DPOD: 6D Pose Object Detector and Refiner

Sandip Patel sp3779, Prabhabakar Naik pmn2119

***Abstract*—For this final term project of the course E4040 Neural Network and Deep Learning, We chose to implement the estimation of 6D pose of the objects using Deep Learning Models. With growing automation trends need for object pose detection in order to interact with the surrounding has become essential. This inspired us to research and implement the solution to object pose detection using Deep Learning. Hence, the purpose of this project is to implement the "DPOD: 6D Pose Object Detector and Refiner" as presented by Sergey Zakharov et al [ZSI19].The original work estimates dense multi-class 2D-3D correspondence maps between an input image and available 3D models. Given the correspondences, a 6DoF pose is computed via PnP and RANSAC. An additional RGB pose refinement was used on top of the estimated pose to improve on the prediction using custom Deep Learning architecture. The main technical challenge was to understand this big end to end architecture, implement it with limited resource both in terms of hardware and time. This was overcome by considering the using the Google computing cloud Platform and selectively using data from the dataset to limit the training time. After careful study and understanding the architecture of the end to end Solution for 6D pose detection we were able to replicate the work by original other with fair accuracy and results.**

I. Introduction

With increasing robotic presence in industrial and consumer market, robots are expected to not just perceive but to interact with the world. Hence, knowing the full 6D pose of the world objects with respect to robot is essential. Object detection is now a well studied field in deep learning and computer vision with algorithms like YOLO[Red+15] and SSD[Liu+15] achieving human level performance in real time. Although estimating 6D pose is easily solved with the point cloud or depth images, solving this problem form RGB images is still an active field of research.

Recently, algorithms like SSD6D[Keh+17], YOLO6D[TSF17] have achieved reasonable accuracy on the LineMOD and OCCLUSION datasets. YOLO6D[TSF17] is trained on real data while SSD6D[Keh+17] is trained on synthetic renderings of the datasets from different viewpoints. Approach in this paper presented differs by the fact that it create correspondence map that that provide relation between each image pixels and 3D model vertices. This gives large number of pixel-wise correspondences allowing better pose estimation, for example YOLO6D[TSF17] uses just 9 regressed virtual points of the object's bounding box.

Usually pose estimation with dense correspondences (large number of image pixels relation with vertices of 3D models) requires enormous annotations efforts for ground truth, like the methods of Gueler et. al.[GNK18] and Taylor et al.[Tay+12] which estimate dense correspondences between the human body model and the humans in the image. But this approach is annotation-free and only requires generating arbitrary texture maps of the objects by spherical projections. We used Point and Perspective(PnP)[Zha00]+RANSAC on the Dense correspondence maps generated from the Deep network to estimate the pose of the objects .

Apart from annotation free high number of pixel to 3D vertices, our approach has another added quality that is of Pose Refinement Network (PRN). PRN takes initial pose estimated by the DPOD network and enhances it. Using the PRN we can improve on the already estimated high quality pose. We will walk through the original paper first and then to our implementation of the concept based on the original paper.

II. Summary of the Original Paper

In this section we will discuss the methodology and the architecture used by the authors of the paper. We will walk through from starting from the data-preparation, followed by neural network architecture to the pose estimation and pose refinement.

A. Methodology of the Original Paper

First we will walk through some of the key process and then will move to the architecture.

1) Data Preparation: Most of the RGB-based detectors can be divided in two groups based on the type of data they are trained on; synthetic and real. For synthetic training data.The first step is to render different pose of the objects from the given 3D models and in plane camera rotations are added Then, for each of the camera poses, an object is rendered on a black back-ground and both RGB and depth channels are stored. while in case of real training data, poses are selected such that the relative orientation between them to ensure selected pose cover all the sides of the object. For our model we used both synthetic and real training data.

2) Correspondence mapping: Once the training data is ready and prepared, Deep learning architecture generates the texture maps from the 3D model of the object. This texture will later be used for correspondence mapping, i.e to build relation between image pixels and 3D model vertices.

To be able to learn dense 2D-3D correspondences, each model of the dataset is textured with a correspondence map as shown in figure 1. A correspondence map is a 2-channel image with values ranging from 0 to 255 as usual. one of the channel image looks like the one in figure 2. Objects are textured using either simple spherical or cylindrical projections. Once textured, we get a bijective mapping between the model's vertices and pixels on the correspondence map. This provide an easy relation of where does the point in image lies on the 3D model. All we have to do is just look at the pixel colour in the correspondence map and that instantaneously

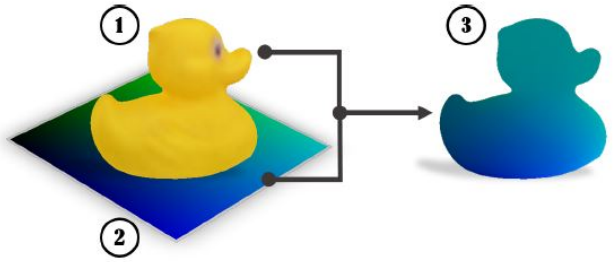

Figure 1. Process of texture mapping a 3D model.

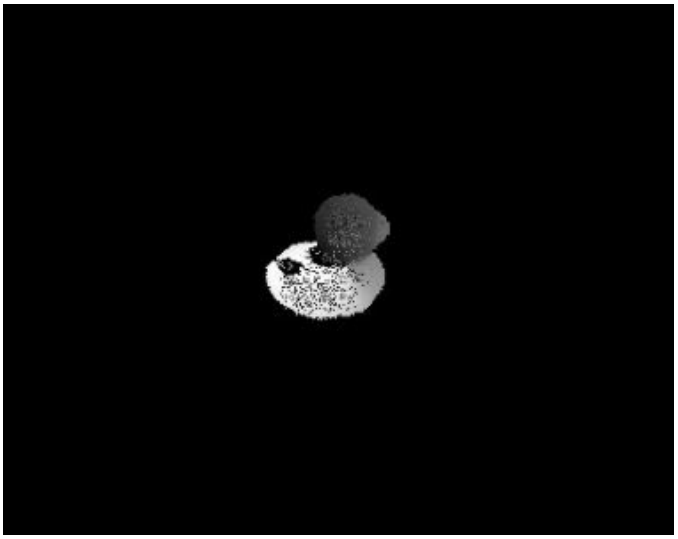

Figure 2. UV map for 'duck' class

relates to the position on the model surface. Once we have the predicted correspondence map we can estimate the object's pose with respect to the camera using PnP and RANSAC. correspondence patches for each RBG patch are stored.

3) Data Augmentation for Object detection and Pose Estimation: Now we have the RGB patch and correspondence patch of the object for which we have to estimate the pose. Generated patches (real or synthetic) are rendered on top of images from MS COCO dataset[Lin+14] producing training images containing multiple objects. It is an important step, which ensures that the detector generalizes to different backgrounds and prevents it from overfitting to backgrounds seen during training. Moreover, it forces the network to learn the model's features needed for pose estimation rather than to learn contextual features which might not be present in images when the scene changes. This step is performed no matter whether the training is being done with synthetic or real patches.

RGB image is additionally augmented with random brightness and noise. Additionally we also created ID masks that identifies each pixel as belonging to one the class objects in the RGB image.

B. Network Architecture and Data Pipeline

Authors of the original papers divided the network into two blocks; Correspondence block and Pose block as seen in the figure 3.

1) The correspondence Block: The correspondence block consists of an encoder-decoder CNN with three decoder heads. Each decoder head does a specific job. 2 decoder regress the two channels of the correspondence maps and third one regress the pixel wise ID mask of the object of the interest. output is of the same size as the input image. Network is optimized based on the composite loss of all three decoder heads with loss of each head is weighted and weights are hyper parameters.

2) The Pose Block: Given the ID Masks and the 2D locations where correspondence maps maps each point, we can estimate the complete 6D pose of the object using Perspective-n-point(PnP) and RANSAC to make the camera pose prediction robust.

Finally a Pose refinement is done using the Deep Learning architecture as well. Pairs of images containing the object in the current (searched) pose and in the predicted pose is given to the network. Figure 3 shows the architecture of the refinement block proposed in the original paper. In order to be able to benefit from the network weights pre-trained on ImageNet, the network has two parallel input branches, each composed of the first five ResNet layers. These layers are initialized from the pre-trained network. One branch receives an input image patch (E_{1_1}), while the other (E_{1_2}) one extracts features from the rendering of the object in the predicted pose. Then features f_r and f_s from these two networks are subtracted and fed into the next ResNet block (E_2) producing the feature vector f . The network ends with three separate output heads: one for regressing the rotation, one for regressing the translation in X and Y directions, and one for regressing the translation in Z direction. We opted for three separate heads as the scale of their outputs is different. Each head is implemented as two fully connected layers.

C. Key Results of the Original Paper

Authors of the original paper evaluated their algorithms with respect to many aspects of the objective. Algorithm is evaluated against other state of the art RGB detector algorithms like SSD6D, YOLO6D and DeepIM which are trained on synthetic and Real data for metrics like run time, pose estimation and detection performance.

Authors used ADD score as the metric to evaluate how well the detection model predicts the pose of the object w.r.t to the ground truth pose. ADD score is given by the following equation:

$$m = \underset{x_2 \in \mu}{avg} \underset{x_1 \in \mu}{min} \left\| (Rx_1 + t) - (\hat{R}x_2 + \hat{t}) \right\|_2 \quad (1)$$

where μ is a set of vertices of a particular model, R and t are the rotation and translation of a ground truth transformation whereas $\hat{R}$ and $\hat{t}$ correspond to those of an estimated transformation. Authors considered the pose to be correct if ADD score is smaller than the 10% of the model's diameter and

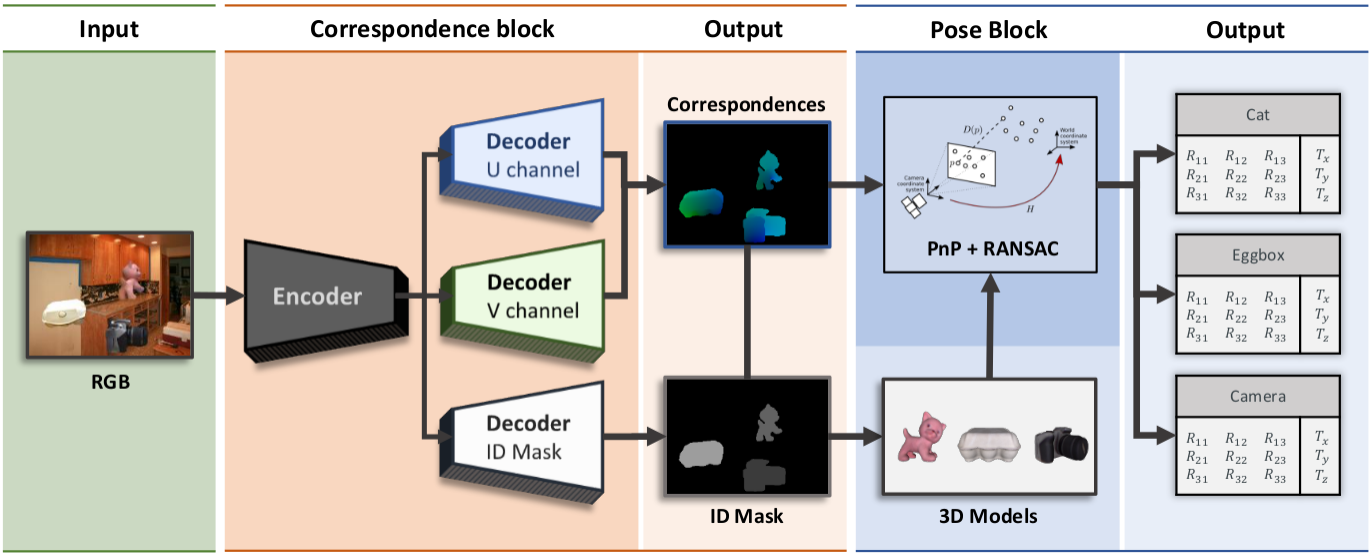

Figure 3. Pipeline description: Given an input RGB image, the correspondence block, featuring an encoder-decoder neural network, regresses the object ID mask and the correspondence map. The latter one provides us with explicit 2D-3D correspondences, whereas the ID mask estimates which correspondences should be taken for each detected object. The respective 6D poses are then efficiently computed by the pose block based on PnP+RANSAC.

Train data	Synthetic			+ Refinement		Real				+ Refinement	
Object	SSD6D [15]	AAE [31]	Ours	SSD6D [22]	Ours	YOLO6D [33]	PoseCNN [34]	PVNet [25]	Ours	DeepIM [18]	Ours
Ape	2.6	3.96	**37.22**	-	55.23	21.62	-	43.62	**53.28**	77.0	**87.73**
Benchvise	15.1	20.92	**66.76**	-	72.69	81.80	-	**99.90**	95.34	97.5	**98.45**
Cam	6.1	**30.47**	24.22	-	34.76	36.57	-	86.86	**90.36**	93.5	**96.07**
Can	27.3	35.87	**52.57**	-	83.59	68.80	-	**95.47**	94.10	96.5	**99.71**
Cat	9.3	17.90	**32.36**	-	65.10	41.82	-	**79.34**	60.38	82.1	**94.71**
Driller	12.0	23.99	**66.60**	-	73.32	63.51	-	96.43	**97.72**	95.0	**98.80**
Duck	1.3	4.86	**26.12**	-	50.04	27.23	-	52.58	**66.01**	77.7	**86.29**
Eggbox	2.8	**81.01**	73.35	-	89.05	69.58	-	99.15	**99.72**	97.1	**99.91**
Glue	3.4	45.49	**74.96**	-	84.37	80.02	-	**95.66**	93.83	**99.4**	96.82
Holepuncher	3.1	17.60	**24.50**	-	35.35	42.63	-	**81.92**	65.83	52.8	**86.87**
Iron	14.6	32.03	**85.02**	-	98.78	74.97	-	98.88	**99.80**	98.3	**100**
Lamp	11.4	**60.47**	57.26	-	74.27	71.11	-	**99.33**	88.11	**97.5**	96.84
Phone	9.7	**33.79**	29.08	-	46.98	47.74	-	**92.41**	74.24	87.7	**94.69**
Mean	9.1	28.65	**50**	34.1	**66.43**	55.95	62.7	**86.27**	82.98	88.6	**95.15**

Figure 4. Results of single object pose detection from original results

accuracy of the poss estimation is reported as the percentage of the correctly estimated poses.

Authors did single object pose detection as well as multiple object pose estimation. Results of single object pose detection is shown in Figure 4 and results of multiple object pose detection is shown in Figure 6.

Table in the figure 4 shows the results of the pose estimation on the LineMOD dataset with single object detection in the frame. it is compared with the other algorithms with or without refinement. The table reports the percentages of correctly estimated poses w.r.t. the ADD score. Conventionally, a pose is considered correct if ADD is smaller than the 10% of the model's diameter. Performance evaluation of the proposed detector in cases when the number of objects to detect increases and when severe occlusions are present was conducted on the OCCLUSION dataset [2] and is shown in Figure 6. Accuracy of object detection on the OCCLUSION dataset is conventionally reported in terms of mean average precision (mAP).

III. Methodology of Student's Paper

A. Objectives and Technical Challenges

Our objective was to recreate and implement the 6D pose estimation model as laid out in the original paper with minor modification where required in order to make the model lighter and easy to implement within the bounds of resources and bounds. We used the exact same model structure and data flow to regenerate the results as closely as possible.

Given the scale of the project, it was clear that we had to compromise on amount of data processed and number of epochs run to suit out time and hardware limits. Hence,

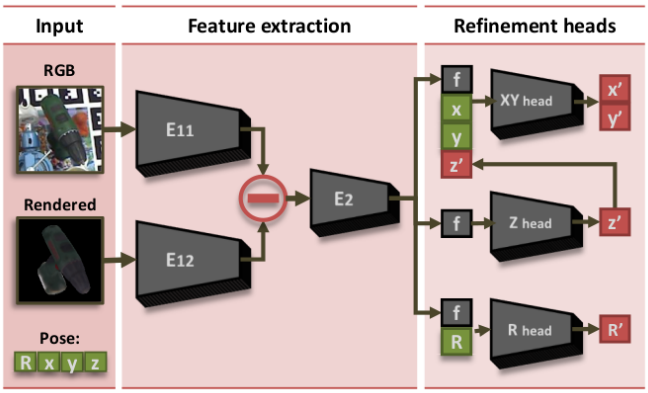

Figure 5. Refinement architecture: The network predicts a refined pose given an initial pose proposal. Crops of the real image and the rendering are fed into two parallel branches. The difference of the computed feature tensors is used to estimate the refined pose.

Method	SSD6D [15]	YOLO6D [33]	Brachmann [3]	Ours
mAP	0.38	0.48	0.51	0.48

Figure 6. Results on multi object pose detection results from original paper

Only real data is used to decrease the rendering and memory overhead. We also decresed the batch size to fit the GPU memory at our personal workstation. As a Result, we had to migrate it based on the GCP to do the heavy lifting. Although the GCP scale the machine specs but at the cost of money. Hence we had to decrease the number of the epoch and training time ,as we need to run the program many times due for optimizing the network.

B. Problem formulation and Design Description

1) Problem Formulation: Putting the Problem formulation in simple words it can be stated estimating the translation and rotation of the object coordinate axis with respect to the camera coordinate axis from the given RGB image data using end-to-end deep learning architecture.

In the analytical world there exist techniques that can calculate the pose from the RGB image given the camera intrinsic matrix and the relation between few points on the 2D object image to corresponding points on the surface of its 3D model. However, it is not scalable to calculate the pose of every image in the image or the frame in the video in real time. But its is a well understood by now in the Deep Learning fraternity that Deep Learning can essentially learn any function than why not the one that learns to do the job we just mentioned above in a scalable manner. And, here we are at the solution.

2) Design Description: Starting with correspondence block. Our correspondence block does the heavy lifting of finding relation (correspondence) between large number of points on 2D image of the object with the points on the surface of the 3D model. This is done with the help of UV texture mapping. Complete process of finding the correspondences goes as follow. A 2D UV texture map is taken and 3D model is unwrapped over it to make it flat or say 2D. And model is wrapped back again along with the texture which is transferred to it. since now each point on the surface model is having a texture (of any type) one can easily locate the position of any pixel in 2D on the surface of the 3D model by matching the texture. Hence, we get the large number of correspondences. Once we have correspondences ready we can used either of the PnP or RANSAC to estimate the pose.

PnP and RANSAC are the state of the art methods in Computer vision to estimate the object pose given the correspondences and camera parameters as we have mentioned earlier. we used both PnP and RANSAC to estimate pose as accurate as we can. Here we will explain the concept behind PnP.

3) Defination of PnP: Given a set of n 3D points in a world reference frame and their corresponding 2D image projections as well as the calibrated intrinsic camera parameters, determine the 6 DOF pose of the camera in the form of its rotation and translation with respect to the world. This follows the perspective project model for cameras:

$$s\, p_c = K\, [R|T]\, p_w \tag{2}$$

where $p_w = [\, x\ y\ z\ 1]^T$ is the homogeneous coordinates of the point on the 3D model surface w.r.t to world frame, $p_c = [\, u\ v\ 1]^T$ is the homogeneous coordinates of the same point on the 2D image, K is the matrix of intrinsic camera parameters, s is the scale factor of the image point. Solving the equation we get the $[R|T]$ matrix which contains the Rotation (R) and Translation (T) information for the camera w.r.t to world frame or say object frame if we choose coordinate frame wisely. the PnP problem is in its minimal form of P3P and can be solved with three point correspondences. However, with just three point correspondences, P3P yields up to four real, geometrically feasible solutions. As the number of points increase the accuracy and solution converges. PnP is prone to errors if there are outliers in the set of point correspondences. Thus, RANSAC can be used in conjunction with existing solutions to make the final solution for the camera pose more robust to outliers and that's precisely what we have done.

Coming to the next part of our Design which is a Pose refinement Block, what essentially we are doing here is basically taking the RGB image and the 3D model rendered from the predicted pose and refine the pose based on the difference between the two. Figure 5 shows the architecture of the refinement block. Two input blocks extracts feature from the RGB ground truth and rendered image from the predicted pose which are subtracted to find the difference between the two. the output function is then feed into separate channels along with the predicted R, X, Y, Z to refine X, Y and Z and R (Rotation) respectively.

This is the exactly what we tried to build and design our program around.

Technical challenges:

IV. Implementation

This section discusses the implementation of the DPOD algorithm in greater detail. DPOD uses U-Net for estimating the correspondence and pre-trained ResNet for pose refinement. The details for these networks, and the student implementation, are delineated in the following subsections.

A. Deep Learning network

1) Correspondence Mapping: Aim of the correspondence step is to learn a mapping from an RGB image to the $\{U, V, Label\}$ space, using intrinsic camera matrix and point cloud data. This mapping is learned using the UNet [RFB15]. The loss is defined as a custom function as the summation of cross-entropy losses calculated between the semantic label, U-mapping and V-mapping. Students tried NLL, L_2, Categorical cross-entropy, but cross-entropy proved to be performing well. ADAM was used as the optimization strategy. The summary of the architecture is shown in the pdf at this link and the computational graphic representation is shown using this link.

2) Pose Refinement: After estimating the pose of the object along with its semantic label, further refinement in the pose is carried out by the *Pose Refinement Network*. The network is a an adaptation of pre-trained ResNet [He+15], takes a 224 x 224, normalized image and predicts X, Y; Z and Rotation for the corresponding object. The *Matching Loss* is calculated as depicted in Algorithm 1. ADAM optimizer is used as in Correspondence Network. The network summary is shown in pdf link.

Algorithm 1: Matching Loss

Result: Total Loss
initialization $Loss \leftarrow 0$;
while *images in batch* **do**
  $TP \leftarrow$ True Pose ;
  $PP \leftarrow$ Predicted Pose ;
  $PtCloud \leftarrow$ Point Cloud ;
  $target :=$ PtCloud @ TP ;
  $output :=$ PtCloud @ PP ;
  $loss := |output - target|$;
  if $loss < Threshold$ **then**
    $TotalLoss = TotalLoss + loss$;
  else
    $TotalLoss = TotalLoss + 100$;
  end
end

B. Software Design

The entire framework is implemented in the Python and the user interface in iPython notebook environment, where the directory structure created as well. Intrinsic camera matrix and class labels are hard-coded in the program and are a characteristic of the dataset. The $GroundTruth$ masks and $UVmapping$ dictionary are implemented as shown in Algorithm 2

Algorithm 2: Ground Truth Mask and UV mapping

Result: Label_mask, U_mask, V_mask, dictionary
initialization $Label\_mask, U\_mask, V\_mask \leftarrow 0$;
initialization $dictionary = \{\}$;
while *images in training_set* **do**
  $PtCloud \leftarrow$ Point Cloud ;
  $Rigid\_trans := [Rot|Trans]$;
  $2DTransformation :=$
  $intrinsicCameraMatrix@Rigid\_trans$;
  $[x, y] := 2DTransformation$;
  $Label\_mask[x, y] \leftarrow class_label$;
  $[unit[0], unit[1], unit[2]] := \frac{PtCloud}{|PtCloud|}$;
  $U := 0.5 + \frac{atan2(unit[2], unit[0])}{2\pi}$;
  $V := 0.5 - \frac{asin(unit[1])}{2\pi}$;
  $U\_mask[x, y] \leftarrow U$;
  $V\_mask[x, y] \leftarrow V$;
  $key = (U, V)$;
  if $key \notin dictionary$;
  then
    $dictionary[key] = PtCloud$;
  end
end

Based on the ground truth UV mapping and $\{U, V, Label\}$ masks, the *Correspondence network* is trained. This trained network is further used to create the preliminary pose estimations. These pose estimations are used as an input to the *Pose Refinement network*. The algorithm for the *Pose Refinement* is depicted in Algorithm 3. Two separate networks helped students in debugging and observe intermediate results.

Pose Refinement network estimates the X, Y; Z and Rotational pose for the objects.

The trained *Correspondence network* and *Pose Refinement network* are used in evaluation mode to predict poses on the test data set. The entire model accuracy is encoded as *ADD_score*. Student model achieved an accuracy of about 81% and mean evaluation time of 0.99576 sec per instance.

The ground truth pose and the pose predicted by the student model are visualized for inspection of the model performance.

V. Results

A. Project Results

After training our Neural network model over Google Cloud Platform, We were able to train the pose detection and pose refinement model to achieve satisfactory and comparable results. We used the same ADD Score metric as explained in the original paper. Our results are only available for the training done on real images and not synthetic images. From the figure 4, clearly model in the original paper performed exceptional well with ADD Score of 95.14 after the refinement is done. Our correspondence block trained for **number of epochs** epochs and pose refinement model for **number of epochs** epochs achieve the comparable ADD score of $ADDscore$.

Algorithm 3: Pose Refinement

Result: X, Y; Z; Rotation; unPredicted
initialization $unPredicted \leftarrow 0$;
while $epochs < maxEpoch$ **do**
  while *image in training Set* **do**
    $Label, U, V \leftarrow UV\_Map(image)$;
    $PosePred \leftarrow$
     $Correspondence\_network(image).predict()$;
    $ID = image.class()$;
    $[X, Y, Z, Rot] := PosePred$;
    if $[X, Y, Z, Rot] = \phi$ **then**
      $unPredicted+ = 1$
    else
      $X_{min} = min(X); X_{max} = max(X)$;
      $Y_{min} = min(Y); Y_{max} = max(Y)$;
      $Object := image[X_{min} : X_{max}, Y_{min} :$
       $Y_{max}]$;
      $PoseRef := PRN(Object, PredPose)$;
      $loss = MLoss(PoseGT, PoseRef)$;
      $PRN.step(loss)$
    end
  end
end

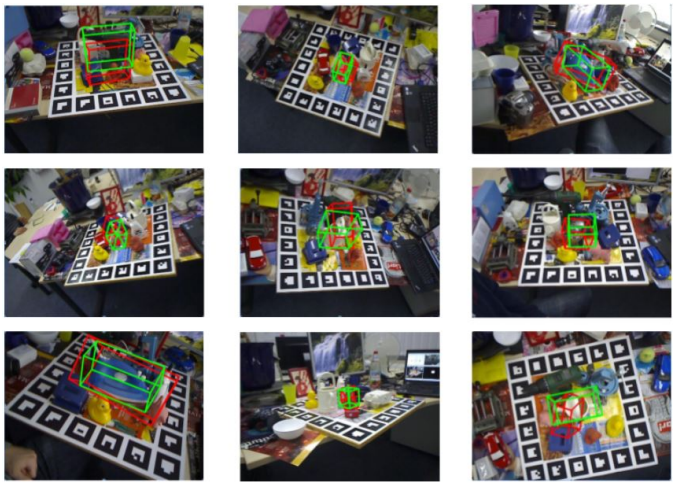

Figure 7. Here student's images are presented with bounding cuboids to compare predicted and ground truth poses.Ground truth are in green and predicted are in red.

Our lower ADD was expected and is justified as we had to compromise on many things from original paper due to resource and time limitation. We decided to give up on data augmentation, searching the hyper parameter space for better convergence, and running small number of epochs to save time and complete the training within practical time period. This gave us time to iterate and experiment with the model architecture.

Our Learning curve containing the training and Evaluation loss for the pose estimation block can be found in the figure 9

Figure 8. Here Original paper's images are presented with bounding cuboids to compare predicted and ground truth poses.Ground truth are in green and predicted are in blue.

below. To understand how well training is done and visually understand, how our results differ from the original paper, we have attached results with RGB images in Figure 7. It shows the estimated pose of the object after refinement using a bounding cuboid with the ground truth. Figure 8 shows the results of original paper in similar fashion.

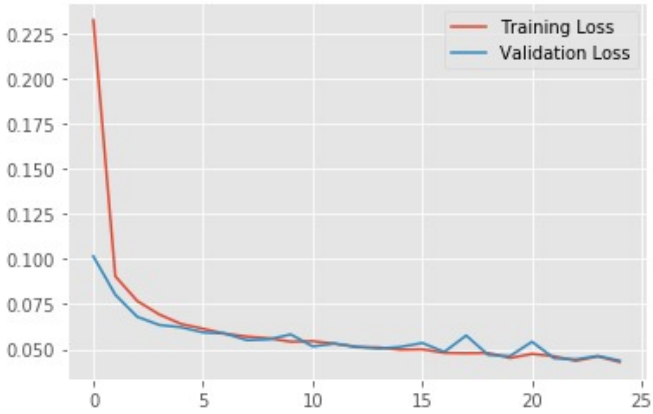

Figure 9. Training Loss and Validation loss: The plot show that the training and validation loss is decreasing and even after 23 epochs (20 hours) its still decreasing . However due to time and resource constraint we had to limit the number of epochs

VI. Discussion and Insights

There were lot of learning's along the Journey of the course and this project let to even more generalised understanding

of Deep Learning. Understanding of Neural Networks as Function estimator was gained. More elaborately speaking following insights were gained along the journey.

Original paper did excellent job of estimating pose with no difference at all visually as seen from Figure 8 however hardware at hand with limit memory allocation and often happening memory overflow prevented setting reasonably large batch size and hence, not leading to even less validation loss.

Around 13000 images and added same amount of rendering when added to the memory, data augmentation does not fall in to the reasonable scope we were planning to work with. Moreover synthetic images were dropped to reduce memory overheads.

Each training cycle takes up to 8 hours for correspondence block and equal amount of time for refinement block. This leaves little room for hyper parameter space search for better optimized results.

The model was trained for upto 25 epochs. As seen in Fig. 9, the training and validation losses are still decreasing. Training for larger times, training and validation losses can still be minimized thus, improving prediction accuracy.

Around 20% of the available 13000 images were used as the training set due to large step time. Each epoch took around 1340 sec to complete. Furthermore, Both *Correspondence_network* and *Pose Refinement Network* have large number of trainable parameters and thus large representation capabilities. Thus, model accuracy can be further improved by training the model on larger training samples. Additionally, pose estimation is a fairly high dimensional problem with each image in LineMOD data set being $3 * 360 * 480$. Reducing the image size to improve the step time severely affected the overall model performance.

VII. Conclusion

The method proposed in this paper (DPOD) regress multi-class object masks and pixel wise correspondence with the vertices of the 3D model performs better than other methods that regress projections of the object's bounding boxes and treat pose estimation as the discrete classification problem. dense correspondence leads to robust and excellent estimation of the pose of the object and the algorithms outperforms the many state of the art algorithm.

References

[Zha00] Z. Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[Tay+12] Jonathan Taylor et al. "The Vitruvian Manifold: Inferring Dense Correspondences for One-Shot Human Pose Estimation". In: *Proc. CVPR*. IEEE, June 2012. URL: https://www.microsoft.com/en-us/research/publication/the-vitruvian-manifold-inferring-dense-correspondences-for-one-shot-human-pose-estimation/.

[Lin+14] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *ECCV*. European Conference on Computer Vision, Sept. 2014. URL: https://www.microsoft.com/en-us/research/publication/microsoft-coco-common-objects-in-context/.

[He+15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015).

[Liu+15] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[Red+15] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[Keh+17] Wadim Kehl et al. "SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again". In: *CoRR* abs/1711.10006 (2017). arXiv: 1711.10006. URL: http://arxiv.org/abs/1711.10006.

[TSF17] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. "Real-Time Seamless Single Shot 6D Object Pose Prediction". In: *CoRR* abs/1711.08848 (2017). arXiv: 1711.08848. URL: http://arxiv.org/abs/1711.08848.

[GNK18] Riza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. "DensePose: Dense Human Pose Estimation In The Wild". In: *CoRR* abs/1802.00434 (2018). arXiv: 1802.00434. URL: http://arxiv.org/abs/1802.00434.

[ZSI19] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. "DPOD: Dense 6D Pose Object Detector in RGB images". In: *CoRR* abs/1902.11020 (2019). arXiv: 1902.11020. URL: http://arxiv.org/abs/1902.11020.

VIII. Appendix

Student contribution towards the project.

Last Name	Patel	Naik
Fraction of total contribution	1/2	1/2
Literature Review	1/2	1/2
Coding	1/2	1/2
Report	1/2	1/2