6D Object Pose estimation

Fabio Gianusso
s327991@studenti.polito.it

Lorenzo Imarisio
s334071@studenti.polito.it

Geard Koci
s328626@studenti.polito.it

Francesco Pedone
s334318@studenti.polito.it

Abstract

In this paper, we are going to introduce a deep learning neural network specifically made for the task of 6D Pose Estimation, a problem that acquired newfound importance as many of its applications emerged with the development of new technologies in recent years. We will first present a simpler model as a baseline that only takes RGB images as input, then we will proceed with the presentation of our own approach. This model takes as input both the RGB and Depth preprocessed information of an image and through the use of a combination of pre-trained, existing models and custom networks it outputs the final pose approximation. A significant part of our work has revolved around the re-training of these pretrained existing models, in order to adapt them to our task. Our research also focuses on experiments to find adequate learning rates and loss function, as a weighted linear combination of different known ones, with the goal of improving the training phase. We will then describe the different approaches and challenges presented during the various steps of the machine learning project pipeline. Finally, we will discuss the results obtained and our takeaways on what other aspects of the model can be improved for better performance. The full code is available at this repository

1. Introduction

6D Object Pose Estimation has emerged as a central problem in many domains of Computer Vision, as in recent years the development of technology has broadened the scope of its applications. From robotic arm control, to autonomous driving and augmented reality, the 6D Pose estimation plays a critical role. This process consists in identifying the orientation and translation of a target object in the 3D space, thus the '6D' referring to the 6 Degrees of Freedom required to define the pose of a rigid object. Traditional methods struggle when dealing with cluttering in the images, poor light conditions, severe occlusion, overlap and confusion. The rise of deep learning has allowed researchers to find more performing and precise models, introducing learning-based methods. In particular, in our work we will show a 6D Pose Estimation method that employs Convolutional Neural Networks combined with an object detection model in order to extract information from RGB-D images. At first, we provide a benchmark with an example of object detection model on RGB Images combined with a pose detection model. Then we proceed to illustrate the main model architecture, the main difference with the baseline is the fact that information on the depth file on the image is also included as an input. This information is at first processed independently from the RGB information, then through a feature fusion phase the two channels are concatenated for further processing to eventually get the output. An important aspect on the training of this model comes from our approach to the loss function, as we will illustrate further on in the paper, we have used a combination of various known losses and evaluation metrics. These losses, that are suitable to evaluate the translation and rotation that compose the pose estimation, have been combined with the presence of different parameters to focus the training of the model on the rotation or estimation accordingly to the situation they were in: in case, for example, of a phase of the training in which the model is struggling to improve the learning of the rotation, we could decide to modify its related parameter by lowering or enlarging it accordingly to how punishing we wanted the loss to be.

2. Related work

DenseFusion [8] is a heterogeneous architecture model that processes the data sources from RGB and Depth images separately and uses a dense fusion network to extract pixel-wise feature embedding, from which the pose is then estimated. In more detail, object segmentation is performed on

a cluttered image, obtaining an image crop that is processed by a CNN to extract the color embeddings of the object. Alongside it, from a depth image, a masked point cloud is obtained and fed to a PointNet model [5] to extract the geometry embeddings. Since it is meant to deal with problems like segmentation noise and occlusion, to aggregate the information from the two channels a local per-pixel fusion is performed instead of a global one. Knowing the camera intrinsic parameters, it is possible to associate the geometric feature of each point to its image feature pixel thanks to a projection onto the image plane. This dense pixel-feature are enriched with global features obtained from the previously obtained pairs of feature put into another network, to provide global context. At the end of the process, a final network predicts the 6D pose of the objects. The key challenge tackled by the model is the extraction of information from the two different channels and its combination. The justification for handling the data differently comes from the fact that the information of color and depth resides in different spaces, so are processed separately to generate color and geometric features from embedding spaces, that retain the intrinsic structure of the data sources.
In this approach the depth pixels are converted in a 3D point cloud and this is then processed by a PointNet-based network to obtain a geometric feature embedding. In our RGB-D model we took inspiration from the idea of combining these two sources of information, the depth channel and the RGB channels, but instead of going through a point cloud, our network simply performs a concatenation of the two.

3. Model architecture

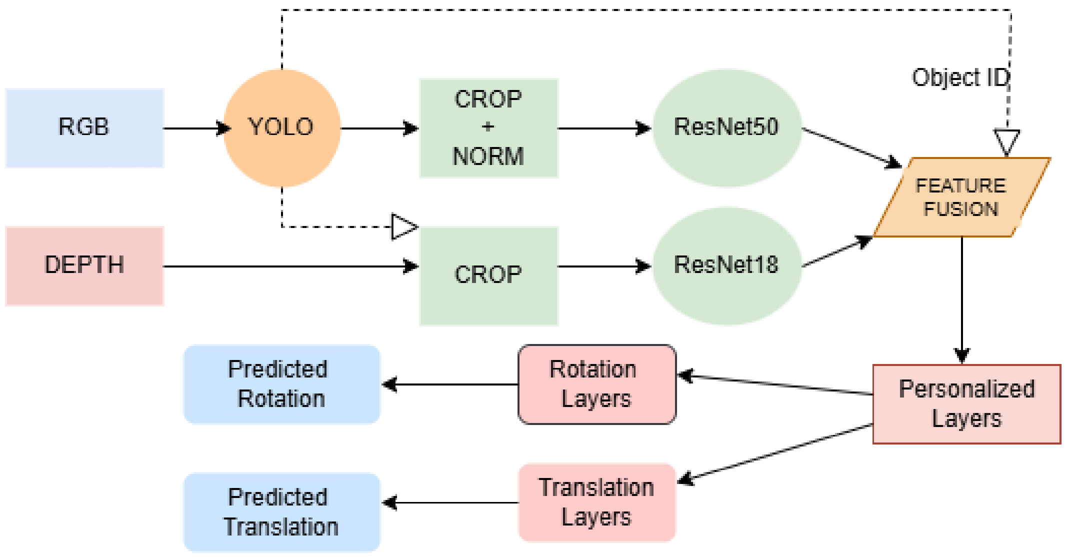

Figure 1. Visualization of the architecture of the model for the RGB-D Pose estimation

3.1. YOLO

YOLO (You Only Look Once) is a neural network architecture for real-time object detection. Its main innovation lies in framing object detection as a regression problem to spatially separated bounding boxes and associated class probabilities, allowing the network to directly predict bounding boxes and class scores from a full image in a single evaluation. Unlike previous methods that employ image classifiers for detection (e.g., R-CNN), YOLO uses a unified architecture optimized end-to-end for detection performance. The input image is divided into a grid and YOLO predicts bounding boxes and class probabilities for each grid cell, their confidence scores, and the class probabilities for the detected objects. This unified approach enables the model to achieve real-time performance. The base YOLO model can process images at 45 frames per second, while a smaller version, Fast YOLO, reaches up to 155 fps, still achieving a mean Average Precision (mAP) twice as high as other real-time detectors. YOLO was originally trained on the PASCAL VOC dataset, which include annotated images spanning 20 object categories. Later versions of the model were also trained on the MS COCO dataset, which provides annotations for 80 object classes and significantly more images, leading to an improvement in the model's generalization capabilities. Although YOLO tends to produce more localization errors compared to other state-of-the-art detection systems, it is less prone to false positives.

3.2. RESNET

ResNet50 is a convolutional neural network (CNN) architecture that belongs to the ResNet (Residual Networks) family, a series of models designed to address the challenges associated with training deep neural networks and is well-known for its depth and efficiency in image classification tasks. ResNet architectures come in various depths, such as ResNet-18, ResNet-32, and so forth, with ResNet50 being a mid-sized variant. The acronym "ResNet" stands for "Residual Network," while the number "50" refers to the model's depth, consisting of 50 layers. The architecture of ResNet50 is composed of four main parts: convolutional layers for feature extraction, convolutional blocks that combine multiple convolutional layers with normalization and activation functions, residual blocks that introduce skip connections to mitigate the vanishing gradient problem during training, and fully connected layers for the final classification. The innovative feature of ResNet is precisely the use of these residual blocks, which allow the activations of one layer to be directly connected to those of a subsequent layer by skipping intermediate layers. This mechanism facilitates gradient flow and enables effective training of very deep networks. Being a CNN, ResNet50 leverages convolutions and pooling to recognize patterns and hierarchical structures within visual data and was trained on large-scale datasets such as ImageNet. Its supervised learning nature allows the model to be refined on labeled images by progressively correcting predictions during training. Thanks to its robustness and ability to extract complex features, ResNet50 is widely used in classification, object detection, and segmentation tasks.

3.3. RGB

As a benchmark for our final model, we built a simpler one that only takes as input RGB images. This simpler architecture uses YOLO [6] to perform object detection on the raw image. The resulting bounding box found this way is then used to crop the image accordingly. After normalization, the result is fed to a CNN, ResNet50[2], which is renowned for its ability to classify images. In particular Resnet50 uses the so called Residual Block, whose key feature is the ability to directly add the unaltered input to the output of the convolutional layers, thus allowing the essential information to be passed through the layers even when the model is struggling to learn. In our case, we did not use the complete Resnet50 architecture, but we removed the final classification layers in order to be able to use the net's ability to extrapolate the features of the image without giving a classification output, but still passing the extrapolated information forward to the next step of the net. This next step is composed a concatenation of such obtained feature with the embedding of the object id found by YOLO. Then the architecture uses personalized layers, that are composed of four main elements:

- FC layers: fully connected layers connect the neurons in the current layer with the ones in the previous one, the feature combination performed by such layer allows for learning global patterns in the data
- Batch Normalization: these layers normalize the output of layers for each mini-batch during the training phase, speeding up the training and allowing for higher learning rates
- ReLu activation function: this is an activation function that has the role of introducing non linearity in the network. This allows the model to learn more complex, non linear patterns and makes for better overall results
- Dropout: a layer that shuts down a neuron with probability p, this happens because is meant to prevent overfitting. The fact that neurons sometimes cannot be selected prevents the network to become too dependent on a single neuron, thus avoiding overfitting

After the custom layers, the information flows into two separate custom networks that then output the rotation and translation predictions that make up the final pose estimation.

3.4. RGB-D

We then created a model that takes both RGB and depth images as input. The architecture of this model is similar to the RGB model, but with some differences. Specifically, after obtaining the bounding boxes using YOLO, we use the same boxes to crop the depth images as well.

For the processing of RGB images we used an extractor based on a pre-trained ResNet-50 from which a 2048-dimensional feature vector is extracted by removing the final fully connected layer. Instead the depth map is processed by a lighter extractor based on ResNet-18 adapted for single-channel input and trained from scratch, in this way we obtained a 512-dimensional feature vector.

To incorporate information regarding the identity of the object, an embedding of the object IDs is used with a dimension equal to 32. This embedding can be replaced by a learned default embedding in case the ID is not available.

The feature vectors extracted from the RGB images, the depth maps and the object ID embedding are concatenated and passed to a common fully connected block consisting of a series of linear layers interleaved with batch normalization, ReLU activation functions and dropout for regularization. This common module reduces the dimensionality to 256.

Then the obtained representation is forwarded to two separate regression heads, dedicated respectively to the estimation of the object's translation and rotation. The translation regression head is composed of two fully connected layers with a ReLU activation function in between. The first layer reduces the dimension from 256 to 128 units, while the sec-

ond produces a final output of 3 dimensions, corresponding to the spatial coordinates (x, y, z) of the object's position in space.
In parallel the rotation regression head follows a similar structure with two fully connected layers and an intermediate ReLU producing a 4-dimensional output representing a quaternion. To ensure that the estimated rotation is valid the produced quaternion is normalized and then converted into a 3×3 rotation matrix.
This dual-head regression structure allows the model to estimate both the position and orientation of the objects simultaneously and consistently, effectively integrating visual and spatial information. The use of quaternion normalization and matrix conversion ensures geometric consistency and facilitates the practical use of the produced rotation estimates.

4. Methodology

4.1. Dataset

4.1.1. LINEMOD

The LINEMOD dataset[1] is one of the most famous datasets when it comes to 6D pose estimation research. It is composed of 15 textureless objects, with each sequence composed of around 1000 images with varying degrees of viewpoint, lighting, occlusion and clutter. Both the RGB and Depth images are available, together, but not limited to, with the ground truths for 6D poses. In the project a subset of the original dataset has been used where the folders for objects 03 (a bowl) and 07 (a cup) have been removed. In accordance with the usual approach when dealing with the dataset, 15 % of the images have been used for training and the remaining 85 % for testing.

4.1.2. Preprocessing

To produce an optimal model, the LINEMOD dataset, that is going to provide the training and validation set, needs preprocessing. This is because despite LINEMOD is a very popular choice of training set, the data itself still needs adjustments in order to be used. For the YOLO model we structured the dataset by dividing images into 'train' and 'val' folders, renaming each file using the convention ClassID_ImageNumber. For each image, a corresponding '.txt' label file was created following the YOLO annotation format: `<class> <x_center> <y_center> <width> <height>`.
In this format, all values are normalized relative to the image dimensions where x_center and y_center represents the center of the bounding box and 'width', 'height' its size. Finally, we prepared a 'data.yaml' configuration file specifying:

- class names indexed from 0 to 15 (with unused placeholders),
- 'nc: 16' indicating the total number of classes,
- paths to the 'train' and 'val' image directories.

This preprocessing ensures that the dataset is properly formatted and ready for training with YOLO.
Instead for the other models the first thing we did is the normalization of the data, that allows for faster convergence and is also useful when dealing with pretrained models, like Resnet50 in our case. In our approach preprocessing consisted in a first phase of resizing the images, so that they presented consistent and uniform dimensions, required when feeding this input to our network, this uniformity also augment the efficiency of the process.

Then the images themselves are transformed into tensors, which is the standard structure used in the Pytorch library[3] and is also the way to ensure that the mathematical models deal with actual numbers. With tensorization the pixel values were transformed from their normal scale from 0 to 255 to a [0,1] scale and then they were finally standardized. As said above, having data on a similar scale improves convergence and efficiency of the model, particularly in pose estimation is crucial that slight spatial differencies are detected accurately and are not overshadowed by an overwhelming variations in the scale. Finally, the ground truth for the translation data were also normalized.

4.2. Technical aspect of tensor transformation

As a separate part of pre-processing, we realized that downloading the dataset with the images and transforming them into tensors is a time consuming process. So in order to obtain the dataset that is then going to be used to train the part of our network that follows the YOLO object detection section, we decided to directly transform the images into tensors one single time, and then to save it on the group's google drive so that the script can directly access it through the Colab ambient.

4.3. Training

4.3.1. Training YOLO

For the object detection part of the model, we implemented YOLO, a convolutional neural network that when introduced distinguished itself from other models because it only requires a single forward pass through the neural network to make a prediction. In particular we used the version 8 of the model, in its nano variant, which lacks in accuracy in comparison to the others models in the v8 family, but as a tradeoff offers more speed and resource efficiency. YOLO v8 is available as a pretrained model, but since the objects' classes present in the LINEMOD dataset did not completely overlap, we opted for retraining the model, which is a straightforward process, but very time consuming. After 30 epochs the training was stopped since we were satisfied with the results.5

4.3.2. MaP metric

The mAP is a metric used to evaluate the performance of models in object detection tasks and information retrieval on images. It measures the accuracy of a model by considering both precision and recall, which account for false positives and false negatives, making mAP a suitable metric for most detection applications. The mean Average Precision is computed by calculating the Average Precision (AP) for each class and then averaging over all classes.To compute AP, the model's performance is summarized using the Precision-Recall curve, which plots precision against recall at various classification thresholds. Let n be the number of classes then:

$$mAP = \frac{1}{n}\sum_{k=1}^{n} AP_k$$

AP is the average of precision at different levels of recall, providing a single score to summarize the model's performance:

$$AP = \frac{1}{N}\sum_{i=1}^{N} Precision(i) \cdot \Delta Recall(i),$$

where N is the number of thresholds used to approximate the Precision-Recall curve for class k.
The computation of mAP relies on the Intersection over Union (IoU), which measures the overlap between the predicted bounding box and the ground truth. A predicted object is considered a correct detection if its IoU with the ground truth exceeds a predefined threshold. This threshold directly impacts the AP score, and consequently the final mAP value. In order to evaluate detection performance we used two common variants of the mAP metric:

- **mAP50:** computes the mean Average Precision using a fixed IoU threshold of 0.50, thus allowing relatively generous matching and emphasizing detection of objects regardless of localization precision.
- **mAP50–95:** averages the mAP over multiple IoU thresholds ranging from 0.50 to 0.95 with a step size of 0.05. This provides a more rigorous and comprehensive evaluation, capturing model performance across varying degrees of localization accuracy.

4.3.3. Loss function

In order to train our model, we employed a personalized Loss function, that we computed on every single sample of the training and not only on the entire batches. The loss is a combination of 3 different evaluation metrics weighted by different coefficients:

- Smooth L1 Loss (Huber Loss)
- Geodesic Loss
- ADD metric

The Smooth L1 Loss[4] focuses on evaluating the predicted translation of the pose estimation. In particular when the difference between the ground truth and the prediction is large, it behaves like an L1 Loss, when its small like an L2 Loss. There is a dedicated parameter β that controls the threshold between these two losses and allows the loss to be more or less sensible to outliers. Its formula is given by:

$$\mathcal{L}_{\text{Smooth L1}}(x,y) = \begin{cases} \frac{1}{2}(x-y)^2/\beta & \text{if } |x-y| < \beta \\ (|x-y| - \frac{1}{2}\beta) & \text{otherwise} \end{cases}$$

The Geodesic Loss focuses[7] on evaluating the rotation part of the pose estimation. Since we are trying to predict a rotation in 3 dimensions, an Euclidean loss wouldn't be geometrically accurate, so we use this more precise Loss that measures the shortest angular distance between the two rotation matrices. Its formula is given by:

$$\mathcal{L}_{\text{geo}}(R_1, R_2) = \cos^{-1}\left(\frac{\text{Tr}(R_1^\top R_2) - 1}{2}\right)$$

Finally we included the ADD metric (Average Distance of Model points)[9], which measures the difference between the predicted pose and the ground truth pose, by computing the average distance between the 3D model's points. Its formula is given by:

$$\text{ADD} = \frac{1}{|M|}\sum_{\mathbf{x}\in M} \|(R_{\text{gt}}\mathbf{x} + \mathbf{t}_{\text{gt}}) - (R_{\text{pred}}\mathbf{x} + \mathbf{t}_{\text{pred}})\|$$

The choice of introducing both the Geodesic and ADD into the Loss function significantly improved the training of the model and the subsequent outputted results. Furthermore these metrics' weighted combination allowed us during different phases of the training epochs to help the model focus more on the translation or the rotation aspect of the pose estimation according to the evaluation values that were being shown. The final formula for the Loss is expressed as:

$$\mathcal{L} = \gamma \cdot \text{ADD} + \alpha \cdot \mathcal{L}_{\text{Smooth L1}} + \beta \cdot \mathcal{L}_{\text{geo}}$$

In order to efficiently train a model, 3 major hyperparameters have been used in our specific case: batch size, number of epochs and learning rate. For computational power constraints, in particular RAM constraints, we couldn't work with batch sizes larger than 8. Our biggest asset for optimal training of the dataset was the manipulation of the learning rate to accommodate the evolution of the loss along the epochs, in particular we manually changed it when we detected the different losses hitting a plateau.

5. Experiments and Results

5.1. Learning rate approach

In this section we start by analyzing in more detail our approach when dealing with the learning rate, one of the hyperparameters we can control for improving our model. As is common knowledge, the learning rate tells the model the extent of the update of the weights in the model, along the direction suggested by the loss. In such a complex model made with different subsections, to have just a single learning rate means that all the different parts of the net explore the space at the same pace, so a part of the model could be struggling while a different one is performing well and there is no way to adjust for this difference. This is the reason why we chose to divide the learning rate into four different parts, one for the layers common to all the elements of the model (our custom ones, ”Personalized Layers” in Figure 1) , one for Resnet, one for the rotation CNN and finally one for the translation CNN. This allows for independently control how the different parts of the model learn, since it allows for higher learning rate for elements that are struggling and lower ones for elements that are already close to a minima. As for the way we chose to update the model, when the training performance was approaching a plateau, we lowered it accordingly, in order to better explore the region. It is also important to denote that during the training of the more complex model, when confronted with a model that was struggling to improve after a significant amount of epochs, we actually resorted to raising considerably the learning rate in order to get out of a possible local minima, with satisfying results.

5.2. Loss parameters approach

As anticipated in previous sections, our loss uses a combination of losses and metrics for optimized learning, regulated by different parameters. The idea behind the update of those parameters during training comes from wanting to make the training focus on the aspect of rotation and translation depending on which one of the two was struggling to improve. The loss is a linear combination of three terms, the main one being the ADD metric, which is comprehensive both of the translation and rotation aspect of the approximation, as it measures the 3D distances of the points of the 6D pose models predicted from the ground truth. This means that the weight associated with this term is kept at a higher value than the ones for the smooth L1 loss and Geodesic loss, as to be expected for a section that express the combination of the two. All three parameters are then update during training with the idea of, firstly, maintain the balance between them, and then raising the value of the parameter of the part of the model that is struggling. For example if it is observed that on the course of a set amount of epochs the rotation value is not improving, its associated parameters can be updated so that the Geodesic loss associated to it is more relevant in the comprehensive Loss, by having a more weighted value on it, the model will focus more upgrading that aspect.

5.3. Results

Table 1. Object detection metrics for each class

Class	Precision	Recall	mAP50	mAP50-95
01	0.996	0.946	0.994	0.850
02	0.995	0.962	0.992	0.817
04	0.973	0.955	0.990	0.772
05	0.951	0.994	0.989	0.854
06	0.991	0.976	0.994	0.855
08	0.971	0.966	0.991	0.860
09	0.989	0.972	0.994	0.857
10	0.997	0.968	0.995	0.887
11	0.974	0.947	0.988	0.783
12	0.899	0.999	0.994	0.863
13	0.996	0.995	0.995	0.831
14	0.999	0.990	0.995	0.877
15	0.979	0.979	0.994	0.826
All	**0.978**	**0.973**	**0.993**	**0.841**

Table 1 shows the performance of the object detection model for each class, with metrics including precision, recall, mAP50, and mAP50-95. The results show high overall accuracy, with average precision and recall values of 0.978 and 0.973, respectively, and a mAP50 of 0.993. Even the more stringent metric, mAP50-95, reaches an average value of 0.841 indicating a good balance between accuracy and generalization in object localization. Some classes, such as class 12, exhibit a recall close to one but slightly lower precision suggesting a higher number of false positives. Overall all classes show very solid performance with mAP50 consistently above 0.98.

Table 2. Pose estimation results with RGB and RGB-D input

Model	Epoch	ADD	Rotation (°)	Translation
RGB-D	189	2.0073	43.40	0.4870
RGB	161	1.8738	43.11	0.4172

Table 2 reports the performance of the pose estimation models using RGB and RGB-D input.The reported values indicate a good overall quality of the pose estimations: an ADD value around 2 suggests a relatively small average distance between the estimated and actual model points, which reflects accuracy in the pose estimation. Meanwhile a translation error below 0.5 indicates good precision in the spatial

localization of the object.
The RGB-D model achieves a mean rotation error of 43.40°, slightly higher than the 43.11° obtained by the RGB model, and a translation error of 0.4870 compared to 0.4172. In terms of ADD as well, the RGB model yields a lower value, indicating a more accurate pose estimation.
However, these results do not necessarily imply that depth information is ineffective. The training of the RGB-D model was limited due to high computational costs. With extended training, it is reasonable to expect that the RGB-D model, thanks to the greater amount of available information, could outperform the RGB-only model but it would require way more time due the larger amount of parameters to be tuned.

5.4. RGB-D Model

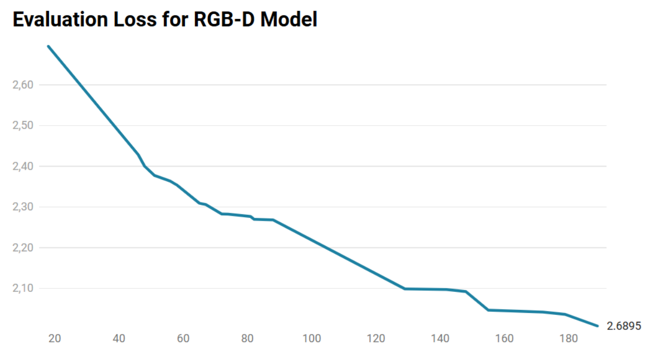

Figure 2. Evaluation loss for the RGB-D Model along the epochs

5.5. Experiments

Several experiments were conducted to design and refine the pose estimation models, with the aim of identifying the most effective architectural choices and loss functions.
In the initial phase, the Mean Squared Error (MSE) was used as the loss function for both translation and rotation. Subsequently, to improve the estimation of orientation, the rotation loss was replaced or complemented with the geodetic loss, which is better suited for comparing rotation matrices.

Another experiment involved using the L1 loss and MSE instead of Smooth L1 Loss for translation, in order to reduce the impact of outliers. To evaluate the overall quality of the predicted poses, we added the ADD metric, which provides a more comprehensive measure by accounting for both translation and rotation errors in a spatially meaningful way.
From an architectural standpoint the rotation head initially produced a 3×3 matrix directly. However, this approach did not guarantee that the output would be a valid rotation matrix. To address this, the model was modified to output a quaternion which is then normalized and converted into a valid 3×3 rotation matrix, thus ensuring the geometric consistency of the output.

The structure of the fully connected layers was also optimized: the first version of the model employed a relatively simple configuration, while later experiments adopted a more refined design consisting of three shared layers followed by two separate branches—one for translation and one for rotation—each composed of two fully connected layers and generally a more complex network with many more layers and techniques as batch normalization and dropout. Finally, a tuning phase was carried out, involving adjustments to the layer depth, dropout rates, learning rates, and embedding sizes, to further enhance the model's performance.

6. Discussion

In this section, we highlight the main takeaways from our implementation, note the limitations, and suggest directions for future improvement. References to quantitative results from Section 5 are marked as placeholders.

6.1. Positive Outcomes

6.1.1. Effective Depth RGB Fusion.

The addition of depth data to RGB inputs can be considered promising to improve pose estimation, especially in scenarios with uniform textures or cluttered backgrounds. Depth cues helped disambiguate object edges and maintain accurate orientation under partial occlusion. The concept is that with the same number of epochs the results are comparable but the RGB-D version shows more improving potential with an eventual larger number of epochs. Indeed, while with the RGB model through epochs there was not further loss decrease, the only limitation we found during the RGB-D training were resources as computational time and GPU availability.

6.1.2. Per Branch Learning Rates.

Assigning separate learning rates to the backbone, RGB branch, and pose heads stabilized training: larger rates for the rotation/translation heads accelerated convergence on their specialized tasks, while a smaller rate for the shared backbone prevented overfitting. Our ablation (Section 5) shows this multi-LR approach outperformed a uniform learning-rate setup.

6.1.3. Phased Loss Weighting.

Employing a three-stage loss schedule—initially balancing all components, then emphasizing rotation, and finally focusing on translation—ensured that the model refined each aspect of pose estimation in turn. This strategy yielded balanced error reductions across both angular and positional metrics, as can be shown in figure 2.

6.2. Limitations

6.2.1. Hardware Constraints.

Due to the limited GPU, we used a batch size of 8, which may have increased gradient noise and slowed convergence. A higher-memory GPU could enable larger batches (e.g., 16 or 32) and potentially improve generalization. For the shown experiments different GPUs from Google Colab Pro were used as T4 and A100.

6.2.2. Fixed Camera Intrinsics.

Assuming known, static camera intrinsics works for LINEMOD but may not hold in real world applications with calibration drift. Incorporating a small module to refine intrinsics during training could mitigate this dependence and improve robustness across different sensors.

6.2.3. Inference Speed.

Although individual components (YOLO v8, ResNet-50) are compact, the combined pipeline probably runs below real time requirements for many robotics tasks.

6.2.4. Custom Neural Networks.

Using our own custom layers arranged sequentially to compute translation and rotation for both RGB and RGB-D models can lead to slower performance. Unlike highly optimized architectures such as ResNet or YOLO, our models aren't as finely tuned, which may result in reduced efficiency.

6.3. Future Work

6.3.1. Automated Hyperparameter Tuning.

Manual selection of learning rates and loss weights proved effective but may not generalize. Implementing Bayesian optimization or population based search could find more robust configurations with less human effort.

6.3.2. Adaptive Calibration.

Adding a lightweight network to predict small corrections to camera intrinsics could allow the model to adjust to sensor variations on the fly, reducing reliance on perfect calibration.

6.3.3. Efficient Depth Encoding.

Exploring alternatives, such as sparse 3D convolutions or ordered point cloud representations, could cut computation without sacrificing geometric fidelity, aiding both accuracy and speed.

6.3.4. Multi-Object Handling.

Current inference assumes a single object per scene. Extending the pipeline to process multiple overlapping detections, perhaps via a joint pose would enable robust pose estimation in cluttered environments.

6.3.5. Real-Time Deployment.

Leveraging model compression (pruning, quantization) and optimized inference engines will be crucial to reach higher speeds.

6.4. Summary

In summary, fusing depth with RGB features combined with per-branch learning rates and phased loss weighting yields substantial gains in 6D pose accuracy. While hardware and dataset limitations constrain batch size, inference speed, and object diversity, the core methodology forms a solid foundation. Addressing these bottlenecks in future work will enable more accurate, efficient, and generalizable pose estimation for real-world applications.

References

[1] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision (ACCV)*, pages 548–562. Springer, 2012. 4

[2] PyTorch. Resnet-50 — torchvision main documentation. https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html, 2024. Accessed: 2025-06-06. 3

[3] PyTorch Contributors. Pytorch: An open source machine learning framework. https://pytorch.org/docs/stable/index.html, 2024. Accessed: 2025-06-06. 4

[4] PyTorch Contributors. torch.nn.smoothl1loss — pytorch documentation. https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html, 2024. Accessed: 2025-06-06. 5

[5] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017. 2

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. 3

[7] Seyed Sadegh Mohseni Salehi, Shadab Khan, Deniz Erdogmus, and Ali Gholipour. Real-time deep pose estimation with geodesic loss for image-to-template rigid registration. *IEEE Transactions on Medical Imaging*, 38(2):470–481, 2019. 5

[8] Chen Wang, Danfei Xu, Cewu Lu, Yuke Zhu, Li Fei-Fei, Roberto Martín-Martín, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3343–3352, 2019. 1

[9] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018. 5