

A Project Report
On
Object Localization and Pose Estimation using DL and CNN

BY
AKSHIT JAIN
2023A4PS0490H

Under the supervision of
PROF. ABHISHEK SARKAR

**SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF
ME F266: STUDY PROJECT**



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)
HYDERABAD CAMPUS
(MARCH 2025)**

ACKNOWLEDGMENTS

I extend my heartfelt appreciation to the Mechanical Department of BITS Pilani Hyderabad Campus for granting me the opportunity to undertake this Design Project. Without their invaluable support and unwavering encouragement, the successful execution of the project on “Object Localization and Pose Estimation using DL and CNN” would not have been possible.

I wish to convey my deep gratitude to my esteemed mentor and guide, Prof. Abhishek Sarkar , for his unwavering patience, unwavering motivation, boundless enthusiasm, and profound expertise. His guidance and unwavering assistance have been a cornerstone throughout every phase of this project

With the guidance and support of the Mechanical Department and Prof. Abhishek Sarkar , I have been able to explore and contribute to this captivating field, and for that, I am truly Thankful.

Akshit Jain



Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled “**Object Localization and Pose Estimation using DL and CNN**” submitted by Mr. AKSHIT JAIN (ID No. 2023A4PS0490H) in partial fulfillment of the requirements of the course ME F266, Study Project Course, embodies the work done by him under my supervision and guidance.

Date:

11/03/2025

(PROF. ABHISHEK SARKAR)

BITS- Pilani, Hyderabad Campus

ABSTRACT

This project is a study about Deep Learning and Convolutional Neural Network and their applications in the field of computer vision, specifically Object Localization and Pose Estimation. Object localization and pose estimation play a crucial role in various computer vision applications; including robotics, augmented reality, self-driving cars etc.

Through this project I plan to develop a basic and intuitive understanding of DL and CNN. I investigate models such as the Segment Anything Model (SAM) and You Only Look Once (YOLO) for object detection and segmentation, along with geometric algorithms like RANSAC and the Perspective-n-Point (PnP) method for accurate pose estimation. This work involves an in-depth study of various vision models(PoseCNN, EfficientPose), analysing their inner workings and evaluating their impact and performance.

By combining data-driven deep learning techniques with classical computer vision algorithms, this project aims to achieve a robust and adaptable system for precise object localization and pose estimation. Experimental results demonstrate the effectiveness of this hybrid approach in improving accuracy and reliability across diverse conditions.

Code: https://github.com/Random-homosapien/Computer_Vision_SOP

Table of Contents

Title Page.....	1
ACKNOWLEDGMENTS	2
Certificate	3
ABSTRACT	4
Chapter 1: Basics of CNN	7
Theory:	7
Code:	7
Execution:.....	8
Chapter 2: Basics of Computer Vision:.....	9
Theory:	9
Code:	9
Conclusion.....	9
Chapter 3: YOLO + SAM	10
YOLO:.....	10
SAM:	12
Experimentation:	13
Code:	14
Results:.....	14
Chapter 4: RANSAC + PNP (DPOD)	17
Introduction:	17
Experimentation:	18
Possible Reasons for Failure:	20
Conclusion:.....	21
Chapter 5: PoseCNN.....	22
Introduction:	22
Theory:	22
Code:	23
Experimentation:	23
Conclusion:.....	24
Chapter 6: EfficientPose.....	25
Introduction:	25
Theory:	25
Code:	26

Execution:.....	26
Conclusion:.....	30
Conclusion.....	31
References	32

Chapter 1: Basics of CNN

Theory:

To develop an intuitive and practical understanding of **Convolutional Neural Networks (CNNs)**¹, I implemented a handwriting recognition model in Python. This project involved training a CNN to recognize handwritten numerical digits using a labelled dataset. The dataset consisted of grayscale images of handwritten numbers, each stored as a **28×28 pixel matrix**, along with their corresponding numerical labels. The objective was to train the model to accurately classify these digits based on their pixel patterns.

For implementation, I utilized **NumPy** for handling array operations, while **TensorFlow** and **Keras** were employed for constructing, training, and optimizing the CNN model. Additionally, **Matplotlib** was used for visualizing training progress, accuracy, and loss curves.

To enhance my understanding of neural network behaviour, I experimented with various **activation functions**, including **ReLU**, **SeLU**, **Sigmoid**, and **Softmax**, to observe their impact on model performance. I also tested different **loss functions** and varied the **number of epochs** to analyse their influence on training efficiency, convergence speed, and overall accuracy.

Through this experimentation, I successfully developed a trained CNN capable of accurately recognizing handwritten digits. Moreover, this project provided valuable insights into how different hyperparameters and configurations influence the learning process of a neural network. These findings will serve as a foundation for future exploration into more complex deep learning architectures and applications.

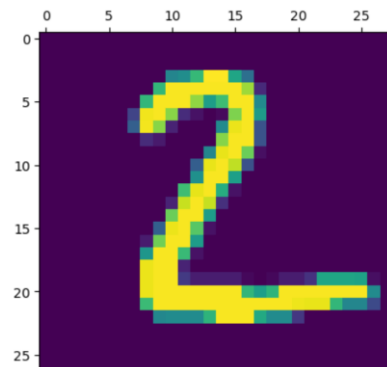
Code:

https://github.com/Random-homosapien/Computer_Vision_SOP/blob/main/Handwriting%20Trained%20model.ipynb

Execution:

```
plt.matshow(x_test[1])
```

```
<matplotlib.image.AxesImage at 0x2918a26a030>
```



Test image 1:

```
y_predicted = model.predict(x_test_flattened)
y_predicted[1]
```

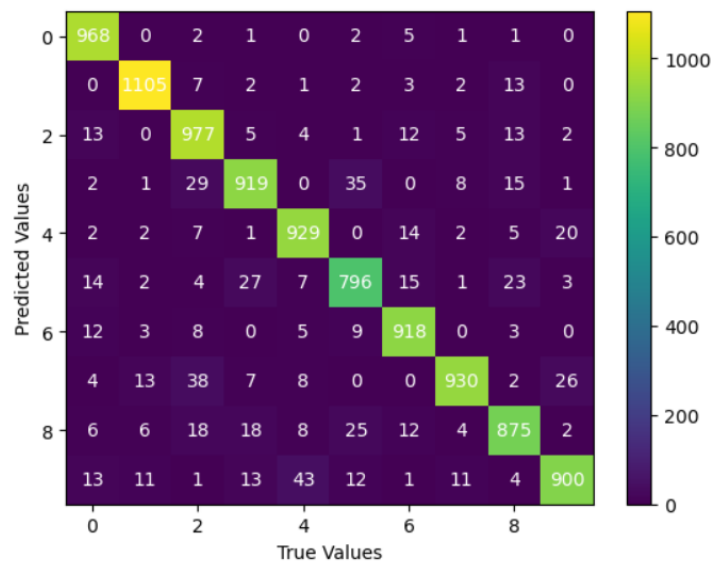
```
313/313 ————— 1s 2ms/step
```

```
array([6.2837452e-01, 9.9996912e-01, 1.0000000e+00, 9.9002779e-01,
       1.5029917e-04, 1.3142133e-02, 4.8387758e-02, 6.9904415e-04,
       8.4758860e-01, 1.9500018e-05], dtype=float32)
```

```
np.argmax(y_predicted[1])
```

```
2
```

1. Array with probability of each number for test image 1
2. Number with highest probability, i.e. predicted value



Confusion Matrix: For the given testing data,
how many *Predicted value* = a , were marked as *True value* = b

Chapter 2: Basics of Computer Vision:

OpenCV^[2] is one of the most widely used Python libraries for computer vision applications. Due to its extensive functionality and ease of integration, it has become a standard tool for tasks ranging from image processing to real-time object detection. To gain a foundational understanding of computer vision concepts, I developed a set of OpenCV-based programs aimed at exploring key functionalities such as face recognition, anonymization, and object measurement.

Theory:

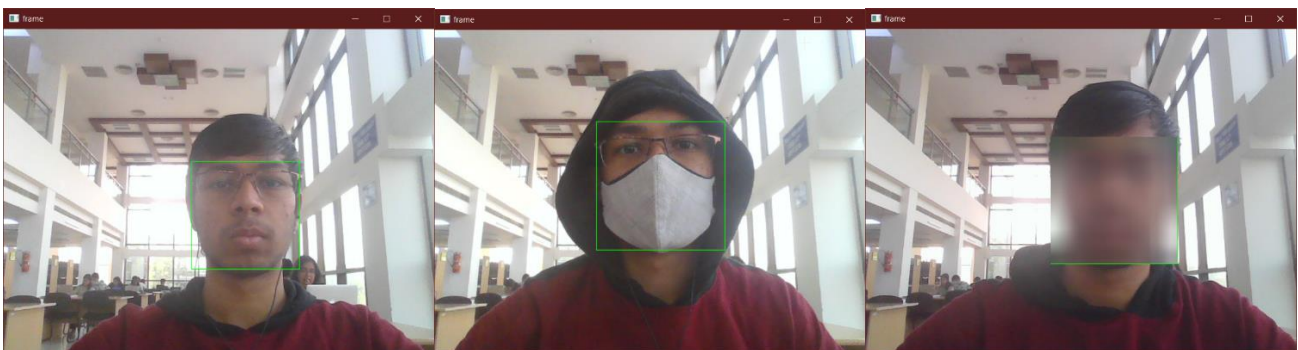
Face Recognizer and Anonymizer

The first project, titled "**Face Recognizer and Anonymizer**,"^[3] focuses on face detection and privacy preservation. OpenCV is utilized to detect human faces in an image or video frame by identifying key facial landmarks. Once a face is detected, the algorithm determines its four corner points and overlays a green rectangular bounding box around it for visualization. Additionally, a face-blurring feature is implemented to anonymize detected faces, ensuring privacy protection in images or video feeds.

This project leverages **Mediapipe**^[4], a machine learning framework, to import a pre-trained face detection model. OpenCV and NumPy are then used to extract facial coordinates, generate bounding boxes, and apply a Gaussian blur to obscure the face. This functionality is particularly useful in applications where facial privacy needs to be maintained, such as in surveillance footage or content anonymization for social media.

Code:

https://github.com/Random-homosapien/Computer_Vision_SOP/blob/main/Handwriting%20Trained%20model.ipynb



Face detection

Face detection with mask

Face Anonymisation

Conclusion

This project serves as an introduction to fundamental computer vision techniques using OpenCV and NumPy. Through face detection and anonymization, I have explored how computer vision can be applied to solve real-world problems. Moving forward, I aim to build more advanced models by incorporating deep learning techniques and expanding the scope of object detection and recognition.

Chapter 3: YOLO + SAM

YOLO:

YOLO (You Only Look Once)^[5] is a **Deep Learning (DL)-based algorithm** for real-time object detection. It uses a **single convolutional neural network (CNN)** to predict bounding boxes and class probabilities directly from an image in one forward pass.

- It is based on a **CNN architecture**, typically a modified version of Darknet, which is a deep neural network.
- It learns **features automatically** from data instead of relying on handcrafted features like traditional computer vision techniques.
- It uses **end-to-end deep learning**: the entire process—from feature extraction to bounding box prediction—is performed by a deep neural network.

Theory:

YOLO reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using this system, you only look once (YOLO) at an image to predict what objects are present and where they are.

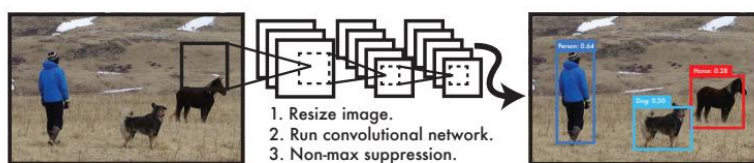


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

YOLO is extremely fast. The base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.

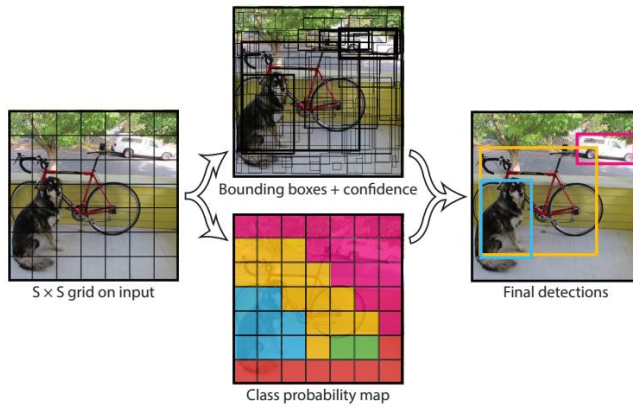
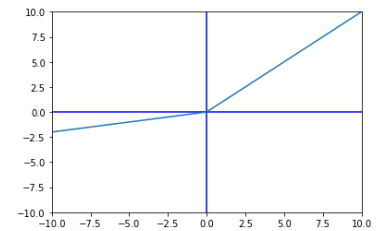


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

For evaluating YOLO on **PASCAL VOC**^[6], it uses $S = 7$, $B = 2$. PASCAL VOC has 20 labelled classes so $C = 20$. Their final prediction is a $7 \times 7 \times 30$ tensor.

They use a linear activation function for the final layer and all other layers use the following **leaky rectified linear activation** (Leaky RELU):

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$



The model is optimized for a type of **sum-squared error loss function**.

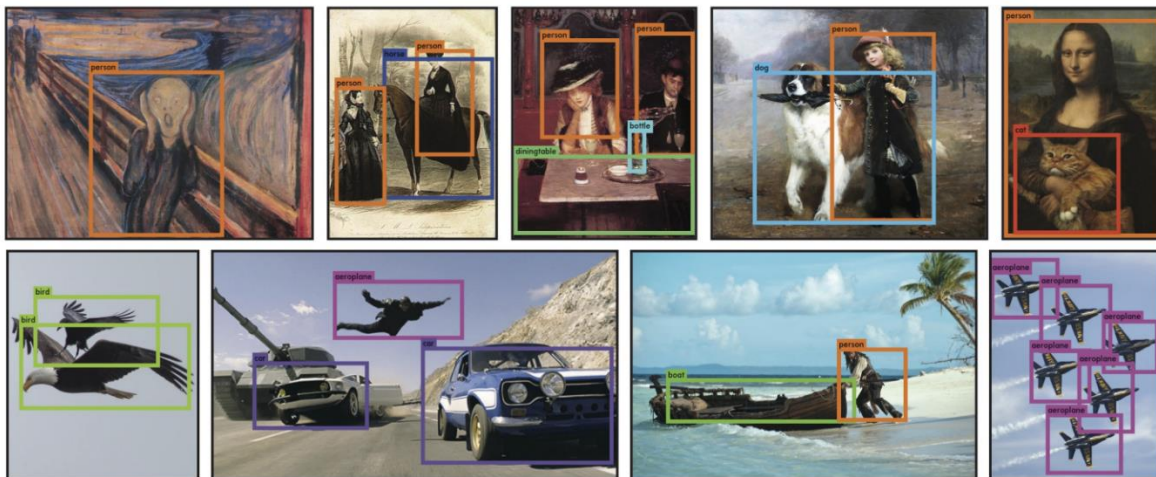


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

SAM:

The **Segment Anything Model (SAM)**^[7] is a **foundation model** for image segmentation developed by **Meta AI**. It is designed to "**segment anything**" in an image with minimal user input and is capable of generalizing to new objects without requiring additional training.

SAM only segments image but is incapable of identification or label assignment.

Theory:

SAM works on META AI dataset called **SA-1B**^[8] (**Segment Anything – 1 Billion**), which has 1 Billion masks and 11 Million images.

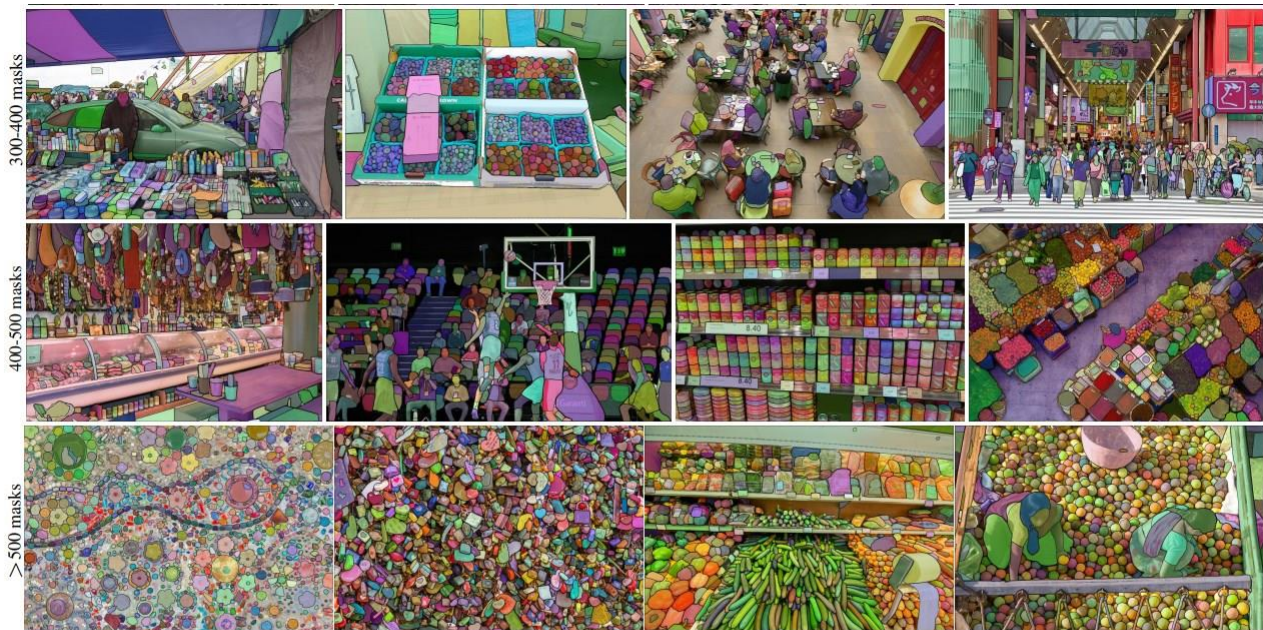


Figure 2: Example images with overlaid masks from our newly introduced dataset, **SA-1B**. SA-1B contains 11M diverse, high-resolution, licensed, and privacy protecting images and 1.1B high-quality segmentation masks. These masks were annotated *fully automatically* by SAM, and as we verify by human ratings and numerous experiments, are of high quality and diversity. We group images by number of masks per image for visualization (there are ~100 masks per image on average).

SAM introduces **promptable segmentation**, where the model generates a segmentation mask based on user input (such as a point, bounding box, or text). The task is designed to allow the model to generalize across various datasets and use cases, similar to **prompting in NLP models** (Natural Language Processing).

SAM consists of three main components:

1. **Image Encoder** – A **Vision Transformer (ViT)**^[9] processes the entire image to create an embedding.
2. **Prompt Encoder** – Converts user prompts (points, boxes, text) into embeddings.
3. **Mask Decoder** – Combines image and prompt embeddings to predict segmentation masks.

SAM can efficiently handle **multiple prompts** and **output multiple masks** when faced with ambiguous input. It operates in **real-time** and supports **interactive segmentation**.

SAM operates on SA-1B dataset. But this dataset wasn't built the normal way. The authors built a **data engine** specifically for creating SA-1B.

SA Data engine:

Assisted-manual stage: At the start of this stage, SAM was trained using common **public segmentation datasets**. Then a team of professional annotators **labelled new masks** by clicking foreground / background object points using a browser-based interactive segmentation tool powered by SAM. After sufficient data annotation, SAM was **retrained** using only **newly annotated masks**. (Only prominent objects were masked in this stage)

Semi-automatic stage:

To focus annotators on less prominent objects, we first automatically detected confident masks. Then we presented annotators with images prefilled with these masks and asked them to annotate any **additional unannotated objects**. To detect confident masks, we trained a **bounding box detector** on all first stage masks using a generic object category.

Fully automatic stage:

First, at the start of this stage, we had **collected enough masks** to greatly improve the model, including the diverse masks from the previous stage. Second, by this stage we had developed the **ambiguity-aware model**, which allowed us to predict valid masks even in ambiguous cases.

With the ambiguity-aware model, if a point lies on a part or subpart, our model will **return the subpart, part, and whole object**. (Thus creating multiple masks)

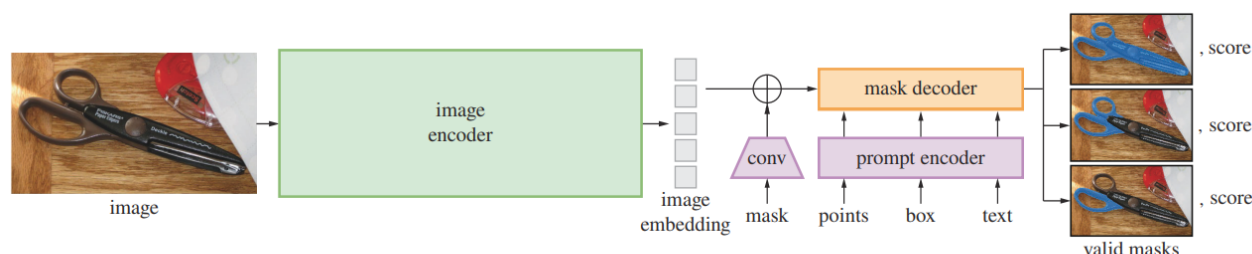


Figure 4: Segment Anything Model (SAM) overview. A heavyweight image encoder outputs an image embedding that can then be efficiently queried by a variety of input prompts to produce object masks at amortized real-time speed. For ambiguous prompts corresponding to more than one object, SAM can output multiple valid masks and associated confidence scores.

Experimentation:

In this model, I integrated the Segment Anything Model (SAM) with YOLO to leverage the strengths of both approaches in object detection and segmentation. YOLO (You Only Look Once) is well known for its ability to **detect objects with high accuracy across a diverse range of classes**. It identifies objects in an image, assigns a class label, and generates a **bounding box** around each detected object.

SAM then takes these bounding boxes as input and produces precise segmentation masks for the identified objects. One of SAM's key advantages is its superior performance in **segmentation**, particularly in **zero-shot learning (ZSL)**^[10]. ZSL refers to the model's ability to segment objects that were **not explicitly included in the training dataset**, making SAM highly adaptable to novel object categories.

By combining YOLO's robust object detection capabilities with SAM's advanced segmentation performance, this approach achieves **accurate and reliable object segmentations**, even for previously **unseen objects**. This integration enhances the model's **generalization ability** and applicability across a wide range of real-world scenarios.

I also used this model to experiment with cloud computing and its effectiveness in model training. **Google-colab** is a cloud computing service that allows users to write, execute, and share Python code in an interactive **Jupyter Notebook environment**, with free access to cloud-based hardware, including **GPUs (Graphics Processing Units)** and **TPUs (Tensor Processing Units)**.

Code:

https://colab.research.google.com/drive/14Uu1b9_vHPA9J1wt4vmxHnk615zicn1c?usp=sharing

or

https://github.com/Random-homosapien/Computer_Vision_SOP/tree/main/YOLO%20%2B%20SAM

Results:

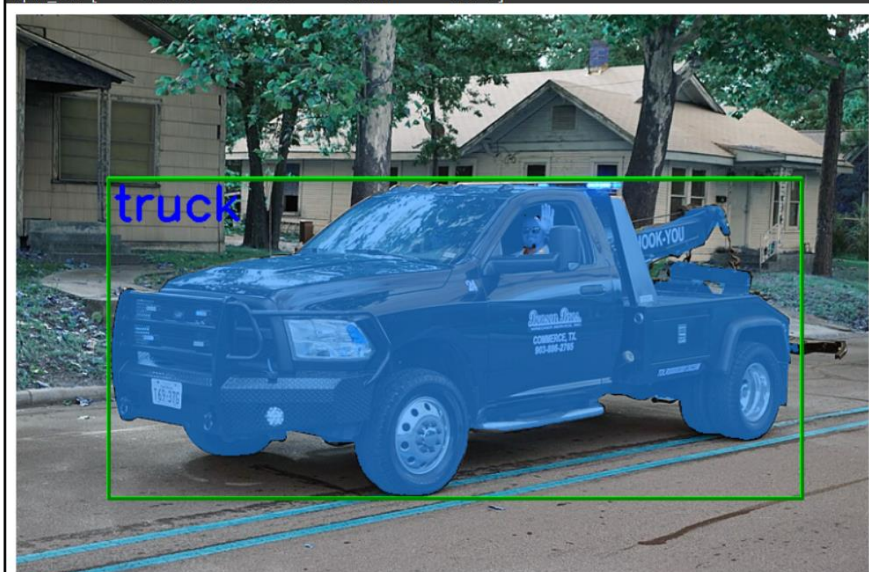


ORIGINAL IMAGE



YOLO OUTPUT == SAM INPUT

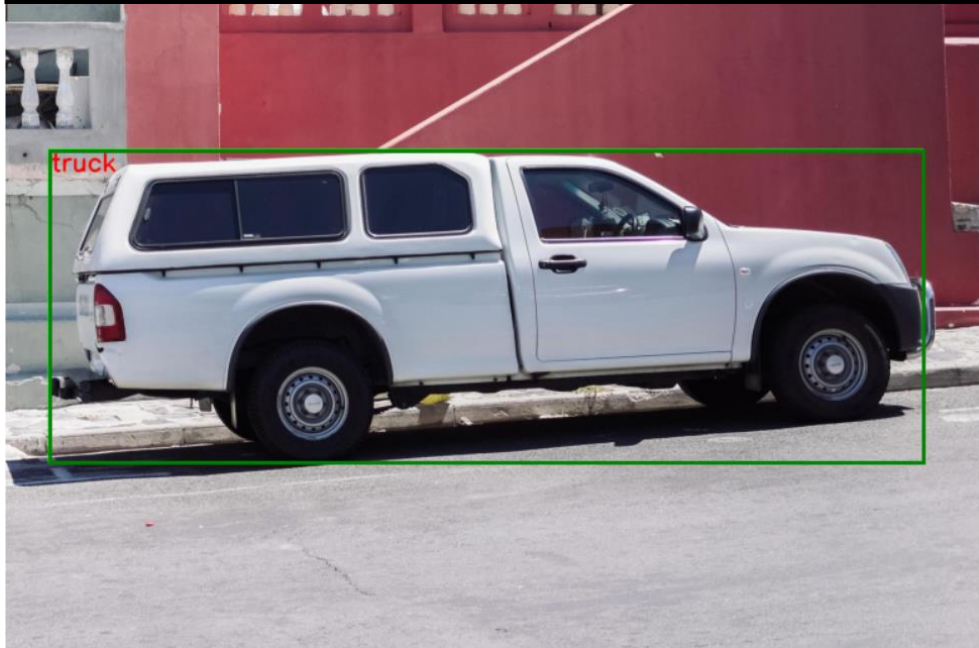
input box [86.574 154.99 736.19 454.45]



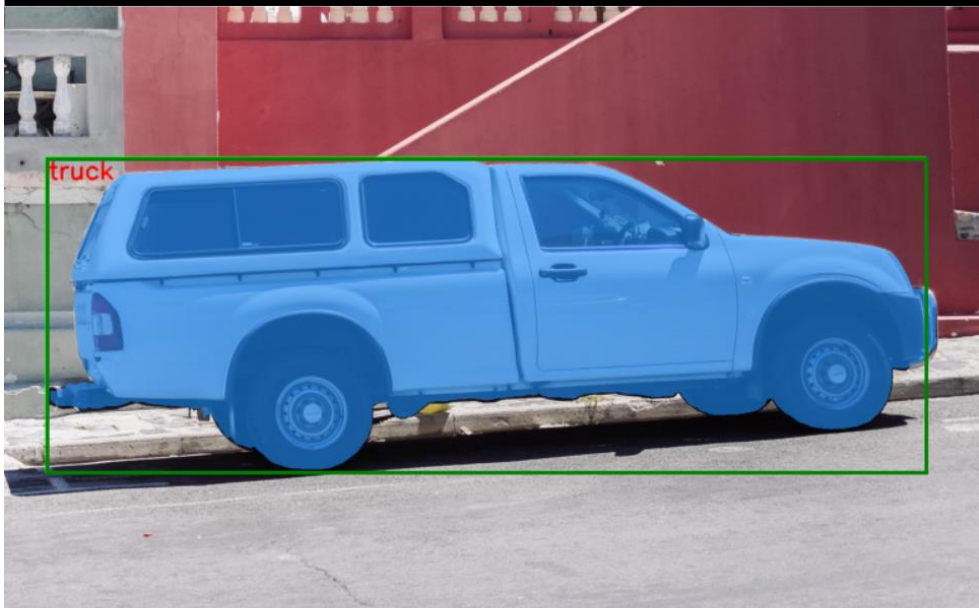
SAM OUTPUT



ORIGINAL IMAGE



YOLO OUTPUT == SAM INPUT



SAM OUTPUT

Chapter 4: RANSAC + PNP (DPOD)

Introduction:

The paper presents **DPOD (Dense Pose Object Detector)**^[11], a novel deep learning-based approach for **3D object detection and 6D pose estimation** from RGB images. Unlike traditional methods that rely on real-world training data, DPOD effectively uses both **synthetic and real training data** for superior performance.

Theory:

Data Preparation:

Both synthetic and real data is used. 3D models are rendered from multiple viewpoints to create training images. These act as synthetic data. Objects are also segmented from real-world datasets and augmented for training.

Objects are assigned **UV texture maps**, creating a bijective mapping between **2D image pixels** and **3D model vertices** on the correspondence map^[11.1].

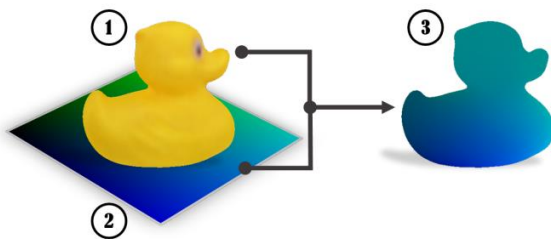


Figure 3: **Correspondence model:** Given a 3D model of interest (1), we apply a 2 channel correspondence texture (2) to it. The resulting correspondence model (3) is then used to generate GT maps and estimate poses.

The final stage of data preparation is the online data generation, which is responsible for providing full-sized RGB images ready for training. Generated patches (real or synthetic) are rendered on top of images from MS COCO dataset^[12] producing training images containing multiple objects.

Dense Object Detection Pipeline:

Correspondence Block (Encoder-Decoder CNN): Predicts **object ID masks**^[11.2] and dense 2D-3D correspondence maps from an RGB image.

Pose Block : The pose block is responsible for the pose prediction. Given the estimated **ID mask**, creates 3D models and uses **PnP + RANSAC** to estimate the 6D pose from correspondences.

Finally it is passed through a pose refiner CNN to get final output.

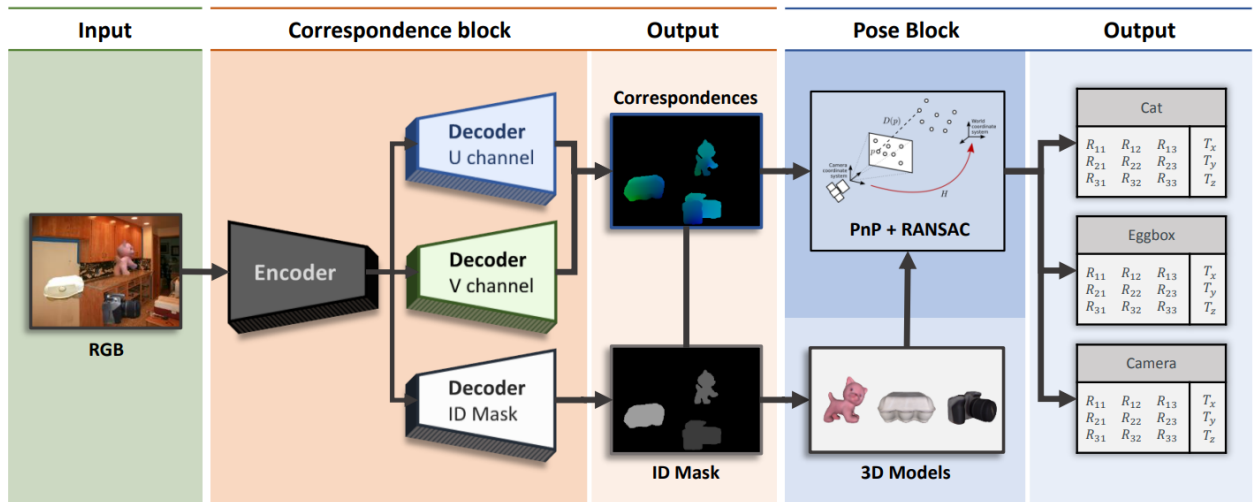
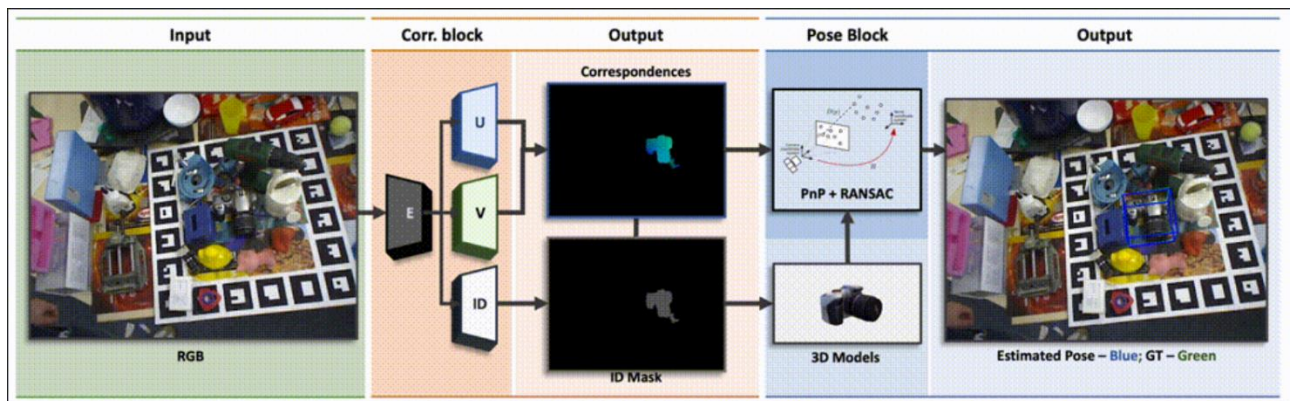


Figure 2: **Pipeline description:** Given an input RGB image, the correspondence block, featuring an encoder-decoder neural network, regresses the object ID mask and the correspondence map. The latter one provides us with explicit 2D-3D correspondences, whereas the ID mask estimates which correspondences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

[11.1] A **correspondence map** is a type of output produced by a deep learning model that establishes a **mapping between 2D image pixels and 3D model vertices**. Essentially, it helps in finding which part of a 3D object corresponds to a specific pixel in a given 2D image.

[11.2] An **ID mask** is a segmentation mask where each pixel is assigned a **class ID** corresponding to a detected object. This helps the system distinguish between different objects in an image.

Code: <https://github.com/zakharos/DPOD>



Experimentation:

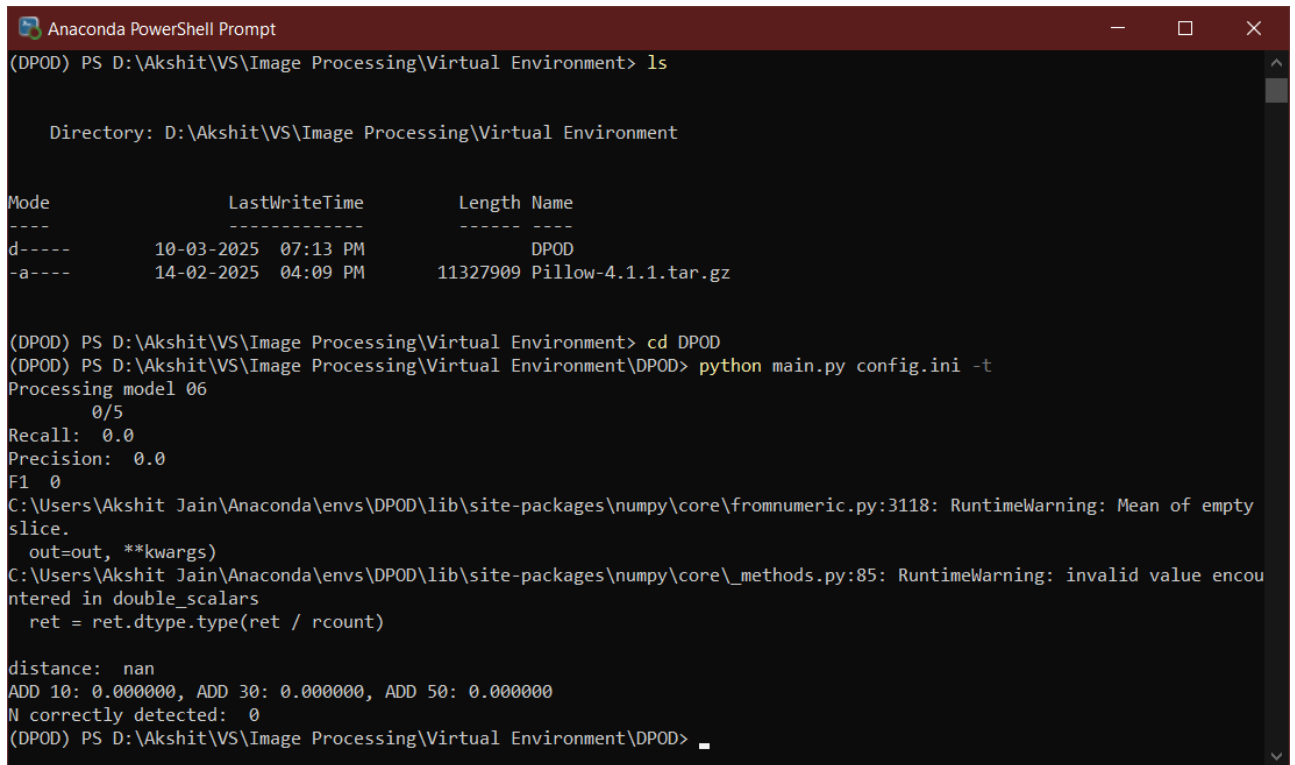
1. Setting up venv:

Virtual Environment is setup and all modules and libraries are installed. Specific versions of libraries are installed which were used when creating the DPOD code.

Certain versions of libraries were discontinued/ not working. Therefore appropriate replacements were used. (closest version).

2. Checking Configuration:

Checks if all libraries are installed or not



```
Anaconda PowerShell Prompt
(DPOD) PS D:\Akshit\VS\Image Processing\Virtual Environment> ls

Directory: D:\Akshit\VS\Image Processing\Virtual Environment

Mode                LastWriteTime         Length Name
----                -
d-----          10-03-2025   07:13 PM             DPOD
-a-----          14-02-2025   04:09 PM    11327909 Pillow-4.1.1.tar.gz

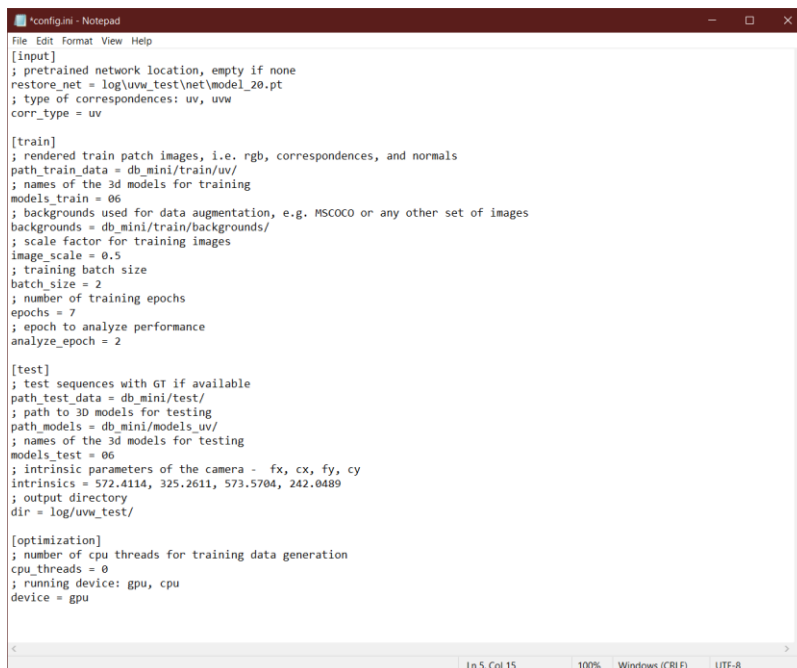
(DPOD) PS D:\Akshit\VS\Image Processing\Virtual Environment> cd DPOD
(DPOD) PS D:\Akshit\VS\Image Processing\Virtual Environment\DPOD> python main.py config.ini -t
Processing model 06
0/5
Recall: 0.0
Precision: 0.0
F1 0
C:\Users\Akshit Jain\Anaconda\envs\DPOD\lib\site-packages\numpy\core\fromnumeric.py:3118: RuntimeWarning: Mean of empty
slice.
  out=out, **kwargs)
C:\Users\Akshit Jain\Anaconda\envs\DPOD\lib\site-packages\numpy\core\_methods.py:85: RuntimeWarning: invalid value encou
ntered in double_scalars
  ret = ret.dtype.type(ret / rcount)

distance: nan
ADD 10: 0.000000, ADD 30: 0.000000, ADD 50: 0.000000
N correctly detected: 0
(DPOD) PS D:\Akshit\VS\Image Processing\Virtual Environment\DPOD> _
```

Checking Configuration (Successful)

3. Checking config.ini file:

Check to see what all parameters are set for training and testing



```
config.ini - Notepad
File Edit Format View Help
[input]
; pretrained network location, empty if none
restore_net = log\uvw_test\net\model_20.pt
; type of correspondences: uv, uvw
corr_type = uv

[train]
; rendered train patch images, i.e. rgb, correspondences, and normals
path_train_data = db_mini/train/uv/
; names of the 3d models for training
models_train = 06
; backgrounds used for data augmentation, e.g. MSCOCO or any other set of images
backgrounds = db_mini/train/backgrounds/
; scale factor for training images
image_scale = 0.5
; training batch size
batch_size = 2
; number of training epochs
epochs = 7
; epoch to analyze performance
analyze_epoch = 2

[test]
; test sequences with GT if available
path_test_data = db_mini/test/
; path to 3d models for testing
path_models = db_mini/models_uv/
; names of the 3d models for testing
models_test = 06
; intrinsic parameters of the camera - fx, cx, fy, cy
intrinsics = 572.4114, 325.2611, 573.5704, 242.0489
; output directory
dir = log\uvw_test/

[optimization]
; number of cpu threads for training data generation
cpu_threads = 0
; running device: gpu, cpu
device = gpu
```

Config.ini (Reference file using which model is trained)

4. Training Model on local device:

Current device doesn't have GPU. Also CPU is subpar. Therefore lab PC used which had GPU.
(Performed steps 1-3 again on new device.)

5. Training the model:

- Model is trained on the miniature dataset provided (db_mini).
- No log files created for training model

	Precision	Recall	F1	ADD 10%	ADD 30%	ADD 50%
6	0	0	0	0	0	0
Log File						

- No output files generated

6. Testing the model:

Unsuccessful testing

```
Anaconda PowerShell Prompt
(base) PS C:\Users\Akshit Jain> cd "D:\Akshit\VS\Image Processing"
(base) PS D:\Akshit\VS\Image Processing> conda activate DPOD
(DPOD) PS D:\Akshit\VS\Image Processing> cd "Virtual Environment"
(DPOD) PS D:\Akshit\VS\Image Processing\Virtual Environment> cd DPOD
(DPOD) PS D:\Akshit\VS\Image Processing\Virtual Environment\DPOD> python main.py --test config.ini
Processing model 06
0/5
Recall: 0.0
Precision: 0.0
F1 0
C:\Users\Akshit Jain\Anaconda\envs\DPOD\lib\site-packages\numpy\core\fromnumeric.py:3118: RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
C:\Users\Akshit Jain\Anaconda\envs\DPOD\lib\site-packages\numpy\core\_methods.py:85: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
distance: nan
ADD 10: 0.000000, ADD 30: 0.000000, ADD 50: 0.000000
N correctly detected: 0
(DPOD) PS D:\Akshit\VS\Image Processing\Virtual Environment\DPOD>
```

Possible Reasons for Failure:

1. Incorrect or Incompatible Library Versions

Since I installed the closest available versions of libraries, some functionalities may be missing or behave differently.

- A small change in **TensorFlow, PyTorch, or OpenCV** can cause models to behave unexpectedly.
- Some older models require specific versions of **CUDA, cuDNN, or other dependencies**.

2. Model Checkpointing or Saving Failed

Your model may have trained but failed to save its weights due to:

- **Incorrect file paths or permissions** (e.g., trying to save to a non-existent directory).

- **Code that does not save the model properly** (e.g., missing `model.save()` in TensorFlow or `torch.save()` in PyTorch).

Checked for `torch.save()` in the files.

Checked for permissions.

Checked for file path. Found location for file storage, no output stored there.

3. Silent Exit Due to Memory Issues:

If training stopped without error logs, it could be due to:

- **Out of memory (OOM)** errors on GPU or RAM.
- A system kill signal due to **excessive resource usage**.

Debugged the file for any out of memory messages

Scaled down the images, reduced the batch sizes, changed number of epoch

4. Undetectable error in code:

- Maybe there was error in the code provided in the Github repository.

Manually check the code to the best of your knowledge to see if you can detect any discrepancies in the code. None found

Therefore,

Due to possible incompatible library versions or undetectable error in code, I couldn't generate an output for DPOD (Dense Pose Object Detector).

Next I tried to access the pre-generated output but that had been removed by the authors.

I also reached out to the authors of the research paper to request provision of pre-generated output but to no avail.

Conclusion:

DPOD introduces a **dense correspondence-based 6D pose estimation approach**, outperforming prior methods in **accuracy, efficiency, and robustness**. Its ability to train on both **synthetic and real data** makes it highly adaptable for real-world use in **robotics, augmented reality, and industrial automation**.

I may continue to explore DPOD later due to its advantages and based on my needs.

Chapter 5: PoseCNN

Introduction:

PoseCNN^[13], a new end-to-end CNN-based approach designed for 6D object pose estimation. It breaks down the Pose Estimation task into 3 different problems of segmentation, 3D translation and 3D rotation estimation.

Theory:

PoseCNN combines template-based and feature-based methods into a deep learning framework.

1. Template-based methods – These rely on pre-defined object templates but struggle with occlusions.
2. Feature-based methods – These use image feature matching but fail for objects with low texture.

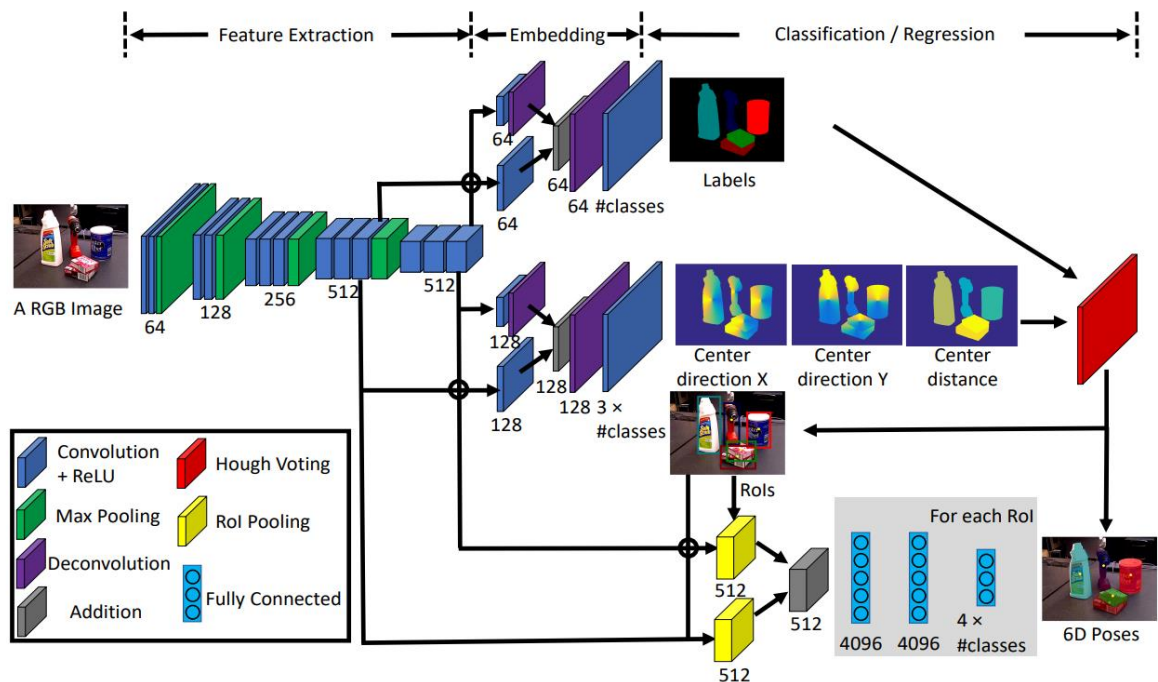


Fig. 2. Architecture of PoseCNN for 6D object pose estimation.

The PoseCNN uses a regression model which decouples pose estimation into three separate tasks:

Semantic Labelling^[14]:

The semantic labelling module classifies each pixel in an image to determine which object it belongs to. This method is more robust than traditional bounding box-based detection.

3D Translation Estimation:

PoseCNN finds an object's 3D location by:

1. Estimating its 2D center in the image.
2. Predicting the depth from the camera.
3. Using a Hough voting mechanism to improve robustness.

3D Rotation Regression:

PoseCNN estimates 3D rotation using quaternions instead of Euler angles to avoid singularities.

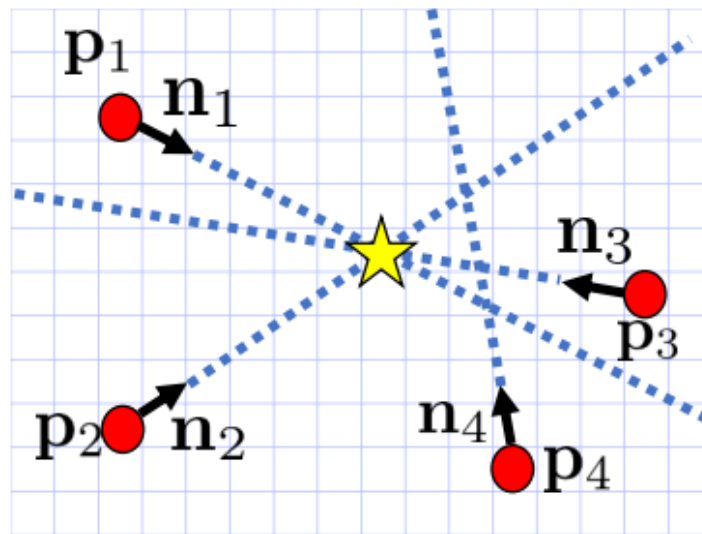


Fig. 4. Illustration of Hough voting for object center localization: Each pixel casts votes for image locations along the ray predicted from the network.

Loss functions:

To train the quaternion regression, they propose two loss functions:

The first loss, called PoseLoss (PLOSS), operates in the 3D model space and measures the average squared distance between points on the correct model pose and their corresponding points on the model using the estimated orientation.

PLOSS does not handle symmetric objects appropriately, since a symmetric object can have multiple correct 3D rotations.

So they also introduce a new loss function called ShapeMatch-Loss to handle symmetric objects. ShapeMatch-Loss (SLOSS) is a loss function that does not require the specification of symmetries.

This loss measures the offset between each point on the estimated model orientation and the closest point on the ground truth model. SLOSS is minimized when the two 3D models match each other.

SLOSS will not penalize rotations that are equivalent with respect to the 3D shape symmetry of the object.

Code: <https://rse-lab.cs.washington.edu/projects/posecnn/>

Experimentation:

Due to the following reasons, I was unable to execute PoseCNN:

- GCC version used by them was outdated:
build fail with cuda: identifier "__builtin_ia32_mwaitx" is undefined

build fail with cuda: identifier "__builtin_ia32_monitorx" is undefined
Couldn't find satisfactory solution without altering integral gcc files

- Ubuntu 16.02 unsupportive of later CUDA and TF versions:
Uses a pip version which can install later versions of these libraries
- Provided weights/pre-trained model were not found:
"Download the VGG16 weights from [here](#) (528M)"
No weights stored in the link
- Cmake had problem with OpenGL paths:
CMake Pangolinm requires OpenGL.
Installs OpenGL again and still its shown to be missing.
- Python 3.5 version can't configure F'strings:
Pip3, CUDA and Tensorflow along with many newer libraries use F'strings which were introduced in Python 3.8 and show error for python 3.5 version in our environment

Conclusion:

PoseCNN presents a robust, deep learning-based framework for 6D object pose estimation by decomposing the problem into three distinct sub-tasks: semantic segmentation, 3D translation estimation, and 3D rotation prediction. By introducing the ShapeMatch-Loss to address object symmetries, PoseCNN successfully overcomes key challenges faced by traditional template-based and feature-based methods. These 2 properties have also paved the way for Efficient Pose to develop.

Although practical execution was hindered by compatibility issues related to outdated software environments, the theoretical contributions and architectural innovations of PoseCNN establish it as a significant advancement in pose estimation, with strong applicability in real-world scenarios such as robotic manipulation and augmented reality.

Chapter 6: EfficientPose

Introduction:

EfficientPose^[15], a new approach for 6D object pose estimation. Our method is highly accurate, efficient and scalable over a wide range of computational resources. Moreover, it can detect the 2D bounding box of multiple objects and instances as well as estimate their full 6D poses in a single shot. This eliminates the significant increase in runtime when dealing with multiple objects.

Theory:

EfficientPose extends the EfficientDet^[16] architecture by introducing two additional subnetworks for predicting 3D rotation and translation, enabling end-to-end estimation of 6D poses. (Just like PoseCNN)

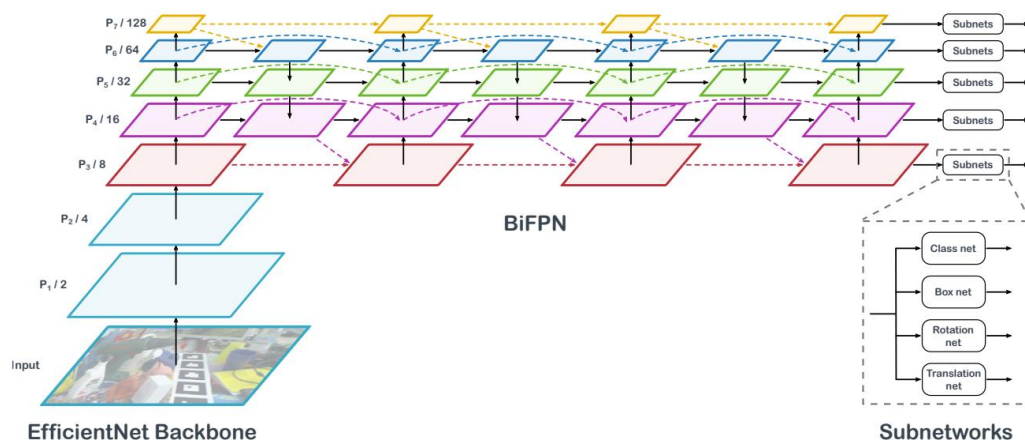


Figure 2. Schematic representation of our EfficientPose architecture including the EfficientNet^[37] backbone, the bidirectional feature pyramid network (BiFPN) and the prediction subnetworks.

EfficientPose handles multiple objects and instances simultaneously in a single shot. This approach eliminates the need for post-processing steps such as RANSAC or PnP, thereby reducing runtime complexity

EfficientPose is efficient to the level of Real-Time detection, i.e. it can detect poses even for videos and web-cameras

Key Features

1. End-to-End Architecture:
 - EfficientPose integrates 2D object detection with 6D pose estimation into a single framework.
 - It adds two subnetworks to EfficientDet: one for rotation prediction and another for translation prediction.
 - These subnetworks share computational resources with existing classification and bounding box regression networks, minimizing computational overhead.

2. Single-Shot Multi-Object Pose Estimation:
 - The architecture can detect multiple objects and instances in one pass, making it suitable for scenarios involving cluttered environments or multiple objects.
 - The runtime remains nearly independent of the number of objects due to its direct regression approach.
3. Direct Pose Regression:
 - Rotation is predicted using an axis-angle representation combined with iterative refinement layers to enhance accuracy.
 - Translation is computed by predicting the 2D center point of the object in pixel coordinates and its depth, which are then converted into full 3D translation using intrinsic camera parameters.
4. 6D Augmentation:
 - EfficientPose introduces a data augmentation technique that adjusts ground truth poses to match image transformations like rotation and scaling.
 - This method compensates for limitations in small datasets by generating diverse training samples without mismatches between images and annotated poses.

It is also completely scalable via a single hyperparameter Φ allowing it to adapt to various computational resource constraints while maintaining high accuracy.

Theoretically this approach can be scaled from $\phi = 0$ to $\phi = 7$ in integer steps but due to computational constraints, the author only used $\phi = 0$ and $\phi = 3$ in their experiments. I on the other hand only trained $\phi = 0$ on my own as the training time for $\phi = 3$ was exponentially greater than $\phi = 0$ and wasn't feasible to be trained by me.

Code: <https://github.com/ybkscht/EfficientPose>

Execution:

1. Direct execution of given pretrained model:

- ▶ Got the same Accuracy for both Linemod^[17] and OCCLUSION datasets as the author
- ▶ Output: mAP: 1.0000
 ADD: 0.9633
 ADD-S: 1.0000
 5cm_5degree: 0.4727
 TranslationErrorMean_in_mm: 10.5903
 TranslationErrorStd_in_mm: 6.5488
 RotationErrorMean_in_degree: 5.9470
 RotationErrorStd_in_degree: 3.5110
 2D-Projection: 0.7047
 Summed_Translation_Rotation_Error: 26.5971
 ADD(-S): 0.9633
 AveragePointDistanceMean_in_mm: 13.0365
 AveragePointDistanceStd_in_mm: 6.0336
 AverageSymmetricPointDistanceMean_in_mm: 7.6579
 AverageSymmetricPointDistanceStd_in_mm: 2.3956
 MixedAveragePointDistanceMean_in_mm: 13.0365
 MixedAveragePointDistanceStd_in_mm: 6.0336

2. Training on Local device:

- ▶ Loaded pretrained weights and ran training on OCCLUSION ($\Phi = 1$). Edited epoch, Learning rate to different values.
- ▶ For LR = 1e-5, and epoch = 150, model converged at epoch 82 with improved weights
- ▶ ADD(-S) improved from 0.79572 to 0.79869
- ▶ Note: Pretrained weights stored at https://github.com/Random-homosapien/Computer_Vision_SOP under EfficientPose/weights

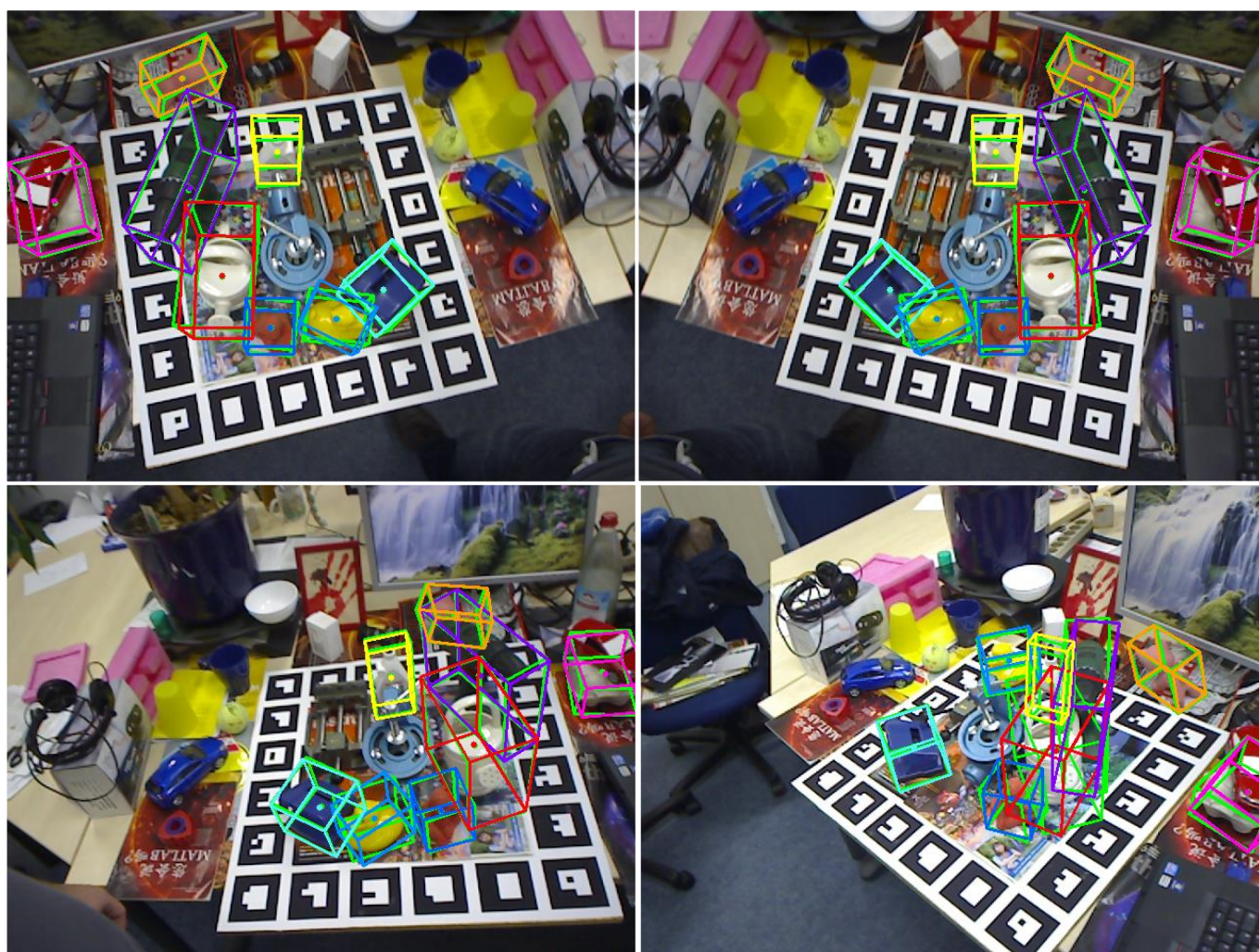
▶ RESULT:

990 instances of class ape with ADD(-S)-Accuracy: 0.5545 (lower)
1025 instances of class can with ADD(-S)-Accuracy: 0.9288 (higher)
1009 instances of class cat with ADD(-S)-Accuracy: 0.6997 (higher)
1031 instances of class driller with ADD(-S)-Accuracy: 0.9573 (higher)
960 instances of class duck with ADD(-S)-Accuracy: 0.6677 (higher)
994 instances of class eggbox with ADD(-S)-Accuracy: 0.9416 (higher)
761 instances of class glue with ADD(-S)-Accuracy: 0.8607 (higher)
1014 instances of class holepuncher with ADD(-S)-Accuracy: 0.7791 (higher)
General ADD(-S)-Accuracy: 0.7987

Therefore:

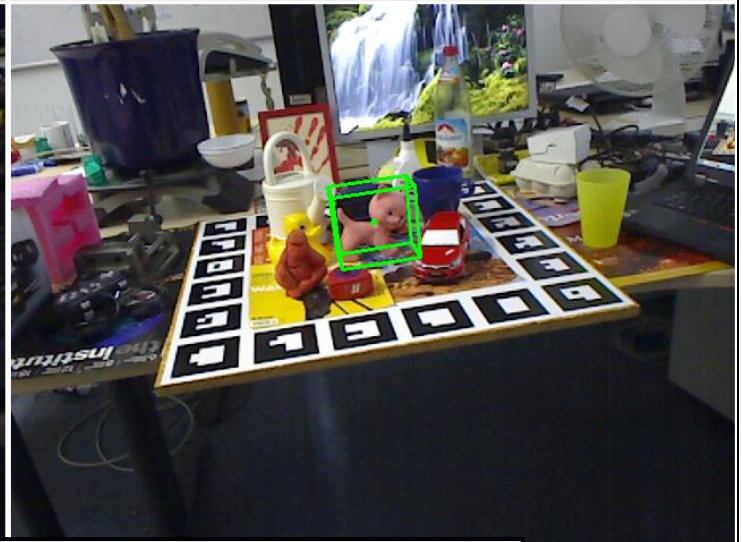
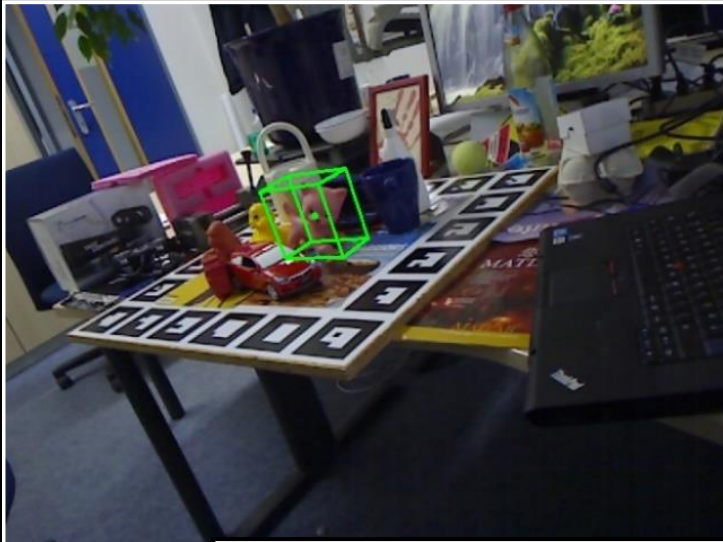
Occlusion ADD(-S) Accuracy:

CLASS	AUTHOR'S ACCURACY	MY ACCURACY
APE	0.5657	0.5545 (Lower)
CAN	0.9112	0.9288 (Higher)
CAT	0.6858	0.6997 (Higher)
DRILLER	0.9564	0.9573 (Higher)
DUCK	0.6531	0.6677 (Higher)
EGGBOX	0.9346	0.9416 (Higher)
GLUE	0.8515	0.8607 (Higher)
HOLEPUNCHER	0.7653	0.7791 (Higher)
GENERAL	0.7904	0.7987 (Higher)

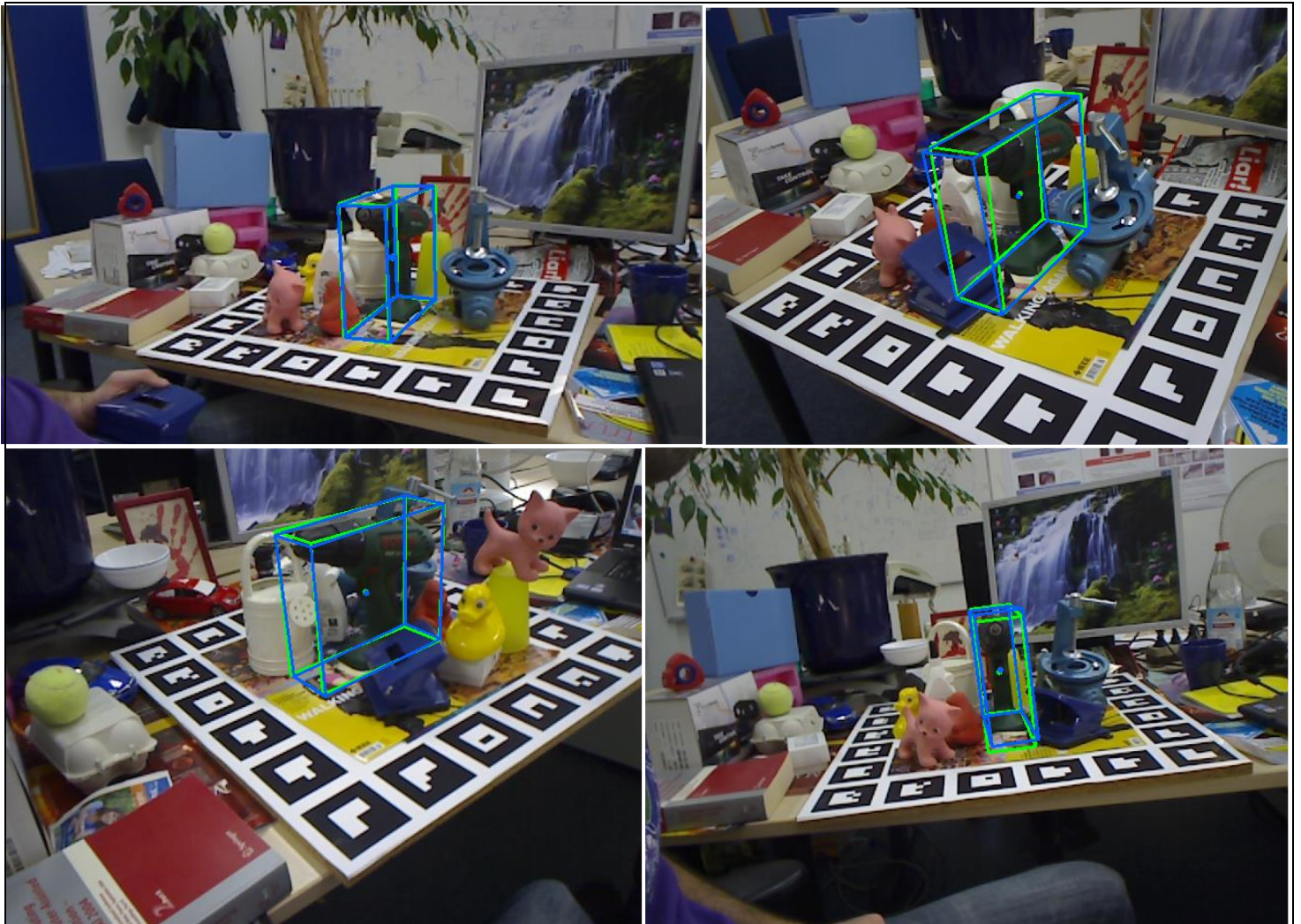


Linemod Dataset:

- ▶ Once again, set epoch to 150, Learning Rate to $1e-5$
- ▶ Converged at epoch 25
- ▶ ADD improved from 0.97106 to 0.97405



Linemode (class: 6) Cat



Linemode (class: 8) Driller

Conclusion:

EfficientPose extends the EfficientDet framework to enable end-to-end 6D object pose estimation by integrating dedicated subnetworks for 3D rotation and translation prediction. Its single-shot, multi-object capability and direct regression approach eliminate the computational overhead associated with post-processing techniques like RANSAC and PnP. Experimental validation demonstrated the model's high accuracy across challenging datasets, with further improvements achieved through fine-tuning. EfficientPose's real-time performance, scalability via the Φ parameter, and adaptability to diverse computational resources mark it as a significant contribution to the field of efficient and scalable pose estimation for dynamic and cluttered environments.

Conclusion: (For the whole project)

This project provided an in-depth exploration into the field of object localization and pose estimation, combining traditional computer vision techniques with cutting-edge deep learning models. Initial work on convolutional neural networks established a strong theoretical foundation, which was subsequently extended through the implementation and analysis of object detection and segmentation models such as YOLO and SAM. Classical algorithms like RANSAC and PnP were revisited to appreciate their significance in pose estimation tasks.

Advanced models such as DPOD, PoseCNN, and EfficientPose were investigated to understand the evolution toward end-to-end learning architectures capable of robust 6D pose estimation. Although certain challenges, particularly environmental and compatibility issues, hindered the practical execution of some models, they nonetheless offered valuable insights into real-world deployment complexities. Notably, EfficientPose demonstrated excellent real-time accuracy and scalability, highlighting the ongoing shift towards highly efficient, single-shot pose estimation frameworks.

The knowledge and technical skills acquired through this project have enhanced my understanding of both theoretical and practical aspects of computer vision. They will serve as a strong foundation for future research and application development in areas such as autonomous robotics, augmented reality, industrial automation, and beyond. This project not only deepened my expertise in deep learning-based vision systems but also strengthened my capability to troubleshoot and adapt in the face of evolving technological challenges.

The techniques explored in this project have significant applications in fields such as autonomous robotics, augmented reality, medical imaging, and industrial automation. Moving forward, further refinement of object detection and pose estimation methods can enhance accuracy and efficiency. The knowledge and experience gained through this study will serve as a stepping stone for deeper exploration into advanced vision models and their real-world applications.

References

1. <https://arxiv.org/abs/1511.08458v2>
An Introduction to Convolutional Neural Networks: Keiron O'Shea, Ryan Nash
2. https://www.researchgate.net/publication/301590571_OpenCV_for_Computer_Vision_Applications
A Brief introduction to OpenCV | IEEE
3. <https://www.youtube.com/watch?v=DRMBghrfxXg>
Face detection and blurring with Python and OpenCV | Computer vision tutorial
4. <https://ai.google.dev/edge/mediapipe/solutions/guide>
Mediapipe Documentation
5. <https://arxiv.org/abs/1506.02640>
You Only Look Once: Unified, Real-Time Object Detection: Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi
6. <https://link.springer.com/article/10.1007/s11263-014-0733-5>
Pascal Visual Object Classes Challenge: A Retrospective
7. <https://arxiv.org/abs/2304.02643>
Segment Anything: Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, Ross Girshick
8. <https://ai.meta.com/datasets/segment-anything/>
SA-1B Dataset: Meta
9. <https://arxiv.org/abs/2010.11929>
An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale: Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby
10. <https://arxiv.org/abs/2305.00109>
Zero-shot performance of the Segment Anything Model (SAM) in 2D medical imaging: A comprehensive evaluation and practical guidelines: Christian Mattjie, Luis Vinicius de Moura, Rafaela Cappelari Ravazio, Lucas Silveira Kupssinskü, Otávio Parraga, Marcelo Mussi Delucis, Rodrigo Coelho Barros
11. <https://arxiv.org/abs/1902.11020>
DPOD: 6D Pose Object Detector and Refiner: Sergey Zakharov, Ivan Shugurov, Slobodan Ilic
12. <https://arxiv.org/abs/1405.0312>
Microsoft coco: Common objects in context

13. <http://arxiv.org/abs/1711.00199>
PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes:
Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, Dieter Fox.
14. <http://arxiv.org/abs/1411.4038>
Fully Convolutional Networks for Semantic Segmentation: Jonathan Long, Evan Shelhamer, Trevor Darrell
15. <http://arxiv.org/abs/2011.04307>
EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach: Yannick Bukschat, Marcus Vetter
16. <http://arxiv.org/abs/1905.11946>
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks: Mingxing Tan, Quoc V. Le
17. http://link.springer.com/10.1007/978-3-642-37331-2_42
Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes: Stefan Hinterstoisser, Stefan Holzer, Vincent Lepetit