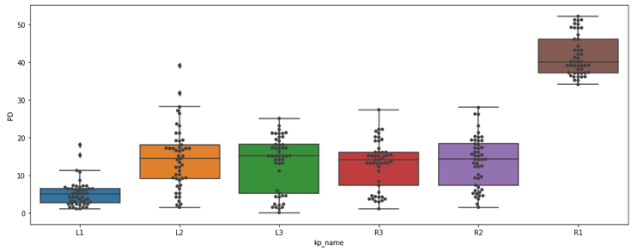

Figure 16: Pixel distance by keypoint

Figure 16 shows a breakdown of pixel distance by keypoint. From this chart, we can see a shared range of values across the middle sections of the bars and a stark difference in values for the left and right most keypoints. This is confirmed by visual inspection of a sample image and its keypoints.

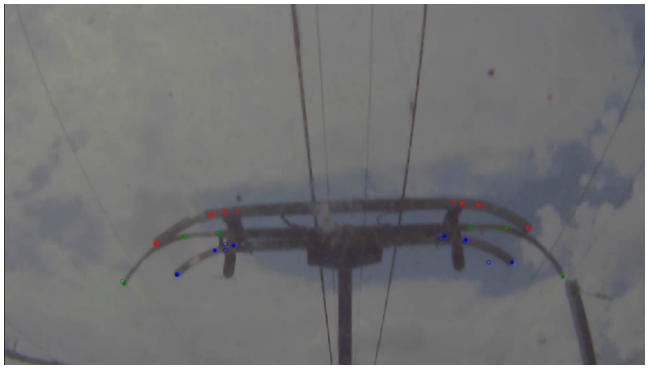

Figure 17: Labeled and predicted keypoints

Figure 17 shows a validation image and its labeled keypoints represented as solid circles and predicted keypoints as stroked circles. Here, we see the increase in pixel distance for each keypoint from left to right.

4. CONCLUSIONS

I have proven the ability of the Mask R-CNN network to learn the movement of the three pantograph components and to successfully predict the position of a series of keypoints associated with each class. While not sufficient for production-level work, the results prove the network is capable of the task. The classification results validate the need for additional labeled data to train the Resnet backbone. Pose estimation analysis shows that a high degree of accuracy is possible given accurate region data from the backbone.

5. FUTURE WORK

I believe the network would show improved results with increased training data. I would recommend a continued iterative approach to training and evaluation until reaching a satisfactory pixel distance measurement.

My code-to-image editor-to-code solution for annotation has proven to be a reliable alternative to commercial and open source

options. It could easily be adapted to annotate other custom classes using a variety of image editing tools.

I would discuss changing the position and alignment of the camera with the owner. Due to the distance from the camera, the smaller parts of the panhead bars are proving challenging to detect. Additionally, I would advise correcting the alignment between the camera and pantograph to reduce the offset distances between the left and right-hand sides of the pantograph.

6. BIBLIOGRAPHY

[1] D. He, Q. Gao, and W. Zhong, "A Numerical Method Based on the Parametric Variational Principle for Simulating the Dynamic Behavior of the Pantograph-Catenary System," *Shock and Vibration*, 2018. https://www.hindawi.com/journals/sv/2018/7208045/ (accessed May 07, 2020).

[2] "[1703.06870] Mask R-CNN." https://arxiv.org/abs/1703.06870 (accessed Mar. 04, 2020).

[3] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," *arXiv:1405.0312 [cs]*, Feb. 2015, Accessed: Mar. 09, 2020. [Online]. Available: http://arxiv.org/abs/1405.0312.

[4] "[1506.01497] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." https://arxiv.org/abs/1506.01497 (accessed Mar. 05, 2020).

About the authors:

Jesse Decker is a graduate student at Mercyhurst University.