34212-Lab-S-Report

Sourabh Roy

March 2025

1 Deep Learning Methods in Robot Vision: A State-of-the-Art Review

Robotics is a rapidly evolving field with diverse applications, where Robot Vision plays a critical role by enabling robots to perceive and interpret their surroundings through visual data. Advances in **Deep Learning (DL)**, particularly in **Computer Vision (CV)**, have greatly improved the vision capabilities of robots. Unlike traditional CV systems processing static images, robot vision operates in real-time dynamic environments with noisy or incomplete data. Robots must not only recognize objects but also understand spatial relationships and make decisions on the fly.
This review covers recent deep learning methods in robot vision. In particular, it describes key state-of-the-art models in object detection and recognition, semantic and instance segmentation and 3D perception and depth estimation.

1.1 Object Detection and Recognition

Object detection involves both classification and localization, and is used to analyze more realistic cases in which multiple objects may exist in an image. In an input image, each object is categorized and appropriate bounding boxes for the identified objects are determined. In robotics, this capability is fundamental for tasks such as grasping, obstacle avoidance, and semantic mapping. Early approaches focused on two-stage detectors, with the R-CNN family[13], including Fast R-CNN, and Faster R-CNN, leading significant progress. These methods use region proposals followed by convolutional neural network (CNN)-based classification and bounding box regression (refining the coordinates of rectangular boxes (bounding boxes) around detected objects). Faster R-CNN, in particular, remains widely used in robotic perception tasks for its accuracy, making it suitable for scenarios requiring precise manipulation.

As robotics moved towards real-time applications, one-stage detectors like the YOLO (You Only Look Once) series[15] became popular. These models perform detection and classification simultaneously, greatly improving inference speed. YOLOv8, for instance, is commonly employed in autonomous drones and mobile robots due to its real-time processing capabilities. Recently, transformer-based detectors such as the Detection Transformer (DETR)[1] introduced new paradigms by leveraging transformers to eliminate the need for anchor boxes and hand-crafted components. Models like Deformable DETR[21] further enhance performance in cluttered and complex robotic environments. Multimodal approaches integrating RGB, depth, and LiDAR inputs have also become prevalent. These provide robustness against occlusion and varying lighting conditions.

1.2 Semantic and Instance Segmentation

Semantic segmentation assigns a class label to every pixel in an image, enabling robots to perform tasks such as terrain classification or free-space estimation. Instance segmentation takes this further by distinguishing individual instances of objects within the same class, which is crucial for precise manipulation and interaction with objects in the environment. Initial deep learning models for segmentation relied heavily on CNN-based architectures. Fully Convolutional Networks (FCNs) laid the groundwork for pixel-wise predictions, while U-Net[12], with its encoder-decoder structure, became widely used in specialized areas like medical and agricultural robotics due to its accuracy and flexibility.

The DeepLab family, particularly DeepLab v3+[3],incorporates atrous convolutions which are a type of convolution operation that increases the receptive field without increasing the number of parameters or losing resolution. This improved boundary delineation. For tasks requiring both detection and segmentation, Mask R-CNN extends Faster R-CNN by adding a dedicated segmentation branch. It is a promising choice for mobile manipulation and scene understanding. More recently, transformer-based models such as Mask2Former[4] have demonstrated superior performance by capturing long-range dependencies within images which improved segmentation accuracy in complex scenes. Real-time models like Fast-SCNN[8] have been developed specifically for low-latency environments, making them excellent choices for navigation tasks in autonomous vehicles.

1.3 3D Perception and Depth Estimation

3D scene understanding involves comprehending the spatial and semantic layout of a 3D environment, including depth estimation, scene reconstruction, and understanding object relationships. Robots need this ability to navigate, plan, and interact with complex environments effectively. Depth estimation and 3D reconstruction have been significantly advanced by deep learning models. Monocular depth networks like DPT[11] predict depth from a single RGB image, essentially allowing a computer to ”see” 3D from a 2D perspective. This is a practical solution when only monocular data is available. Stereo matching networks such as PSMNet[2] and GA-Net[19] leverage stereo pairs to enhance depth estimation accuracy. A stereo pair refers to two images of the same scene captured from slightly different viewpoints allowing for 3D reconstruction and depth perception.

Processing point clouds directly has also become a major area of focus. A point cloud is a set of data points in 3D space that represent the surface or shape of an object or scene, with each point having x, y, and z coordinates. PointNet[9] and PointNet++[10] are the pioneering models in processing point cloud data while VoxelNet[20] offers a latest end to end solution by unifying feature extraction and bounding box prediction into a single stage, end-to-end trainable deep network.

1.4 Conclusion

Deep learning advancements have pushed the limits of robot vision. Despite these strides, real-time performance, occlusion handling in segmentation, and depth estimation in low-texture environments remain critical challenges. Future research should prioritize efficient network architectures, probabilistic modeling for uncertainty estimation, and cross-domain adaptation to enhance generalization in diverse robotic applications.

2 Hyperparameter Exploration and Optimization in Deep Learning for Robot Vision

2.1 Introduction to Image Classification

Image classification is a fundamental task that attempts to comprehend an entire image as a whole. The goal is to classify the image by assigning it to a specific label. Typically, this involves images in which only one object appears and is analyzed.

In this work, the CIFAR-100 dataset is used, which contains 60,000 color images (32x32 pixels) divided into 100 fine-grained classes. 40,000 images were used for training and 10,000 for validation and 10,000 for testing. It covers a wide range of object categories, making it a challenging benchmark for deep learning models due to the visual similarity between some classes and the limited resolution. The choice of CIFAR-100 is well justified for this work, especially in the context of robot vision. Robots operating in dynamic, real-world environments should be able to identify objects, often under constraints such as low resolution, limited processing power, and noisy visual input. CIFAR-100 simulates these challenges by providing small-sized images with difficult class distinctions.

2.2 Methodology

Data Processing Pixel values of the images were normalized to the range [0, 1] by dividing by 255, ensuring consistent input distributions. The labels were converted into one-hot encoded format to match the neural network’s output structure, with the number of classes determined dynamically from the data. Each image was resized to (32×32) with 3 RGB channels, defining the input shape as ((32, 32, 3)). These steps ensured the data was uniformly scaled, properly encoded, and compatible with the CNN architecture.

Network Topology The model follows a CNN-based architecture optimized using Hyperband. It starts with a fixed convolutional block of two 32-filter layers (3×3 kernels), ReLU-based ELU activations, and max-pooling. ELU was selected as the activation function due to its superiority over ReLU in image classification in CIFAR100 as highlighted in [5].

This is followed by variable **convolutional blocks** (1 to 3), where each block stacks two Conv2D layers with progressively increasing filters ($32{*}2^i$, where i, $1 \leq i \leq 3$, denotes the i^{th} layer), ELU activations, max-pooling, and dropout for regularization. The reason for choosing neurons of the order 2^k is that increasing the width exponentially compensates for the reduced depth which allows networks to approximate complex functions efficiently[14]. The network concludes with a dense block: a fully connected layer sized based on the final convolutional output, followed by dropout and a softmax output layer for 100-class prediction. The Adam optimizer is used with a tunable learning rate.

Hyperparameter Choice and Exploration Three key hyperparameters were identified based on their influence as discussed in prior literature: **learning rate** (lr), **dropout rate** and **number of convolutional blocks**. In [17] and [18], the authors demonstrate that these hyperparameters consistently rank as the most significant ones based on importance weight comparisons from FANOVA tests, thereby supporting their selection.

We determine the best hyperparameter configuration using two methods: manual estimation and Hyperband. For both methods, we select 3 hyperparameter valuations as shown in the table below.

Hyperparameter name	Valuation 1	Valuation 2	Valuation 3
Learning rate	10^{-3}	10^{-4}	10^{-5}
Dropout rate	0.25	0.3	0.5
Number of Blocks	1	2	3

Table 1: Valuations for each hyperparameter.

In manual estimation, we explore by training and testing models on every hyperparameter value. When we vary a particular hyperparameter, we keep the rest of the hyperparameters fixed at a default value. For example, while experimenting with learning rate, we vary the learning rate values while the dropout rate will be fixed at 0.25 and the number of CNN blocks will be fixed at 2.

Hyperband [7] is an efficient hyperparameter optimization method that adaptively allocates resources to promising configurations while pruning underperforming ones early. It builds on Successive Halving to evaluate configurations with increasing budgets, eliminating weaker ones in each iteration. This approach speeds up optimization and is faster and more resource-efficient than popular hyperparameter estimation algorithms such as Bayesian Optimization and Random Search, making it ideal for deep learning tasks with limited computational budgets.

We present a discussion on which method is optimal in the Results section.

Training and Experimental Simulations We implemented the **Hyperband** algorithm using `Keras tuner`. In the manual estimation, for each trial, the model was trained and evaluated **3 times** to mitigate the effects of stochastic variability. During training, we used `ImageDataGenerator` to apply real-time data augmentation. These transformations in the input images enhance dataset diversity, reducing overfitting by making the model more robust to variations and distortions in input data. We also used **early stopping** to monitor the validation performance during training and terminate the process once the model ceased to improve. It helps prevent overfitting by ensuring that the model does not continue to train on the data after reaching optimal performance. Given the limited size of CIFAR-100, these measures are critical to improve the generalization capability of the CNN.

We compare the best hyperparameter configuration obtained from manual estimation and Hyperband by constructing two models, each configured with the respective hyperparameter sets. Using the same pipeline as before, we train and evaluate thrice and report the average test accuracy in Results. Due to resource constraints, all models were trained for 20 epochs—a compromise between computational efficiency and performance.

To visualize the performance across different hyperparameter values, training and validation accuracy and loss were plotted as the mean, with the standard deviation represented as a shaded region, providing a clear indication of the variability in results. The code for this work can be found in the repository: Cognitive-Robotics-Coursework.

2.3 Results

The results of the experimental simulations are presented in this section, highlighting the impact of each hyperparameter on the model's performance. First, the results of the manual estimation have been presented with performance metrics such as training, validation and test accuracy and loss, recorded across 3 runs.

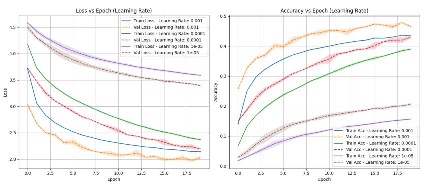


Figure 1: Learning rate loss and accuracy vs epochs for training and validation datasets.

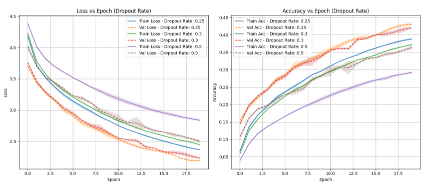


Figure 2: Dropout rate loss and accuracy vs epochs for training and validation datasets.

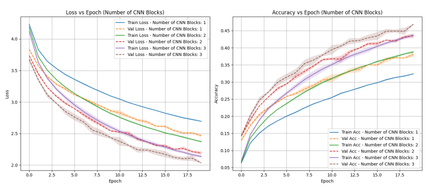


Figure 3: Number of CNN blocks loss and accuracy vs epochs for training and validation datasets.

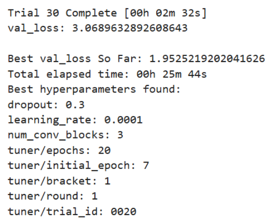
Interestingly, in all the 6 plots above, the model has lower validation loss than training loss and higher validation accuracy than the training accuracy. This can be attributed to either the validation set containing easier classes than the training set or perhaps the model was over-regularized due to the negative outputs from ELU activation function. The narrow shaded regions of the standard deviations indicate that the model's performance is consistent across different training runs.

Hyperparameter name	Valuation 1	Valuation 2	Valuation 3
Learning rate	**0.4835**	0.4341	0.2013
Dropout rate	**0.4377**	0.4220	0.3602
Number of Blocks	0.3779	0.4335	**0.4665**

Table 2: Average test accuracy for different hyperparameter valuations.

The average accuracies for the test dataset for different hyperparameter values have been displayed in the table above. In case of learning rate, the average test accuracy fluctuates from 48.35% to 20.13% with different valuations which are the highest and lowest accuracy measures respectively, across all training simulations. This proves that the learning rate is indeed the most sensitive model hyperparameter. In case of dropout rate, there is little difference in the performance metrics for the first and second valuations during validation. While, in training, the first valuation performs better.

The best hyperparameters graphically are: learning rate = 10^{-3}, dropout rate = 0.25 and number of CNN blocks = 3. Each of them had the highest accuracy plot and the lowest loss plot for training and validation in their category. However, this contrasts with the results from the Hyperband, shown below.



```
Trial 30 Complete [00h 02m 32s]
val_loss: 3.0689632892608643

Best val_loss So Far: 1.9525219202041626
Total elapsed time: 00h 25m 44s
Best hyperparameters found:
dropout: 0.3
learning_rate: 0.0001
num_conv_blocks: 3
tuner/epochs: 20
tuner/initial_epoch: 7
tuner/bracket: 1
tuner/round: 1
tuner/trial_id: 0020
```

Figure 4: Best parameters returned by Hyperband.

For the number of CNN blocks, unsurprisingly the largest valuation was preferred by both methods. The reason for the discrepancy in other hyperparameters can be explained by the fact that: although both Hyperband and manual estimation trials were run for 20 epochs, Hyperband's configurations performed better, likely because it explored a broader hyperparameter space and identified a more optimal combination that manual estimation trials missed.

Model Hyperparameters	Average Test Accuracy	Time of completion
Hyperband	**0.4695**	**0h 25m 44s**
Manual Estimation	0.4217	3h 35m

Table 3: Comparison between models defined using best hyperparameters from Hyperband and Manual Estimation.

Clearly, Hyperband outperforms manual hyperparameter tuning in both model performance and computational efficiency. This highlights that for large and complex datasets in deep learning tasks, algorithmic approaches to hyperparameter optimization are the superior choice.

2.4 Future Work

The steep slopes of the accuracy and loss plots in Figure 2 and Figure 3, indicate that 20 epochs are not adequate for model training. It shows that the training was stopped before model convergence. Hence, one can increase the number of epochs (in both Hyperband and manual trials) as well as the runs (from 3 to 10 for example) to get better convergence and more stable results. Further, comparing Hyperband with slower but more reliable methods such as Population-Based Training (PBT) [6] on state-of-the-art Robot vision architectures such as Faster R-CNN and benchmark datasets, can offer valuable insights into their applicability for Robot Vision tasks.

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[2] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network, 2018.

[3] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[4] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation, 2022.

[5] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[6] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks, 2017.

[7] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[8] Rudra P K Poudel, Stephan Liwicki, and Roberto Cipolla. Fast-scnn: Fast semantic segmentation network, 2019.

[9] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017.

[10] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017.

[11] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction, 2021.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[13] Rabi Sharma, Muhammad Saqib, Chin-Teng Lin, and Michael Blumenstein. A survey on object instance segmentation. *SN Computer Science*, 3(6):499, 2022.

[14] Matus Telgarsky. Benefits of depth in neural networks, 2016.

[15] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, November 2023.

[16] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds, 2019.

[17] Ruinan Wang, Ian Nabney, and Mohammad Golbabaee. Efficient hyperparameter importance assessment for cnns, 2024.

[18] Mikolaj Wojciuk, Zaneta Swiderska-Chadaj, Krzysztof Siwek, and Arkadiusz Gertych. Improving classification accuracy of fine-tuned cnn models: Impact of hyperparameter optimization. *Heliyon*, 10(5):e26586, 2024.

[19] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip H. S. Torr. Ga-net: Guided aggregation net for end-to-end stereo matching, 2019.

[20] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection, 2017.

[21] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021.

A Raw test accuracies for manual estimation of hyperparameters

Learning Rate	Trial 1	Trial 2	Trial 3	Average
10^{-3}	0.4797	0.4829	0.4879	0.4835
10^{-4}	0.4329	0.4317	0.4377	0.4341
10^{-5}	0.1998	0.2027	0.2014	0.2013

Dropout Rate	Trial 1	Trial 2	Trial 3	Average
0.25	0.4372	0.4383	0.4376	0.4377
0.3	0.4222	0.4221	0.4217	0.4220
0.5	0.3607	0.3589	0.3610	0.3602

No. of CNN Blocks	Trial 1	Trial 2	Trial 3	Average
1	0.3771	0.3751	0.3815	0.3779
2	0.4337	0.4309	0.4359	0.4335
3	0.4686	0.4639	0.4670	0.4665