Technische Hochschule Rosenheim

Faculty of Engineering Sciences

Localization of objects with Machine Learning using YOLO algorithm

Master's Project

Submitted by

Kunal Jayant Kumthekar (922957)

Pavan Sai Vankalapati (923503)

Rishidhar Kasam (924232)

Sriman Naini (924034)

Vinoth Kumar Ponnusamy (923165)

Vishal Reddy Paladugu (923516)

Guided by

Prof. Dr.-Ing. Michael Wagner

Acknowledgement

This report was created in the third semester of our master's course from March 2020 until August 2020. at the Rosenheim University of Applied Sciences under the guidance of Prof. Dr. -Ing. Michael Wagner.

We would like to take this opportunity to express our gratitude to Prof. Dr. -Ing. Michael Wagner for providing us with his immense knowledge regarding the subject and constantly supporting and guiding us throughout our project. We would also like to thank all our colleagues and friends, who have helped and supported us throughout our project.

We would also like to thank our university, Rosenheim University of Applied Sciences for providing us with the necessary facilities and resources.

Kunal Jayant Kumthekar
Pavan Sai Vankalapati
Rishidhar Kasam
Sriman Naini
Vinoth Kumar Ponnusamy
Vishal Reddy Paladugu

Abstract:

Objective of this work is to detect key-point pairs on a dataset of real images of shafts in a container by training a Convolutional Neural Network (CNN) model using state-of-the-art TinyYOLO v2 architecture. The model would be trained on a synthetically generated shaft image dataset, created using Blender™. This work is based on the hypothesis that CNN's trained on synthetically generated images obtained through simulations, would be successful in inferring key-point's on real images of shafts captured through a camera. An attempt would be made to replace the synthetic image dataset of completely visible shafts with a synthetic image dataset of shafts which are stacked such that visibility is partial. The first stage of this work includes creation of an artificial shaft dataset using Blender™, a free and open-source 3D computer graphics toolset. Second stage of this work includes making modifications to the existing YOLO v2 based CNN model to detect partially visible shafts inside a container. The final stage involves the prediction of key points on a dataset of real images of shafts in a container.

Contents

List of figures

1. Introduction

Object Detection is a computer vision technology which helps in identifying and locating the objects and their classes in each image. Object Detection is a combination of two phenomena namely Image Classification and Object Localisation. Image Classification is defined as the process in which the class or the type of the image is predicted. For example, if an image with an object is given as input, then the output is the image with the defined class label. Object Localisation is defined as the process in which the objects present in an image are located and their locations are indicated using a bounding box or a key-points.

"**The objective of our project is to localise the shafts in a real image based on the training done on the generated synthetic dataset, with the help of YOLO algorithm**"

There are different algorithms which can be used for Object Detection. Some of the widely known object detection algorithms are

- Region Proposals such as R-CNN, Fast R-CNN, Faster R-CNN and cascade R-CNN.
- You Only Look Once (YOLO)
- Single Shot Multibox Detector (SSD)
- Retina-Net

We have preferred to use YOLO because it performs in a better way compared to the other object detection algorithms. YOLO is faster and more accurate compared to the other object detection algorithms in a real-time environment. YOLO uses a single neural network for the whole image instead of the use of classifiers which is done in the other algorithms. The version of YOLO used for our project is TinyYOLO-v2.

1.1 YOLO v2

YOLO is a state-of-the-art real time object detection framework. YOLO v2 utilizes the Darknet-19 architecture for object classification. In YOLO, the algorithm back propagates only the losses in bounding boxes/grid cells which have a higher probability of having a particular class. In this way the algorithm does not need to train for a specific class, and hence the predictions can be made much more generalized. As the confidence scores are calculated in the grid cells, there comes a time of the dataset training when a particular threshold of the confidence scores is reached. As the threshold values are reached, a class having the maximum confidence score in a particular grid cell is assigned to that grid cell. In this way YOLO can have fast real time processing and can be trained to achieve greater accuracy. Considering all these attributes, YOLO v2 predicts fewer false positives and is an ideal architecture for even multiclass classification. In this algorithm the input image is divided into equally sized grids say (s*s). Bounding box and class of objects present in every grid predicted. In our project bounding boxes replaced by key point pairs. Every grid cell predicts key point pairs, they are centre of the shaft (X0, Y0) and angle of the shaft (alpha). and we have only one object type which is shaft so, we have just one class.

1.2 TinyYOLO-v2:

TinyYOLO (also known as tiny DarkNet) is a lighter version of the YOLO object detection deep learning neural network. The normal YOLO v2 architecture consists of 24 convolutional layers followed by 2 fully connected layers. In the case of TinyYOLO-v2, there are 9 convolutional layers, each with batch normalisation operation, leaky ReLU (Rectified Linear Unit) based activation function and followed by 6 max pooling layers.

TinyYOLO is faster than its predecessor. The fastest architecture of YOLO achieves a speed of 45 FPS whereas TinyYOLO achieves speeds up to 244 FPS on a computer with a GPU. Below is a pictorial representation of the architecture.

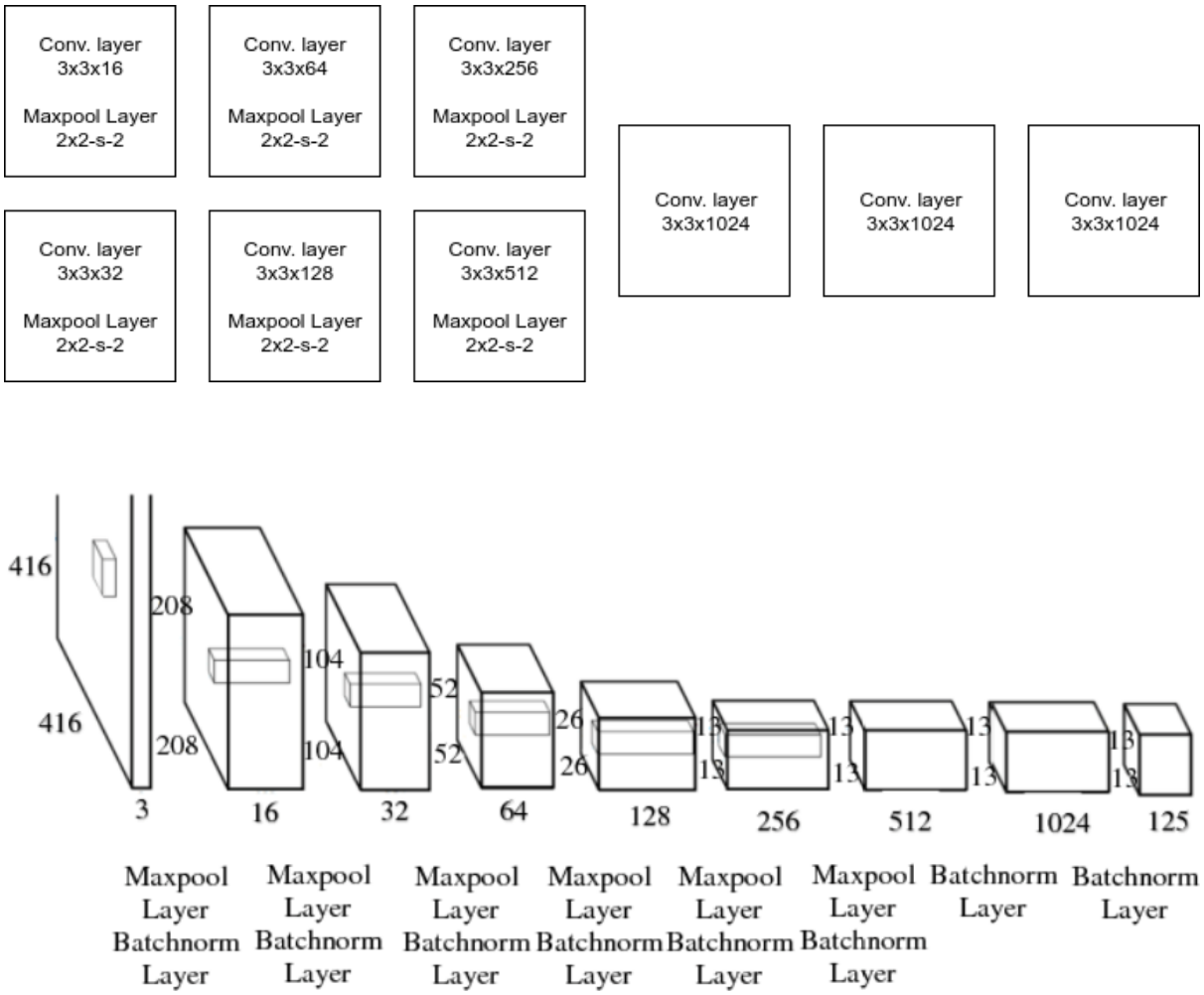


Figure 1 TinyYOLO v2 architecture

The layers used in this algorithm are somewhat similar to that used in the basic CNN with just a small change. The layers are namely,

- **Convolution Layer:** It consists of 4 steps. They are
  - Step 1: Line up the feature and the image
  - Step 2: Multiply each image pixel by the corresponding feature pixel.
  - Step 3: Add them up.
  - Step 4: Divide by the total number of pixels in the feature.

And thus, the value obtained is put at the position of the feature. Following the above steps, the filter moves to each and every position and therefore generates an output with a reduced image size.

- **Batch Normalisation:** It is defined as a technique which is used to improve the speed, performance and stability of the artificial neural network. It is used to normalize the input layer by re-scaling and re-centering.

- **ReLU (Rectified Linear Unit) Layer:** ReLU transform function only activates a node if the input is above a certain quantity. If the input is below zero, output is zero and if the input rises above a certain threshold, it has a linear relationship with dependent variables.

$$f(x) = 0 \text{ if } x < \text{given threshold}$$

$$f(x) = x \text{ if } x >= \text{given threshold}$$

- **Pooling Layer:** In this layer, we shrink the image into a smaller size by using the following steps
    - Pick a window size.
    - Pick a stride.
    - Move the window across the filtered images.
    - From each window take the maximum value.

A basic CNN also consists of another layer called Fully Connected Layer which is not used in TinyYOLO. This is the difference in TinyYOLO compared to the basic CNN.

1.2.1 Custom Loss:

YOLO model architecture is simple, but the loss function is quite complex. YOLO uses the mean-squared error between predicted and actual observations to calculate the loss. To measure the error, the loss function takes the ground truth(target) value (True) and predicted value (Pred).

The custom loss function comprises of:

1. **Localization loss**

It calculates the error between true and predicted keypoint pairs, centre point(x, y) and alpha angle. We multiply this loss by a weight factor 'λ coord' to give more penalty and directions scale is multiplied with only alpha.

2. **Confidence loss**

--If the object is detected confidence loss as follows

$$\lambda_{obj} * \lambda_{conf} \sum_{i=0}^{s^2} [(True_{conf} - Pred_{conf})^2]$$

-- if the object is not detected confidence loss as follows

$$\lambda_{no\_obj} * \lambda_{conf} \sum_{i=0}^{s^2} [(True_{conf} - Pred_{conf})^2]$$

In some cases, we could not detect objects or key points. It gives rise to class imbalance problems, so we train the model to identify the background more frequently than detecting objects or parts. To do this we penalize it by noobj.

3. **Classification loss (class loss)**

If we have different classes, then classification loss plays a role. But in our case, we have only one class, so this loss doesn't affect our final loss.

Final Loss: We combine localization, confidence, and classification losses.

$$\text{Final loss} = \lambda coord \sum_{i=0}^{s^2} [(True_{kpxy} - Pred_{kpxy})^2]$$

$$+ \lambda_{ds} * \lambda coord \sum_{i=0}^{s^2} [(True_{alpha} - Pred_{alpha})^2]$$

$$+ \lambda_{obj} * \lambda_{conf} \sum_{i=0}^{s^2} [(True_{conf} - Pred_{conf})^2]$$

$$+ \lambda_{no\_obj} * \lambda_{conf} \sum_{i=0}^{s^2} [(True_{conf} - Pred_{conf})^2]$$

$$+ \sum_{i=0}^{s^2} \sum_{c \in class} [(True_{class} - Pred_{class})^2]$$

2. Methodology:

The methodology used in this work is as follows-

1. Creation of synthetic shaft dataset using Blender™.
2. Migration of the working algorithm from TensorFlow v1.xx to TensorFlow v2.xx.
3. Modification of existing CNN model and dataset pre-processing by introducing thresholds for avoiding partially hidden shafts for training.

2.1 Creating Synthetic Images from Blender using Python:

- The objective is to generate several images and respective annotation files. The shafts are created using Blender and the scripting language is Python. Blender has a complete Python installation.
- A case is designed at first and then physics is added to that case. Adding physics enables “Rigid body simulation”.
- Later Shaft is created and then physics properties are added to it too.
- After designing and adding physics properties to the shaft, making vertex groups of the shaft will be helpful to calculate the confidence intervals.

- Before duplicating, rotation is applied. Then it can be duplicated into essential number of shafts.
- Camera is added fitting the case resolution and light can be adjusted as per the user's requirement.
- Annotation file is created by extracting the required values.
- Each annotation file considered the following values:

  • Class

  • Pixel coordinates of the centre of shaft (X, Y)

  • Angle between the shaft and X-axis (Global)

  • Angle between the shaft and XY plane

  • Confidence intervals

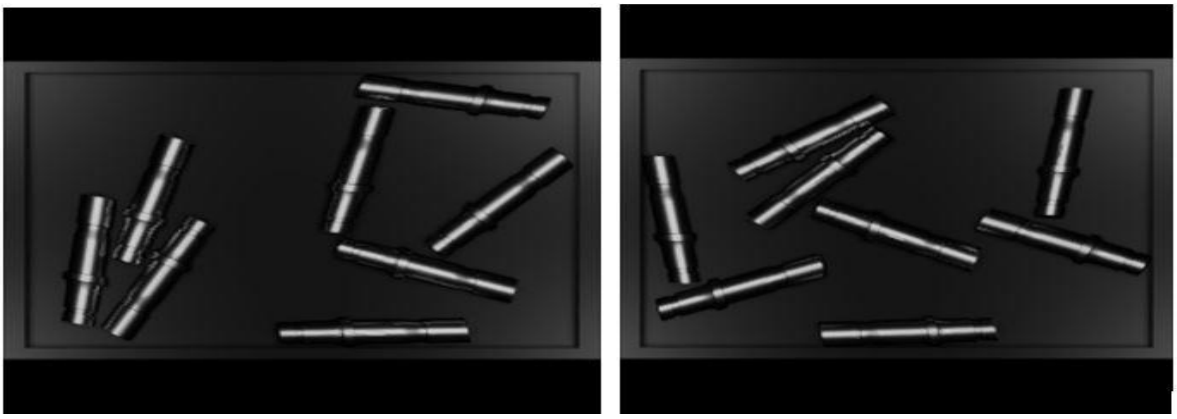


Figure 2 Synthetic Images

```
0 68.59434509277344 165.91537475585938 2.0800457318597516 3.079460023092576 1.0 1.0
0 153.33865356445312 204.4207000732422 3.0849672985547265 0.04812875588471499 1.0 1.0
0 176.6009979248047 163.25100708007812 0.322771281003952 0.03381577332551089 1.0 1.0
0 62.07167434692383 126.14572143554688 1.8756749629974365 3.076829621689864 1.0 1.0
0 35.74238967895508 164.35633087158203 1.7102231979370117 3.079037921624728 1.0 1.0
0 208.9655303955078 126.73431396484375 2.346468448638916 3.0778297429257115 1.0 1.0
0 197.9403533935547 57.17698669433594 0.20444469153881073 3.074224295216151 1.0 1.0
0 150.55764770507812 106.5189208984375 1.795143723487854 3.1330354857542595 1.0 1.0
```

Figure 3 Annotation file

2.2 Migration of code from Tensorflow 1.14 to 2.2.0

TensorFlow is a free open source deep learning framework owned and maintained by Google™. Computations though TensorFlow are particularly fast, versatile, and can be used across multiple platforms. In this work, a previously built TinyYOLO-v2 based CNN model on TensorFlow 1.14 was migrated to TensorFlow 2.2.

As of the date of creation of this document, the following limitations apply,

- Visual Studio 2019 and newer are not supported by CUDA toolkit 10.0.
- NumPy version 1.16.0 is required, any version above or below the stated version may cause binary incompatibility errors.
- Install h5py using conda at the end of setup to limit dependent packages being upgraded or downgraded to incompatible versions.

2.2.1 **Specifications**

- Windows 10 (version 1909 OS Build 18363.752)
- Anaconda Navigator 1.9.12 or newer
- CUDA (version 10.2)
- cuDNN (version 7.6.5)
- Nvidia CUDA capable GPU (depends on the Graphic card)
- Microsoft Visual Studio 2019 with Visual 2015 build tools, optionally Microsoft Visual Studio Code.

2.2.2 Commonly Encountered Issues:

1. ‘pip [WinError5]’ error occurs during installation?

- Run the command “conda update conda” before attempting to create an environment.
- If issue persists, set up the environment in a non-system drive by adding the following argument when creating the conda environment:

$$conda\ env\ create\ --prefix\ "\{InstallLocation\}"$$
$$-f\ "\{ymlFileLocation/environment.yml\}"$$

- Replace $\{InstallLocation\}$ with the absolute path to a non-system directory where you would like to set up the environment.

2. The new environment is not visible in Visual Studio Code:

• Reload the developer window by restarting Visual Studio Code or press Ctrl + Shift + P and then type "Reload Window".

2.3 Modifications

2.3.1 Annotation File:

The Annotation file contains every parameter that is required for shaft localization. These parameters include the centroid of individual shafts, orientation of each shaft, the visibility level, etc. The figure below displays the pictorial representation of parameters in an annotation file.

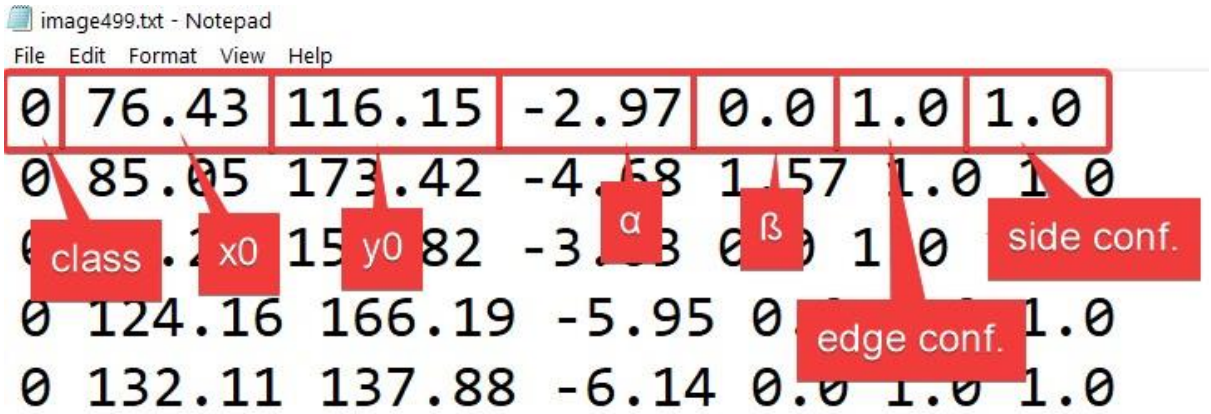


Figure 4 Annotation file layout

- **Class**: represents the class (type) of object. In our case only shafts are the trainable objects hence class remains "0".
- **x0, y0**: Represents the centre coordinates of shaft in the grids.
- **Alpha** (α): Represents the angular position of the shafts with respect to grid coordinate system. The angle is measured in radians.

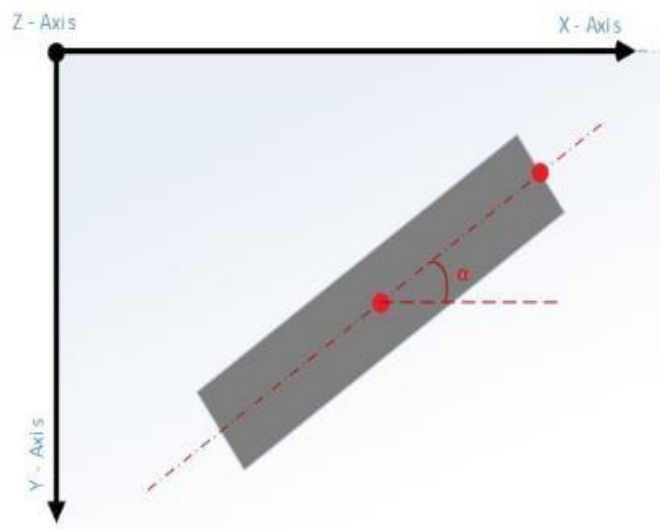


Figure 5 Shaft orientation angle (α)

- **Beta (β)**: Represents the angle of inclination with respect to the X-Y plane on which shaft is placed horizontally.

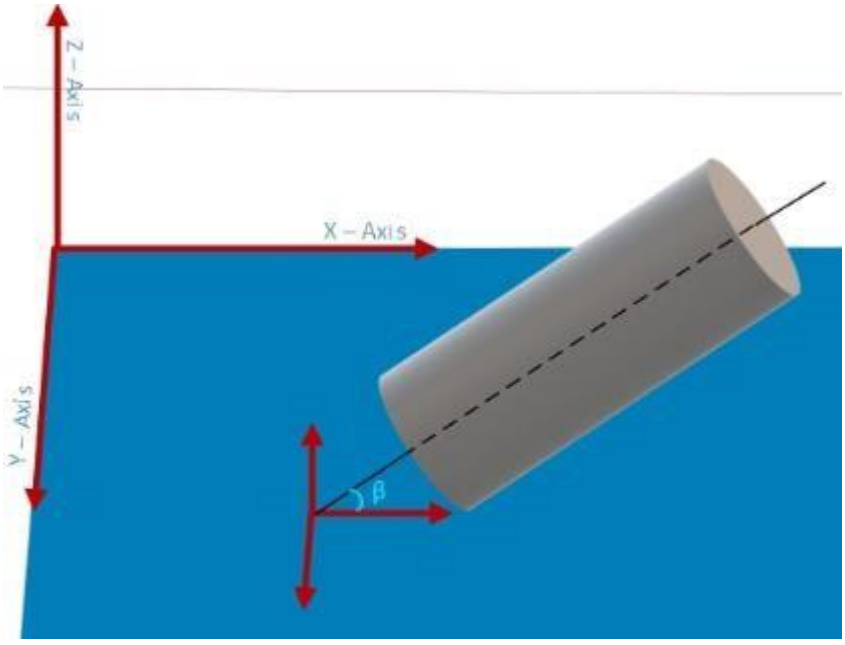


Figure 6 Shaft inclination angle (β)

- **Edge/ Side confidence:** This confidence parameter is a ratio of number of visible vertices along the edge or side of a shaft to ideal number of vertices on the edge/side of a shaft.

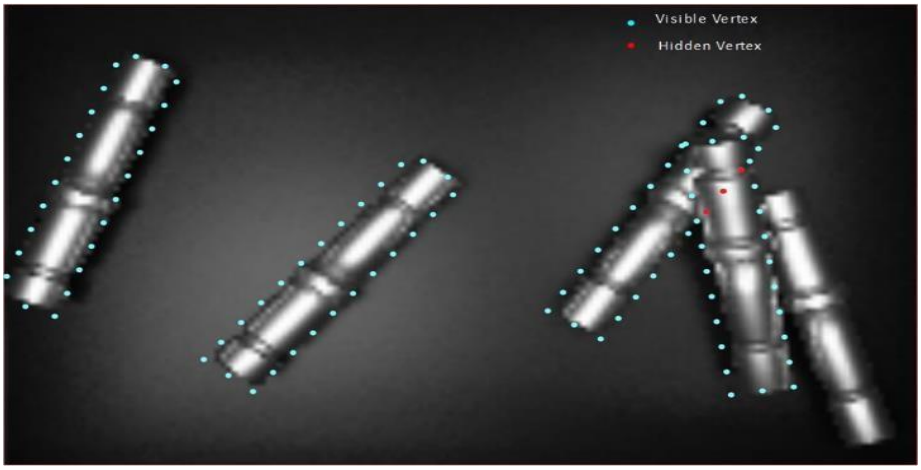


Figure 7 Vertices on simulated shaft

2.3.2 Project Pipeline:

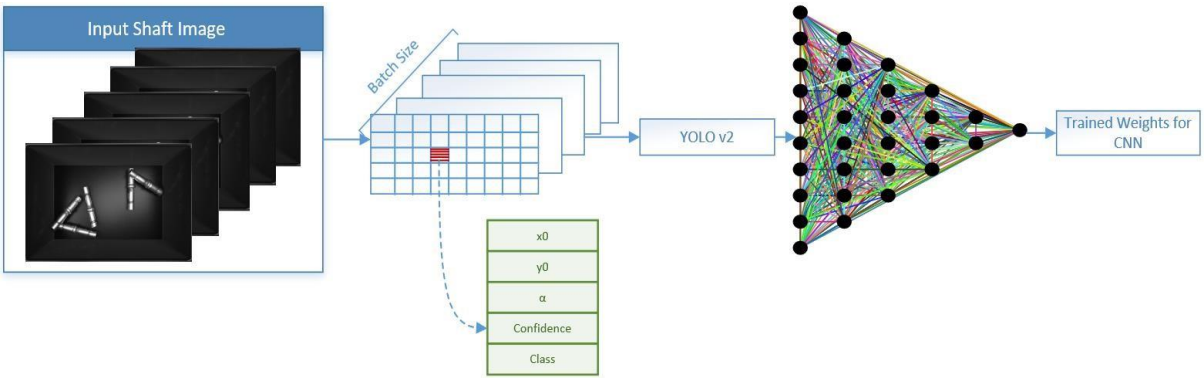


Figure 8 Project Pipeline

- Project pipeline begins with data pre-processing, beginning with reading the annotation files and their corresponding input artificial shaft image.
- Shafts would be localised or predicted within grids. Every grid would consist of attributes mentioning the presence of the shafts.
- These attributes are: x0, y0, α, confidence, class.

- The entire dataset of artificial images is converted into mini batches for ease of training purpose.
- The images from the mini batch are then transformed with the help of the image augmentation library called 'Imgaug'.
- This transformation of the images is accompanied along with the simultaneous transformation of the data values from the annotation file, indicating the position of the shafts in the transformed image.
- The transformations applied to the images are randomized.
- Image transformation is done to avoid or minimize the chances of the CNN model being trained on the same image sample multiple times, which could lead to underfitting.
- The YOLO v2 based neural network, consisting of a number of dense layers, is now trained with these mini batches of images.
- The batch size can be treated as a hyperparameter, that is, the number of images per batch is set and configured by the user.
- As the training process begins, a file containing weights (learning parameters) is generated which would be later used for the prediction process on real shaft images

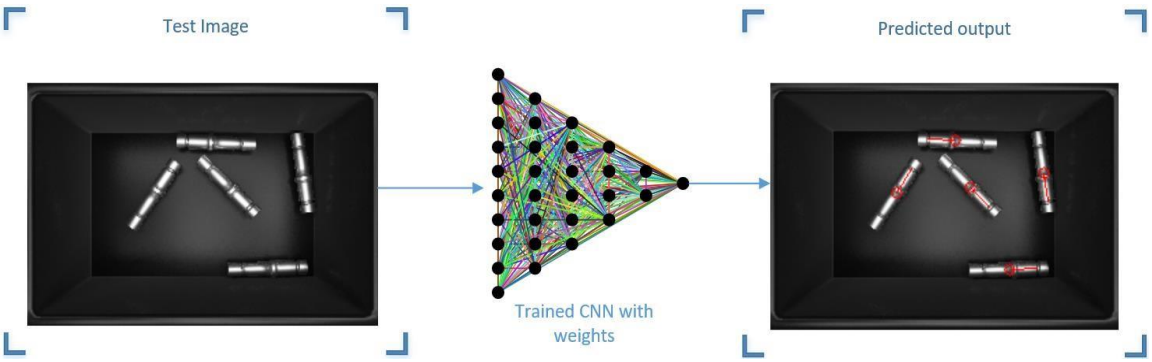


Figure 9 Inference with trained model

- As mentioned in the above picture, any test (essentially an image not from either training or cross validation images) real shaft image is plugged into the pre-trained weights (generated by training artificial shaft images) and predictions of keypoint images are obtained onto the shafts.
- The precision of detection, however, depends upon the accuracy of features learned by the neural network.
- There are a number of factors affecting the accuracy, namely, number of epochs (number of passes the mini batches make through the CNN), learning rate, and others.

2.3.3 Visibility Threshold:

To achieve the objective of avoiding the shafts having lower confidence values than a particular threshold, two confidence thresholds were introduced in the algorithm.
These thresholds are namely, 'obj_edge_vis_thresh' and 'obj_side_vis_thresh'. These thresholds are set for both training as well as validation image datasets.

```
"train": {
"train_image_folder": ".\\dataset\\train\\",
"train_annot_folder": ".\\dataset\\train\\",
"train_times": 10,
"pretrained_weights": "shaft.h5",
"batch_size": 16,
"learning_rate": 1e-3,
"nb_epochs": 250,
"warmup_epochs": 3,
"object_scale": 5.0,
"no_object_scale": 1.0,
"coord_scale": 1.0,
"class_scale": 1.0,
"direction_scale": 10.0,
"obj_edge_vis_thresh": 0.85,
```

```
16.         "obj_side_vis_thresh": 0.85,
17.         "saved_weights_name": "shaft.h5",
18.         "debug": true
19.         },
20.         "valid": {
21.         "valid_image_folder": ".\\dataset\\valid\\",
22.         "valid_annot_folder": ".\\dataset\\valid\\",
23.         "valid_times": 1,
24.         "obj_edge_vis_thresh": 0.85,
25.         "obj_side_vis_thresh": 0.85
26.         }
```

Setting a threshold value of 0.85 would mean any shaft having a confidence value less than 0.85 would be avoided for training purposes.

The threshold values are implemented in a '.json' file to avoid any hardcoding of values.

```
if (edge_threshold < float(vWords[5])) and (
    side_threshold < float(vWords[6])
```

The above image represents the part of the code where the threshold values are compared with the edge/side confidence values read from the annotation values (v_words[5] as mentioned represents edge_confidence while v_words[6] represents the side_confidence of shaft).

2.3.4 Data Pre-processing:

Annotation file provides us with the centre coordinates of shafts. However, in order to represent the shaft's direction and accuracy of key points along the shaft's axis a second key point computation is necessary. Following lines of code represents the computation of second key point pair (x1,y1).

```
def annot_on_image(imageLoc,objectLis):
    image_path = imageLoc     image
= cv2.imread(image_path)     for
shaft in objectLis:
        x0_floored = int(np.floor(shaft["x0"]))         y0_floored =
int(np.floor(shaft["y0"]))         x1_floored =
int(np.floor(shaft["x1"]))         y1_floored =
int(np.floor(shaft["y1"]))         cv2.circle(image, (x0_floored,
y0_floored), 4, (0, 0, 255), 1)
cv2.line(image,(x0_floored,y0_floored),x1_floored,y1_floored),(
0,0,255),1)
    cv2.imwrite(image_path[:-4]   +   "_detected_test"   +image_path[-4:],
image)
```

```
1. obj["x0"] = float(vWords[1])
2. obj["y0"] = float(vWords[2])
3. angle_alpha = float(vWords[3])
    obj["x1"] = float(obj["x0"] - 20.0 * math.cos(angle_alpha))
    obj["y1"] = float(obj["y0"] - 20.0 * math.sin(angle_alpha))
```

As mentioned above, once the centre coordinates of shafts (x0,y0) are read from an annotation file, new key point pairs (x1, y1) are computed with respect to alpha angle.

```
cv2.circle(image, (x0, y0), 4, (0, 0, 255), 1)
cv2.line(image, (x0, y0), (x1, y1), (0, 0, 255), 1)
```

After the two key point pairs are computed, simple OpenCV functions are used to draw a line and a circle to denote direction as mentioned above.

3. Model Training and Results

Model training was performed with a combination of different datasets and different model structures of varying depth.

3.1 Real Images (Dataset 1) with 13 Convolutional Layers:

Initially a model variation consisting of 13 convolution layers was trained on a dataset containing 266 images of shafts in a container. The images utilized during this stage are real images, i.e. the images were captured through a camera positioned above a container with the shafts.

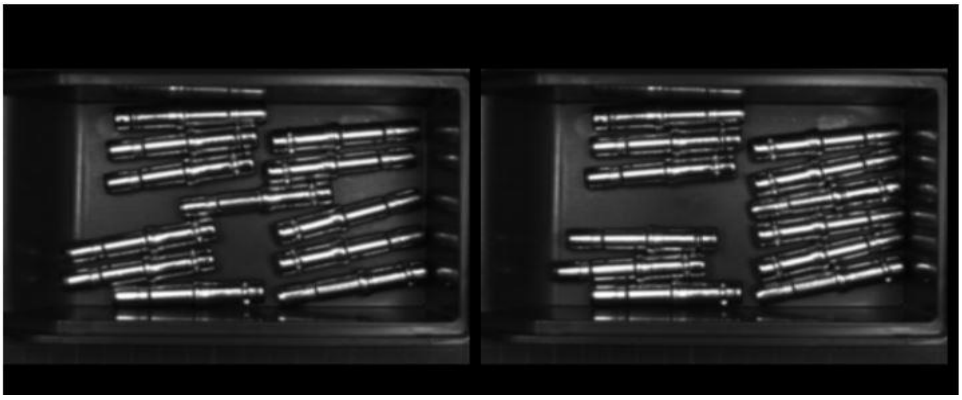


Figure 10 : Real Images captured through a camera

The dataset used for training was split into two with the ratio 4:1 for training and validation, respectively. The model was trained on the dataset for 100 epochs.

The trained model, i.e. the above-mentioned model trained on real images of shafts in container, was used to infer key-points and the orientation angle on test images from the same dataset, i.e. real images.

Figure 11 Inference using model trained on real images

The inferred key-points and predicted orientation angle of the shafts are shown above. The inference performance on test images is such that all shafts are recognized, whereas the predicted orientation angle is within acceptable limits.

```
68.16776031600244 73.8692101204331 0.11984888 0.72966343 [0.72966343]
149.58580151948692 75.18817132784477 0.8782802 0.7337114 [0.7337114]
77.55749120530348 90.89888704352231 0.15183425 0.7285704 [0.7285704]
163.5982600876712 90.27161163130879 0.886868 0.7311424 [0.7311424]
122.04367565018063 126.15486525869868 0.11622869 0.7145729 [0.7145729]
57.70320123499535 130.50448000128978 0.6294248 0.73019725 [0.73019725]
217.24163274120517 133.71505201649754 0.46289876 0.7300503 [0.7300503]
201.5337822691677 142.85471599241453 0.4246128 0.716796 [0.716796]
126.05759301470346 146.1888967336646 0.13152531 0.7291324 [0.7291324]
174.45209000515067 146.64317974036769 0.39849785 0.72705203 [0.72705203]
76.39834731808838 171.82089639257092 0.835801 0.7273345 [0.7273345]
106.93866047799078 187.9290257578315 0.89671624 0.727512 [0.727512]
12 keypoints are found
```

Figure 12 Model prediction for Image in figure 11 on the left

3.2. Synthetic Images (Dataset 2) with 13 Convolutional Layers:

Synthetic images obtained through simulations, performed using an opensource 3D computer graphics toolkit called as Blender ™, were used. The images obtained through part simulation were initially not photo realistic even though the part geometry was identical to that of real shafts.

Figure 13 Synthetic Images Dataset 2

The above synthetic shaft dataset was trained for 100 epochs. The depth of the model was kept unchanged, i.e. model shape was identical to that of the previous case. The shaft dataset was split into same ratio of training and validation images as that of the above case (Case 1).

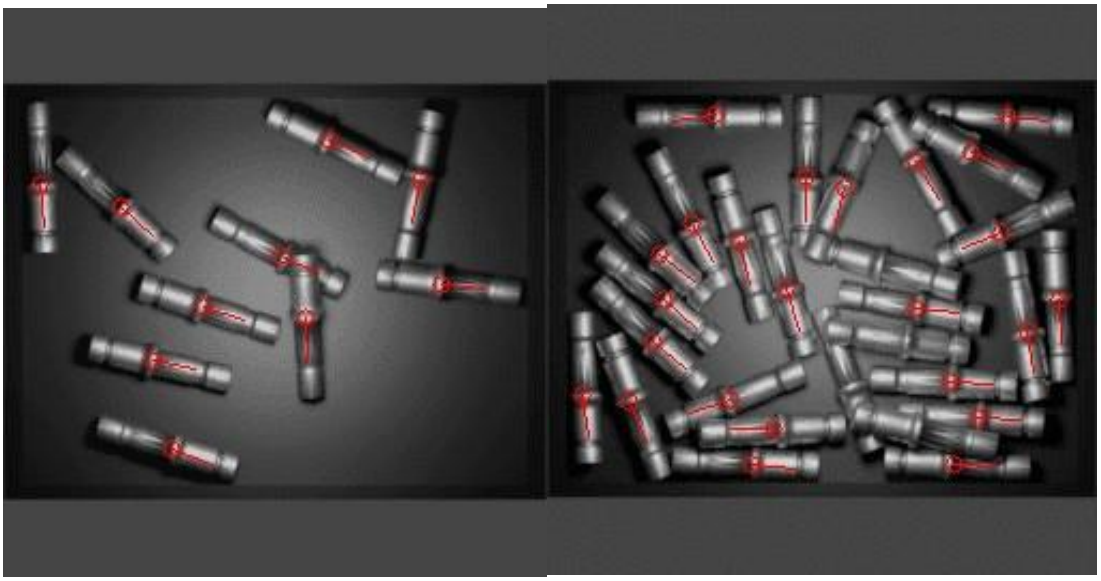


Figure 14 Inference using model trained on dataset 2

Model predications for test images consisting of synthetic images of the same quality as that of the dataset are shown above. Inferred key points and predicted orientation angles for shafts where within acceptable limits. The model trained on synthetic images was saved and then was used for performing inference tasks on real images of shafts.

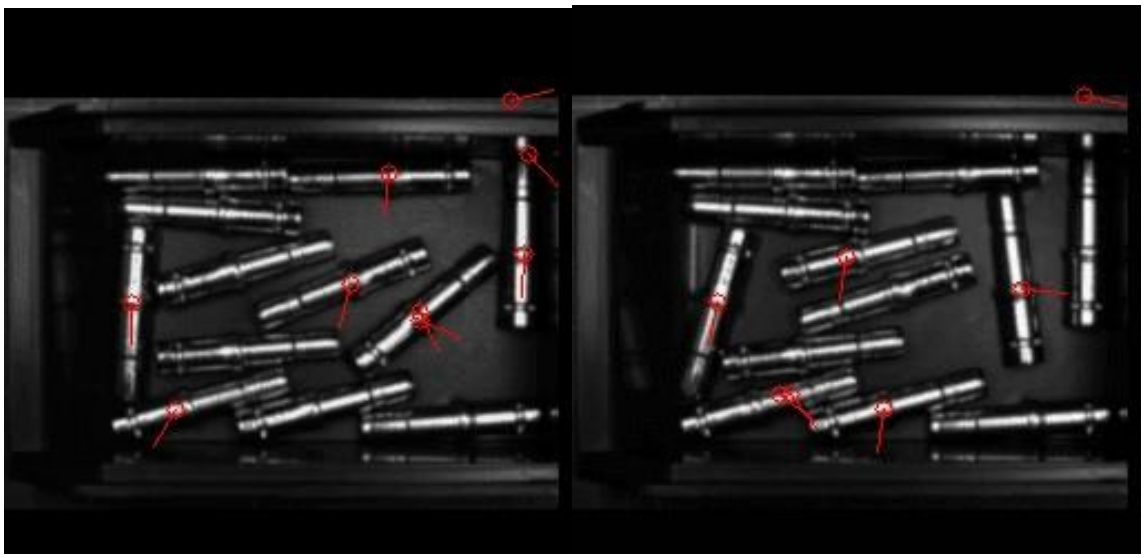


Figure 15 Inference using model trained on dataset 2 on real shaft images

The predictions for real images are shown above. Unfortunately, the predictions made on real shaft images does not localize the image well. It could be observed that both precision and recall are poor.

It was hypothesized that the reasons for the above poor results are as follows:

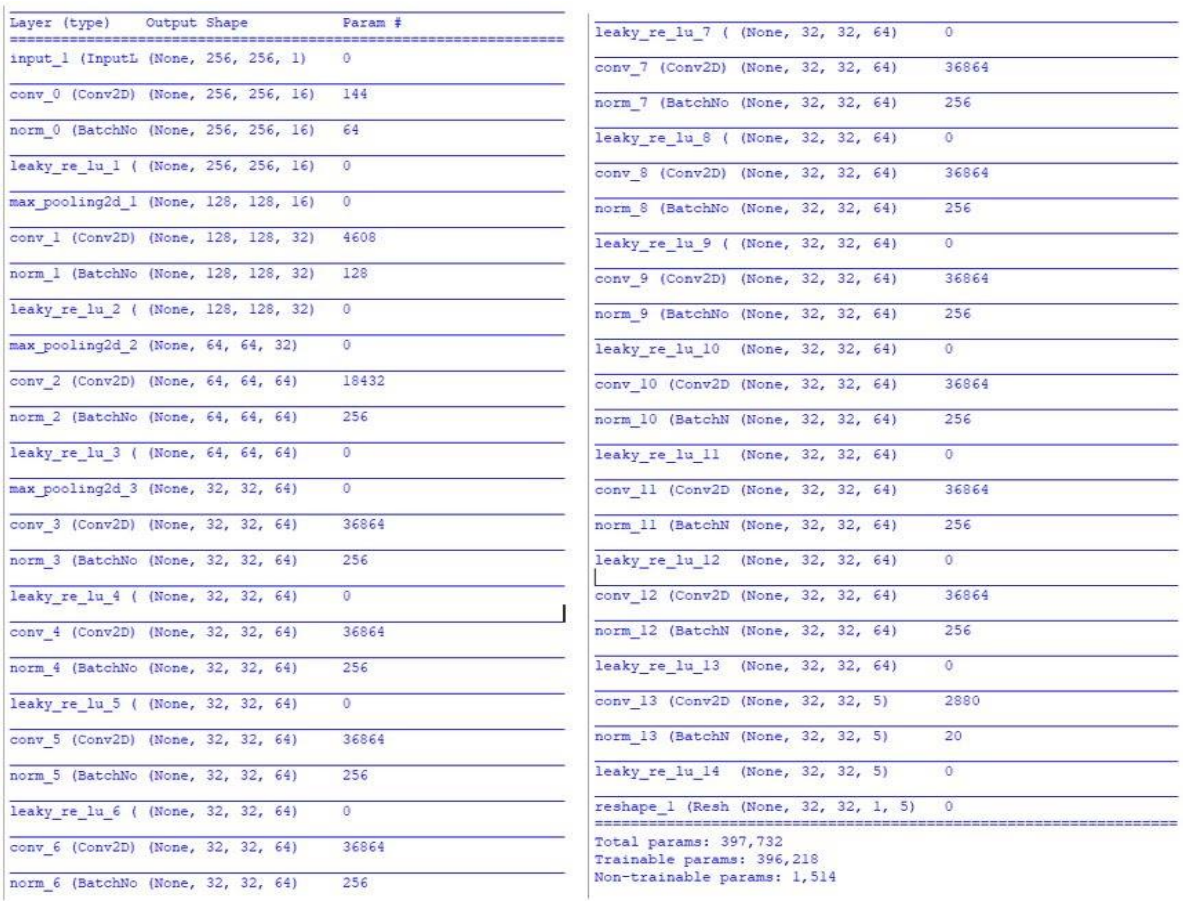
- The length of the shafts is shorter as compared to the real shafts.
- The synthetic shafts do not resemble the real shafts as we can clearly see from the above images.
- The background is also different compared to that of a real image.

3.3 Synthetic Images (Dataset 3) with 13 Convolutional Layers:

Based on our observations in the above dataset, we created a new dataset, A new dataset (Dataset 3) comprising of 1000 images was created, in which the simulated shafts where made to resemble the real shaft images as much as possible. The shape of the model was once again kept unchanged. The model was trained once again for 100 epochs on the new dataset (Dataset 3).

Figure 16 Synthetic Images of Shafts (Dataset 3)

Inference tasks were performed with the trained model on test images comprising of synthetic images from the dataset 3. We trained our model with 13 convolutional layers. Trainable parameters were 3,96,218 and not-trainable parameters were 1,514. we can see that in Image 14.



Layer (type)	Output Shape	Param #
input_1 (InputL	(None, 256, 256, 1)	0
conv_0 (Conv2D)	(None, 256, 256, 16)	144
norm_0 (BatchNo	(None, 256, 256, 16)	64
leaky_re_lu_1 (	(None, 256, 256, 16)	0
max_pooling2d_1	(None, 128, 128, 16)	0
conv_1 (Conv2D)	(None, 128, 128, 32)	4608
norm_1 (BatchNo	(None, 128, 128, 32)	128
leaky_re_lu_2 (	(None, 128, 128, 32)	0
max_pooling2d_2	(None, 64, 64, 32)	0
conv_2 (Conv2D)	(None, 64, 64, 64)	18432
norm_2 (BatchNo	(None, 64, 64, 64)	256
leaky_re_lu_3 (	(None, 64, 64, 64)	0
max_pooling2d_3	(None, 32, 32, 64)	0
conv_3 (Conv2D)	(None, 32, 32, 64)	36864
norm_3 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_4 (	(None, 32, 32, 64)	0
conv_4 (Conv2D)	(None, 32, 32, 64)	36864
norm_4 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_5 (	(None, 32, 32, 64)	0
conv_5 (Conv2D)	(None, 32, 32, 64)	36864
norm_5 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_6 (	(None, 32, 32, 64)	0
conv_6 (Conv2D)	(None, 32, 32, 64)	36864
norm_6 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_7 (	(None, 32, 32, 64)	0
conv_7 (Conv2D)	(None, 32, 32, 64)	36864
norm_7 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_8 (	(None, 32, 32, 64)	0
conv_8 (Conv2D)	(None, 32, 32, 64)	36864
norm_8 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_9 (	(None, 32, 32, 64)	0
conv_9 (Conv2D)	(None, 32, 32, 64)	36864
norm_9 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_10	(None, 32, 32, 64)	0
conv_10 (Conv2D	(None, 32, 32, 64)	36864
norm_10 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_11	(None, 32, 32, 64)	0
conv_11 (Conv2D	(None, 32, 32, 64)	36864
norm_11 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_12	(None, 32, 32, 64)	0
conv_12 (Conv2D	(None, 32, 32, 64)	36864
norm_12 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_13	(None, 32, 32, 64)	0
conv_13 (Conv2D	(None, 32, 32, 5)	2880
norm_13 (BatchN	(None, 32, 32, 5)	20
leaky_re_lu_14	(None, 32, 32, 5)	0
reshape_1 (Resh	(None, 32, 32, 1, 5)	0

Total params: 397,732
Trainable params: 396,218
Non-trainable params: 1,514

Figure 17 Model architecture of 13 convolutional layers

Figure 18 Inference results for synthetic test images

Model predications for synthetic test images are shown above. Shaft localization predications were satisfactory. The orientation angle prediction for majority of detected shafts were within acceptable limits.

The trained model was now used to perform key-point and orientation inference on real images of shafts. Shaft localization was much better in comparison to that of the model trained on dataset 2, i.e. precision for shaft localization was much higher than that of the previous model. Below are the prediction results for real images of shafts in a container.

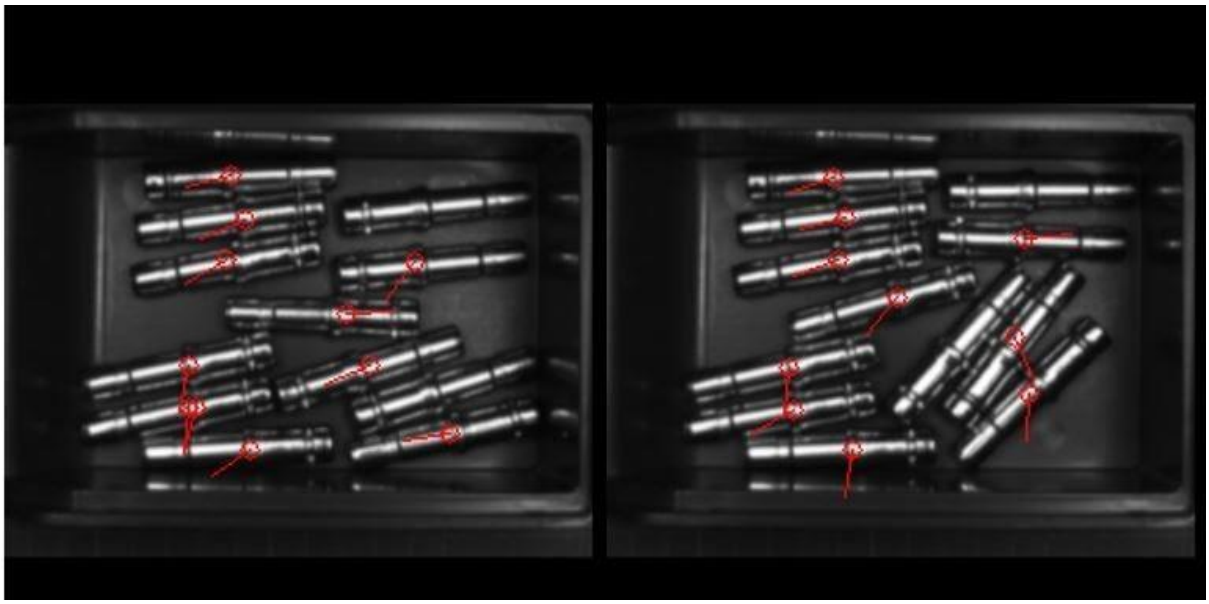


Figure 19 Inference using model trained on dataset 3 on real shaft images

```
96.82188581690342 72.89921045546886 0.8408904 0.74262 [0.74262]
102.30466382742178 91.29864028443795 0.79778105 0.74232167 [0.74232167]
94.61944317787871 108.77348189538678 0.7692826 0.7320924 [0.7320924]
174.65237881150753 109.32704734809998 0.6612329 0.7573623 [0.7573623]
145.91225836777508 131.8423275818634 0.07720229 0.71260506 [0.71260506]
78.29489001003697 153.83114380018404 0.5527313 0.72271645 [0.72271645]
155.19807595613773 153.92772707909216 0.7999278 0.75385994 [0.75385994]
78.3619653639111 171.9946230309065 0.51977164 0.7117193 [0.7117193]
81.15254998314855 171.87535971552265 0.55388486 0.70617026 [0.70617026]
189.38475860842274 182.4618396680767 0.855696 0.7402818 [0.7402818]
104.61856093983774 189.39363201481405 0.7487756 0.7462472 [0.7462472]
11 keypoints are found
```

Figure 20 Model prediction for Image on the left in figure 19

The results were much better when compared to previous data set, but these results were also not up to our expectations, so we started experimenting to get better results. First we changed the number of convolutional layers to 09 and started training our model.

3.4 Dataset 3 (case1) with 9 Convolutional Layers:

The dataset considered for training was kept the same i.e. dataset 3. However, the number of convolutional layers was reduced from 13 to 9. After training the dataset for 100 epochs, the following inference has been made:

Figure 21 Inference for synthetic shaft images

```
103.00791641150803 65.15760998188226 0.17687541 0.74533516 [0.74533516]
44.3866868109489 80.93795038034105 0.34787855 0.7572869 [0.7572869]
113.46660165227242 94.7824551207765 0.2025432 0.7251442 [0.7251442]
111.42040831208713 117.275922318335 0.27307978 0.763278 [0.763278]
46.665910336425824 122.281051513548 0.13909394 0.725843 [0.725843]
236.4555775350506 150.47559321137243 0.49163282 0.7486147 [0.7486147]
130.10954624341326 171.98078419038993 0.75096285 0.7535282 [0.7535282]
180.12201582208905 179.59965186148224 0.67403805 0.7509155 [0.7509155]
8 keypoints are found
```

Figure 22 Model prediction for Image on the left in figure 21

As shown above, predictions made on the synthetic test images with the trained model with 9 convolutions are comparatively similar to that of the predictions made on CNN with 13

convolutions. However, when predictions were made on real shaft images, the recall as well as precision of key-point localization was inadequate. Below are the predictions made on real shaft images:

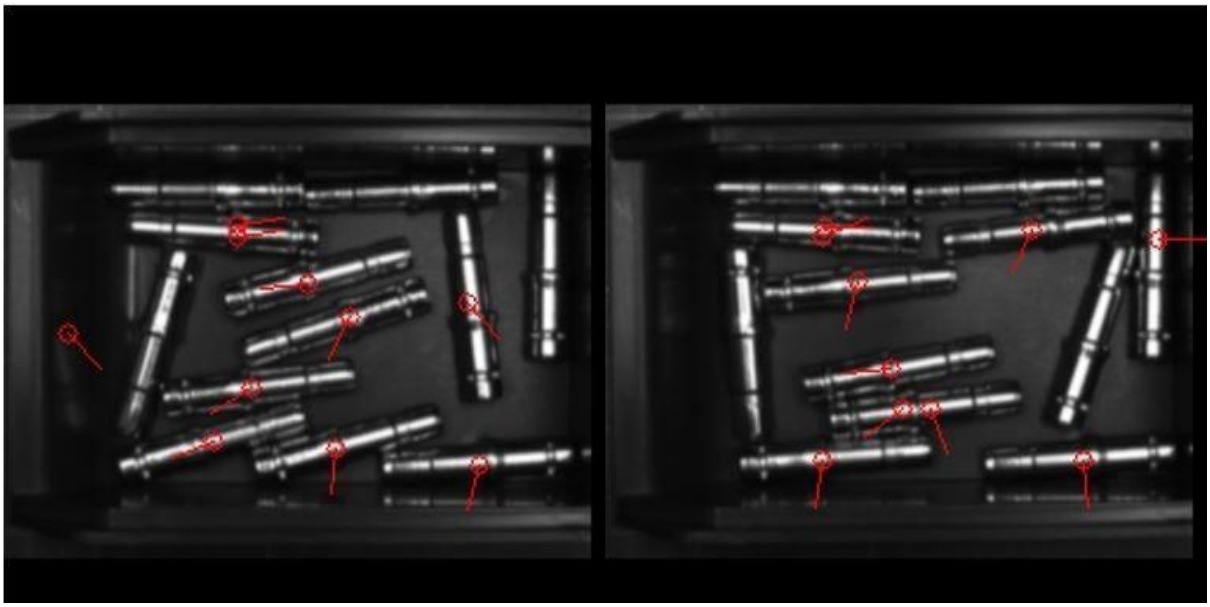


Figure 23 Predictions made on real shaft images

It is clear from above images that the reduction in the number of layers was detrimental to the performance of the model for predictions on real shaft images. Number of false positive as well as the number of false negative predictions appear to have increased.

3.5 Dataset 3(case2) with 19 Convolutional Layers:

The depth of the model was modified to contain 19 convolution layers as the predictions made with the model containing 9 convolutional layers was poor. The trained model was used for inference tasks on real shaft images. Predictions made on synthetic shaft images are given below:

Figure 24 Predictions made on synthetic shaft images

```
208.5413108387703 71.12714304700864 0.79179215 0.72765404 [0.72765404]
100.12341299521962 80.6247001556052 0.698679 0.73088753 [0.73088753]
123.99442616040461 109.33361023115185 0.7223239 0.7290842 [0.7290842]
48.56306169522178 112.80361743416883 0.68000966 0.7304144 [0.7304144]
147.99440404935027 136.66831912554375 0.5771845 0.73189354 [0.73189354]
193.4628394134802 165.3888201969751 0.77839535 0.7283902 [0.7283902]
69.28624184418368 170.31407865007517 0.5930842 0.72853035 [0.72853035]
100.65950311192519 175.14424966360224 0.5694926 0.72021335 [0.72021335]
8 keypoints are found
```

Figure 25 Model prediction for left image 24

It is clear from the above image that the predictions made on synthetic shaft images continued to have a higher recall as well as precision. The predicted orientation angle of shafts was also within acceptable limits.

Below are the predictions made on real shaft images with 19 convolutions:

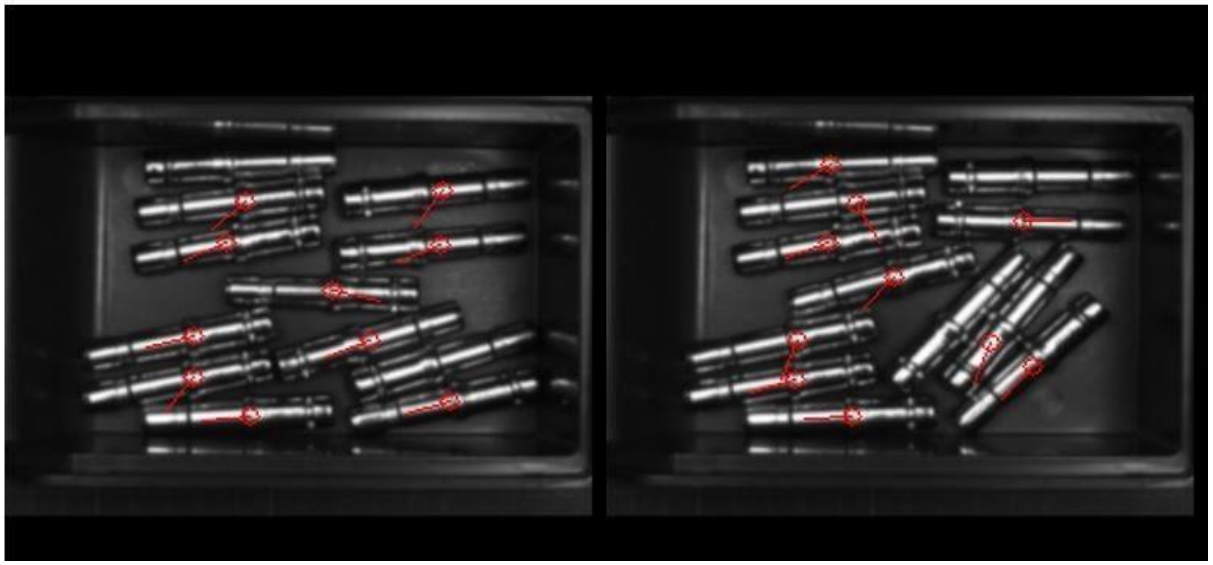


Figure 26 Prediction made on real shaft images

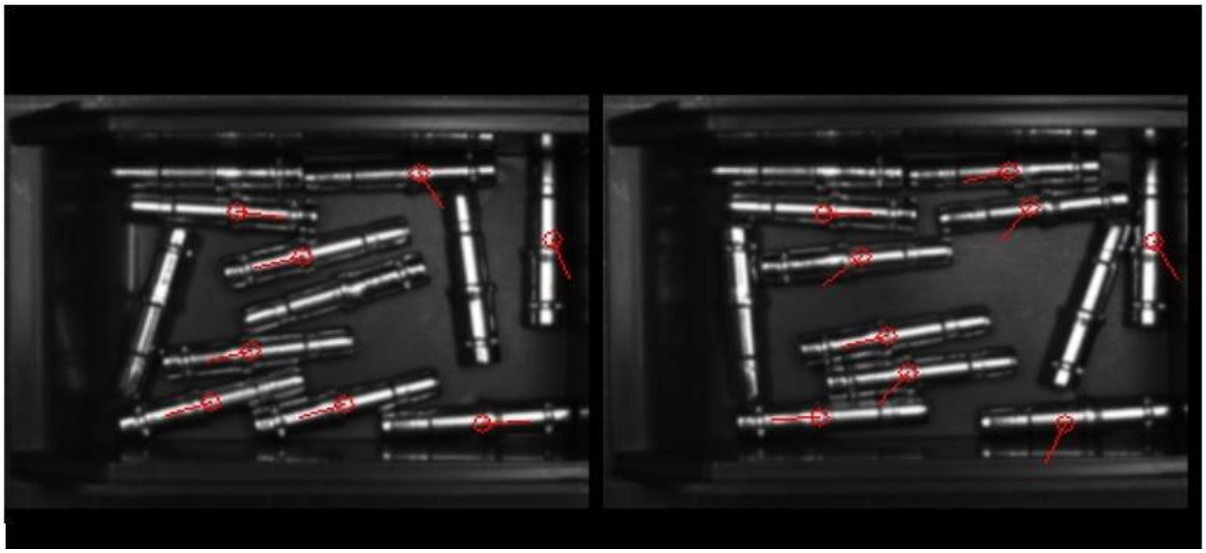


Figure 27 Prediction made on real shaft images

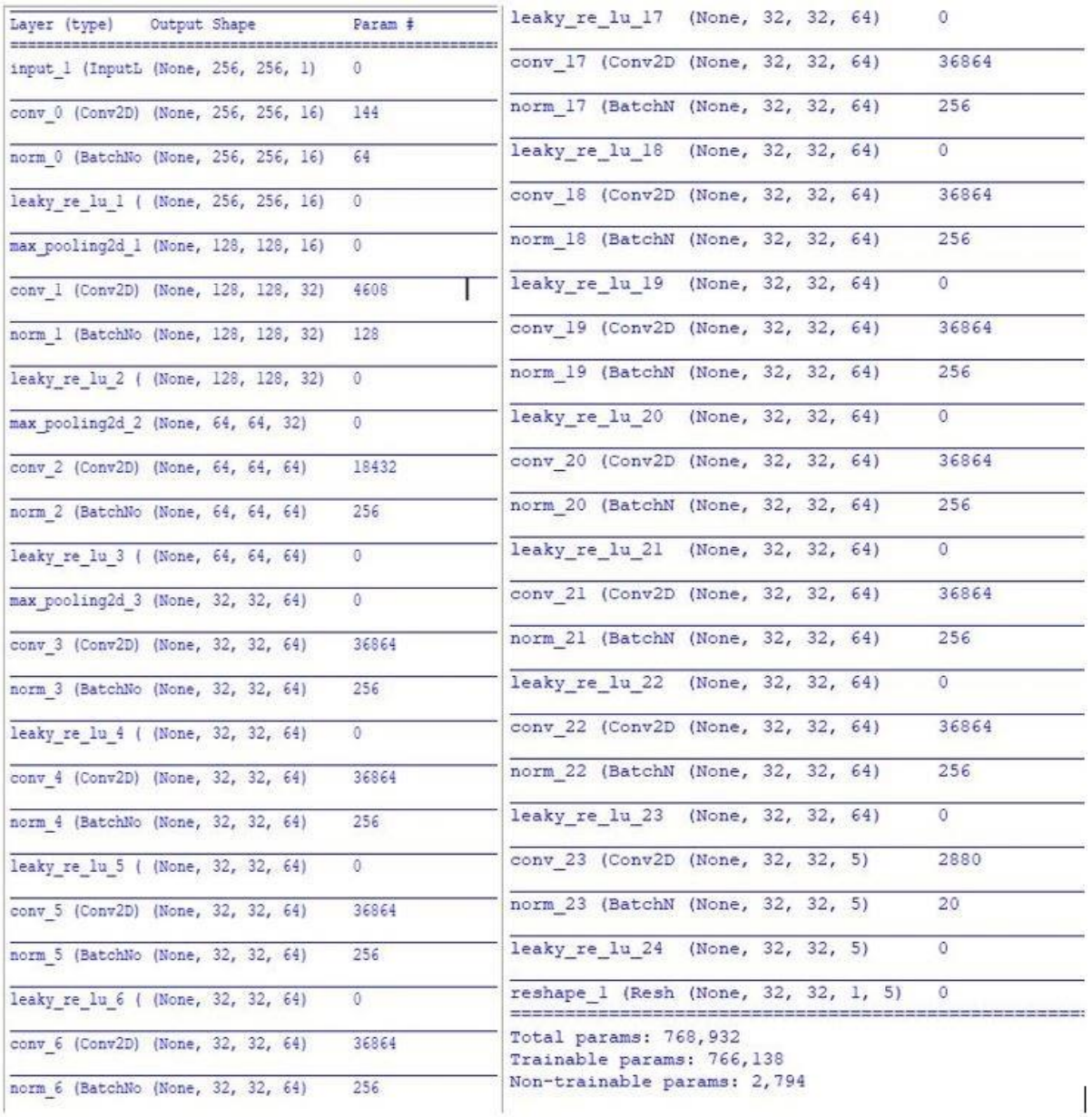
```
186.12553466289134 85.89010110545931 0.6612181 0.71470386 [0.71470386]
102.99543179677974 90.41202983132254 0.7143345 0.71948695 [0.71948695]
94.15457709940858 109.34592042350029 0.7888443 0.7288512 [0.7288512]
185.20804304227073 110.43887785614456 0.78836197 0.7201039 [0.7201039]
139.60676763816355 131.975047344794 0.15248942 0.72893316 [0.72893316]
80.33352446913297 152.40714211802228 0.84008497 0.73107994 [0.73107994]
155.32800644923125 154.17614186960049 0.79740655 0.7302607 [0.7302607]
80.63479334048041 171.80365421108942 0.67143315 0.7263612 [0.7263612]
189.52547115710396 182.96387401961204 0.8255632 0.7257163 [0.7257163]
104.84390872442603 189.65086744199212 0.8800741 0.7240703 [0.7240703]
10 keypoints are found
```

Figure 28 Model prediction for left image 26

As shown above, the predictions made on 19 convolutions did not predict any false negatives nor false positives. Recall of predictions was higher as compared to CNN with 13 convolutions, as most of the shafts were localised. However, the orientation angle predictions of the shafts were not within acceptable limits.

3.6 Dataset 3(case3) with 24 Convolutional Layers:

Five more convolutional layers were appended to the pre-existing model to have model consisting of 24 convolutional layers in total. The model was once again trained for 100 epochs with the same dataset as that of the previous case.



Layer (type)	Output Shape	Param #
input_1 (InputL	(None, 256, 256, 1)	0
conv_0 (Conv2D)	(None, 256, 256, 16)	144
norm_0 (BatchNo	(None, 256, 256, 16)	64
leaky_re_lu_1 (	(None, 256, 256, 16)	0
max_pooling2d_1	(None, 128, 128, 16)	0
conv_1 (Conv2D)	(None, 128, 128, 32)	4608
norm_1 (BatchNo	(None, 128, 128, 32)	128
leaky_re_lu_2 (	(None, 128, 128, 32)	0
max_pooling2d_2	(None, 64, 64, 32)	0
conv_2 (Conv2D)	(None, 64, 64, 64)	18432
norm_2 (BatchNo	(None, 64, 64, 64)	256
leaky_re_lu_3 (	(None, 64, 64, 64)	0
max_pooling2d_3	(None, 32, 32, 64)	0
conv_3 (Conv2D)	(None, 32, 32, 64)	36864
norm_3 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_4 (	(None, 32, 32, 64)	0
conv_4 (Conv2D)	(None, 32, 32, 64)	36864
norm_4 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_5 (	(None, 32, 32, 64)	0
conv_5 (Conv2D)	(None, 32, 32, 64)	36864
norm_5 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_6 (	(None, 32, 32, 64)	0
conv_6 (Conv2D)	(None, 32, 32, 64)	36864
norm_6 (BatchNo	(None, 32, 32, 64)	256
leaky_re_lu_17	(None, 32, 32, 64)	0
conv_17 (Conv2D	(None, 32, 32, 64)	36864
norm_17 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_18	(None, 32, 32, 64)	0
conv_18 (Conv2D	(None, 32, 32, 64)	36864
norm_18 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_19	(None, 32, 32, 64)	0
conv_19 (Conv2D	(None, 32, 32, 64)	36864
norm_19 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_20	(None, 32, 32, 64)	0
conv_20 (Conv2D	(None, 32, 32, 64)	36864
norm_20 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_21	(None, 32, 32, 64)	0
conv_21 (Conv2D	(None, 32, 32, 64)	36864
norm_21 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_22	(None, 32, 32, 64)	0
conv_22 (Conv2D	(None, 32, 32, 64)	36864
norm_22 (BatchN	(None, 32, 32, 64)	256
leaky_re_lu_23	(None, 32, 32, 64)	0
conv_23 (Conv2D	(None, 32, 32, 5)	2880
norm_23 (BatchN	(None, 32, 32, 5)	20
leaky_re_lu_24	(None, 32, 32, 5)	0
reshape_1 (Resh	(None, 32, 32, 1, 5)	0

Total params: 768,932
Trainable params: 766,138
Non-trainable params: 2,794

Figure 29 Model architecture of 24 convolutional layer

Predictions on synthetic Images.

Figure 30 Predictions made on synthetic shaft images

Figure 31 Predictions made on synthetic shaft images

```
165.30623478513616 72.41596466559916 0.75659287 0.7255075 [0.7255075]
67.69305814384009 91.972884236091 0.661171 0.72709143 [0.72709143]
153.79416199586697 135.39883589567796 0.6181435 0.7261344 [0.7261344]
99.97751854947263 137.64508234375865 0.7219556 0.72609204 [0.72609204]
211.94907786499974 139.9695949081379 0.45995784 0.7268912 [0.7268912]
118.14696364761923 147.967433036832 0.6983137 0.7285266 [0.7285266]
171.8125597764215 152.8635854233887 0.6422073 0.7249421 [0.7249421]
85.65654763742315 202.94933509408673 0.9167563 0.7267372 [0.7267372]
8 keypoints are found
```

Figure 32 Model prediction for left ofimage 30

Predictions on real shaft images, up until now, were not up to acceptable limits; mainly characterized by either low recall or low precision in both shaft localization and orientation angle prediction. However, when the number of convolution layers were increased to 24, the inference results for localization and orientation for images of real shafts were significantly better alongside with satisfactory quality of prediction for images of simulated shafts. This is in contrast previous models were the quality of prediction results on synthetic images did not translate well to real images of shafts. Below are the inferences on real shaft images:

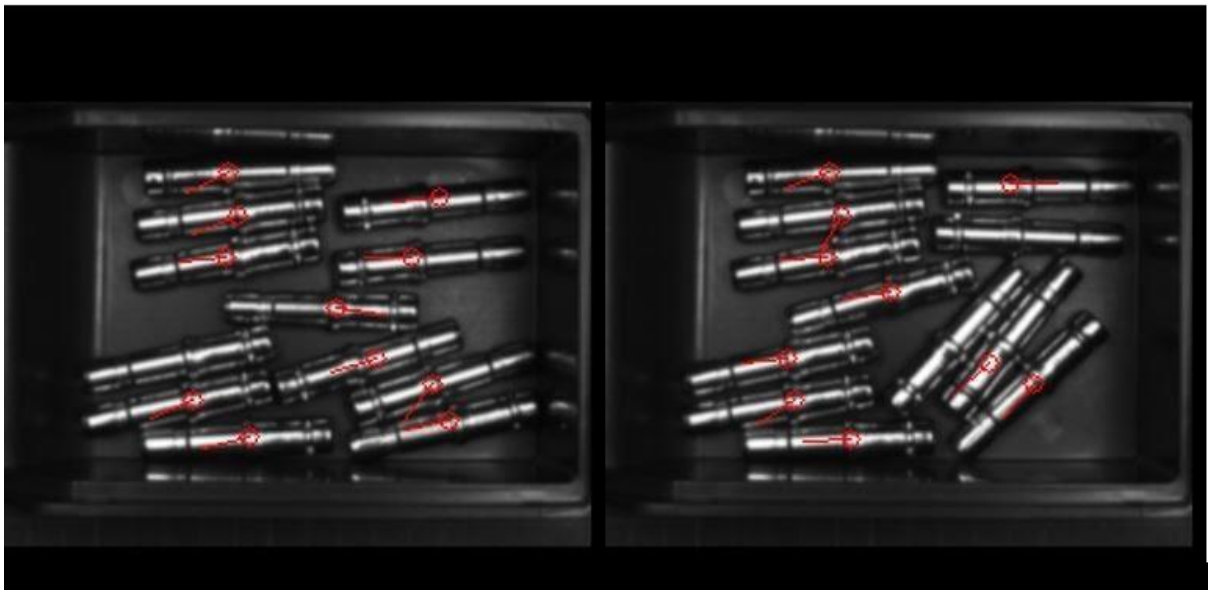


Figure 33 Predictions on real shaft images

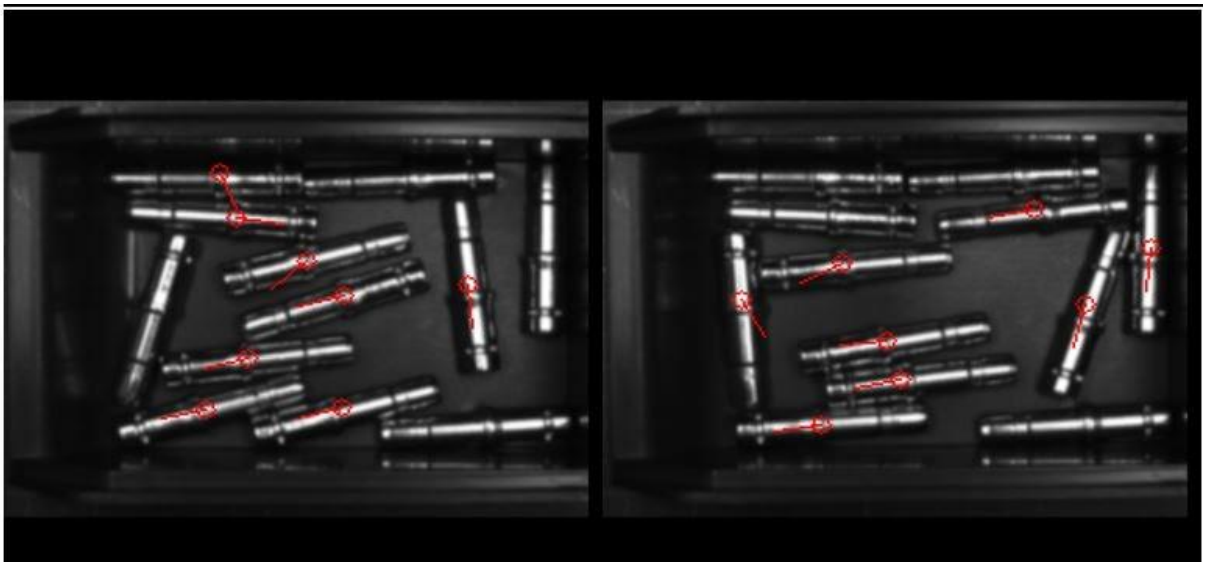


Figure 34 Predictions on real shaft images

```
95.10464237641024 72.89205729893611 0.78159744 0.730592 [0.730592]
185.35656325346528 83.90619160767505 0.8825143 0.726017 [0.726017]
99.45334572394101 91.92474766942597 0.8037215 0.7235766 [0.7235766]
94.52042289122058 109.33772034401638 0.8724911 0.7287726 [0.7287726]
173.66370154226655 109.425376526777 0.8961505 0.7253822 [0.7253822]
141.4478002845308 131.91007867076283 0.13872963 0.7251057 [0.7251057]
158.34162379312443 153.4847397322901 0.82352 0.72641104 [0.72641104]
182.01590526137963 164.14477510323434 0.6626926 0.71904355 [0.71904355]
80.85455652515314 171.93040252665145 0.7984893 0.7274492 [0.7274492]
189.98802864754947 180.7289025962756 0.8498329 0.72468716 [0.72468716]
104.52247820082212 187.9158485024704 0.8472948 0.72694916 [0.72694916]
11 keypoints are found
```

Figure 35 Model prediction for left of image 33

Improvements in key point localisation and orientation angle prediction for real images can be observed in the above the images, alongside with improvements in recall and precision.

3.7 Dataset 3 (case4) with 29 Convolutional Layers:

We have increased the number of convolutional layers from 24 to 29 and trained the model for 100 epochs. The inferences made on synthetic shaft images remained to be identical to that of the previous cases. However, the predictions for shaft localization in real images exhibited deterioration as compared to the previous case i.e., 24 layers.

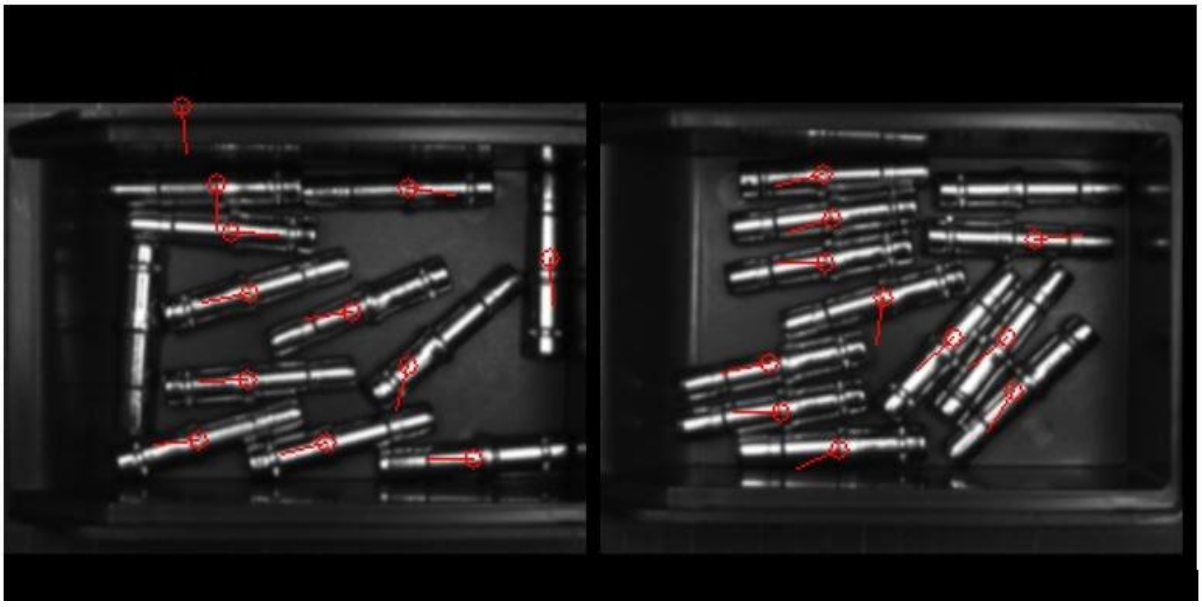


Figure 36 Predictions on real shaft images

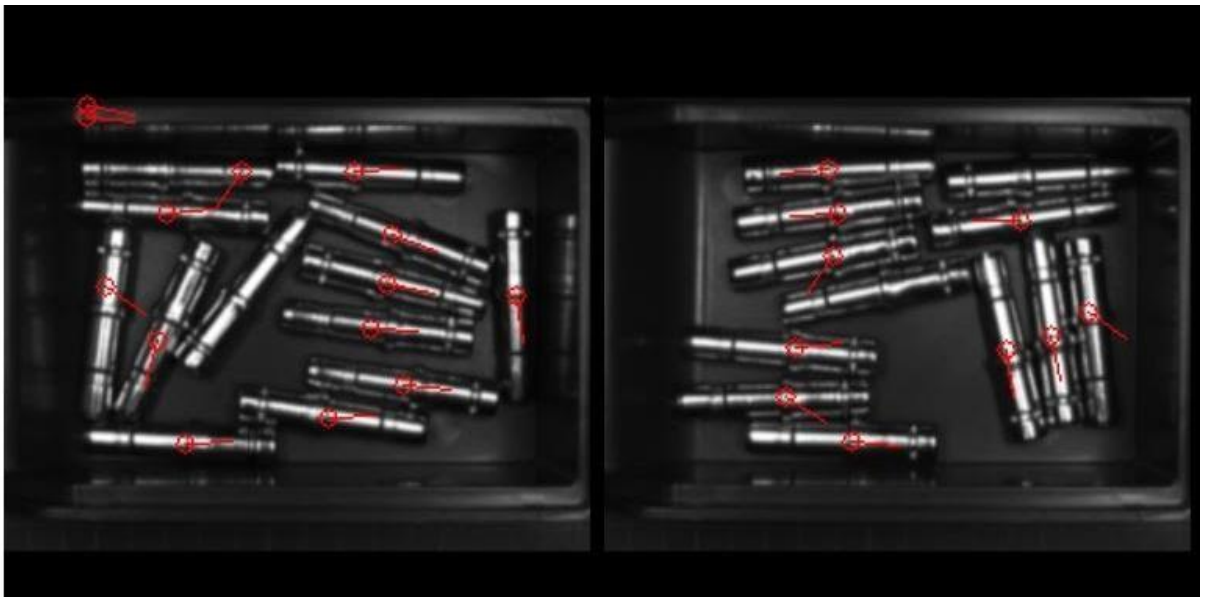


Figure 37 Predictions on real shaft images

```
95.01396637498186 72.63111777995057 0.87472844 0.7443227 [0.7443227]
99.78680860301188 91.54069782104374 0.88292176 0.74456185 [0.74456185]
177.89748981965766 94.21647808647432 0.8960221 0.73923004 [0.73923004]
97.69978919825581 109.70730964724301 0.6589616 0.72321767 [0.72321767]
206.27281745009392 133.59181306713265 0.24806836 0.72488624 [0.72488624]
190.64081731033392 143.24186330248492 0.44988334 0.7404264 [0.7404264]
81.59403492579904 148.83664438152192 0.069334514 0.7213991 [0.7213991]
171.98412024255873 149.2441375687709 0.46669108 0.74097735 [0.74097735]
77.5618730834248 169.90956189111233 0.24506785 0.7441334 [0.7441334]
105.26561544313094 188.43079052094527 0.12304867 0.7371545 [0.7371545]
10 keypoints are found
```

Figure 38 Model prediction for right of image 37

As shown in the above picture, the predictions on real shaft images show some false positive predictions. Even though the recall for above predictions is similar to that of the previous case, the false positive predictions indicate that the algorithm have not trained properly, because of high number of parameters. So, we increased the number of epochs from 100 to 150 and again started training below are the results we got.

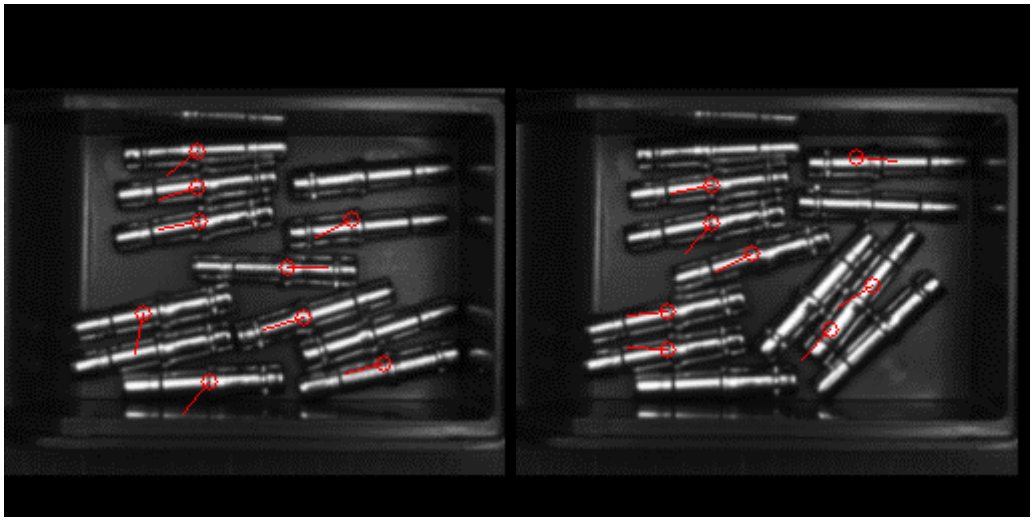


Figure 39 Predictions on real shaft images

It is clear from above images that the Number of false positive as well as the number of false negative predictions appear to have increased. Angle of shaft also not accurately predicted.

Further, as a part of experimentation, the number of layers were increased to 33. However, they exhibited decreased recall as well as precision. Hence, it was concluded that the best prediction results were observed when trained with 24 Convolutional layers. When we increase the convolutional layers to a number more than 24, the accuracy is decreasing.

4.0 FUTURE SCOPE:

Even after increasing the convolutional layers we could not predict all the key point pairs exactly as we got exact results on real image data set. The results we obtained are satisfactory. Further improvements can be made to maximize the detection process.

So, we have hypothesized that this might be because of the padding layer we are adding in the network. Padding is a process in which an additional row and column are formed on either side of the image. The reason we add padding is to keep the image size constant as of input image even after convoluting the image. We have defined the padding in such a way that, it considers the values of the final row and column of the image and considers those values for the newly generated rows and columns. This data due to the padding, which is of no use to the

original image, causes an extra error to the image. So, to minimize this error we created a dataset 4 with 512*512 size images.

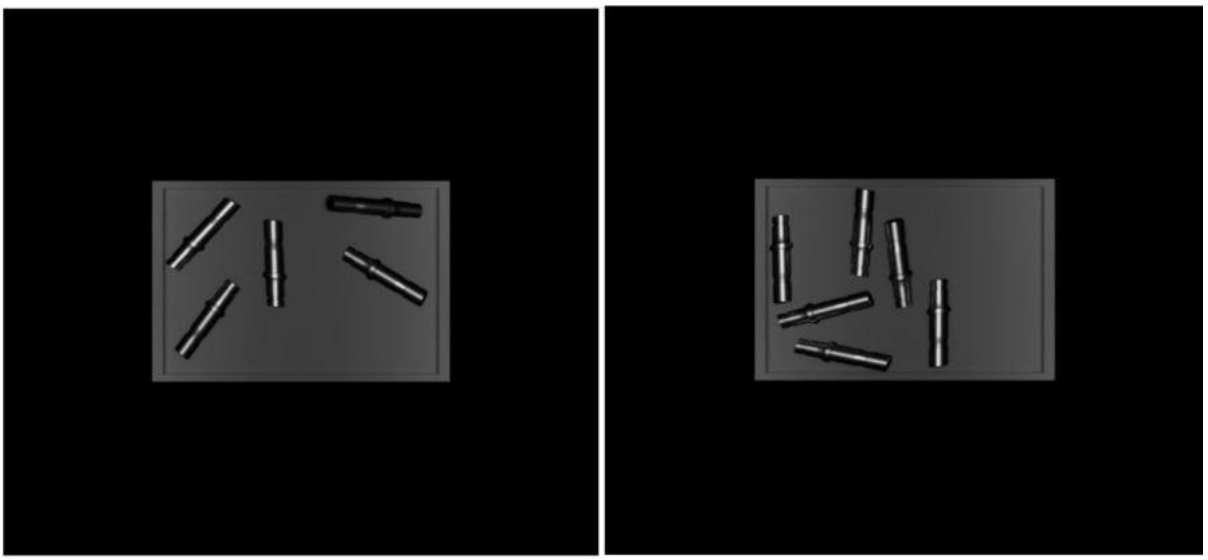


Figure 40 Synthetic images of shafts (Dataset 4)

From the image, we can observe that the tray containing the shafts is placed in the middle and the surrounding is just a black background. This is done in order to minimize the error generated due to the addition of a padding layer while constructing the convolution layers for our network. Since the size of the image is changed, we need to slightly modify our code in order to obtain our output size which is 32 x 32.

The prediction was done on synthetic image, the results were pretty good. There were no false predictions and angle of shaft also accurately predicted. The results improved a bit when compared the predictions made on synthetic images of 24 CNN layers. We also wanted to make predictions on real shaft images, but we do not have real shaft images of 512*512 size. It would be a good step forward if the detection on the real images is better or similar to what we have achieved with our Dataset 3 with 24 convolution layers.

Predictions made on synthetic images.

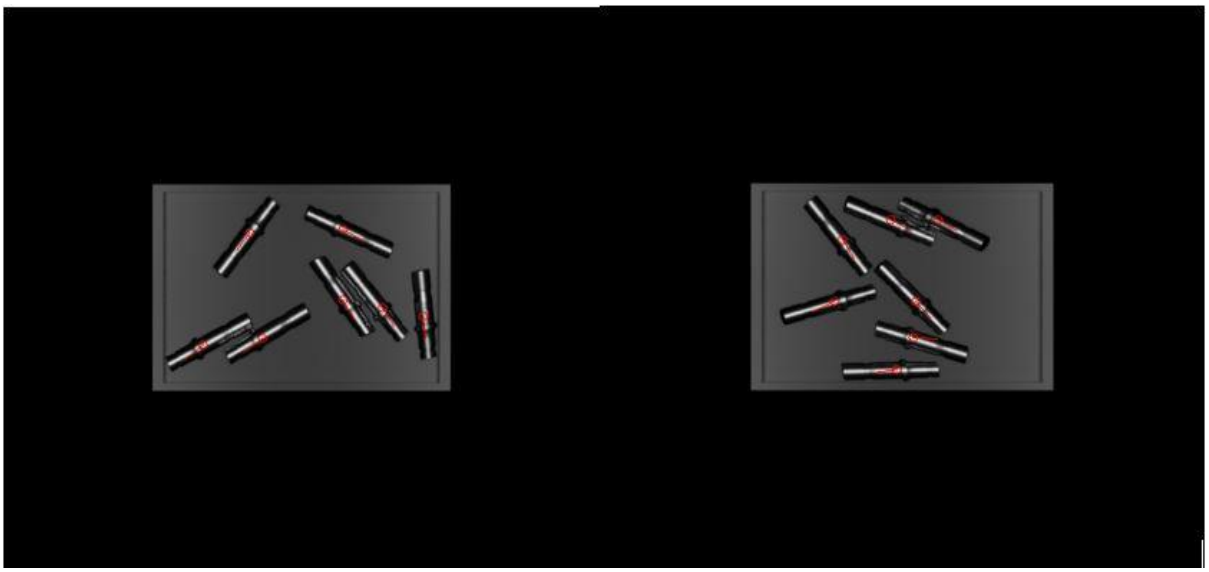


Figure 41 Predictions on synthetic images (512x512) from Dataset 4

Conclusion:

The YOLO-v2 based Convolutional Neural Network (CNN) model was trained with different synthetic shaft image datasets. Synthetic shaft dataset 3 whose resemblance was as close as possible to the real shaft images, when used for training the CNN model, proved to have the highest recall as well as precision in keypoint localization on real shaft images. The model prediction performance was only surpassed by models trained on real images.

The number of convolutional layers of the neural network were changed to study recall and precision in key point localization for every time the depth of the model was changed. It was observed that the CNN had the best results when trained on dataset 3 with 24 convolutional layers.

A higher dimensional dataset (dataset 4) with resolution of 512x512 was trained on to obtain key point predictions on the same dataset images. The trained weights obtained from training dataset 4 were not used to predict key points on real shaft images due to the absence of similar higher dimensional real shaft images. Due to better feature extraction from higher resolution images, dataset 4 could prove to provide higher recall as well as precision.

References:

https://developer.ridgerun.com/wiki/images/f/fc/Tinyyolo_architecture.png

https://d3i71xaburhd42.cloudfront.net/fc7574c3ba94f5d478f5cdbc870e8af354bd62dd/2-Figure2-1.png

https://fairyonice.github.io/Part_4_Object_Detection_with_Yolo_using_VOC_2012_data_loss.html

https://realelectricwizard.wordpress.com/2019/04/12/exploring-the-yolo-evolution/

https://github.com/experiencor/keras-yolo2/blob/master/Yolo%20Step-by-Step.ipynb

https://ac-job-assets-prod.imgix.net/007772482/20181130113050Z/fachhochschule-rosenheim_20.png?auto=format&fill-color=fff&fill=solid&fit=fillmax&h=157&w=300