6D pose estimation of ground vehicles using a single monocular RGB image with Deep Learning

In collaboration with Swordfish Computing

Sharath Suresh Kumar
University of Adelaide
Adelaide, South Australia
a1768686@student.adelaide.edu.au

Brian Du- Supervisor
Swordfish Computing
Adelaide, South Australia
brian.du@swordfish.com.au

Mingyu Guo- Supervisor
Universiy of Adelaide
Adelaide, South Australia
mingyu.guo@adelaide.edu.au

ABSTRACT

6D DoF pose estimation from monocular RGB images is a fundamental task for various fields of computer vision. We establish the state of current pose-estimation methods,current evaluation metrics and datasets. We also implement EfficientPose- which is a fully end-to-end trainable Pose Estimation model built on EfficientNet. EfficientPose showed good performance in learning and estimating pose from our provided dataset. However, care must be taken not to overtrain it. This project serves as an exploration of using an end-to-end deep learning model to estimate pose.

1 INTRODUCTION AND MOTIVATION

Pose estimation is the process of obtaining the transformation of an object, relative to a camera or another object. Inferring the pose of an object from a 2D image is an interesting challenge, since it requires constructing 3D information from a 2D image.

6 Degrees of Freedom (DoF) is the 'absolute pose' of an object in a defined space, since it provides the rotational and translational properties of an object- roll, pitch, yaw, x, y, z. Thus estimating the 6D pose of a ground vehicle using images is an incredibly powerful tool- for instance allowing an autonomous vehicle to detect and avoid other vehicles on the road from just a RGB camera feed.

Although LiDAR (Light Detection and Ranging) is a well-proven technology for measuring distance and creating accurate 3D maps[10], LiDAR technology is expensive and bulky- making them unfeasible for mass adoption in smaller drones and cameras[10]. In contrast, monocular cameras that capture 2D RGB images, are inexpensive, light and are already utilised in many systems[23]. Hence, there is considerable interest in estimating pose using a single 2D RGB image from a monocular camera.

Hence, the focus of this project is to interpret the 6DoF pose of ground vehicles from a single monocular RGB image using an end-to-end pose estimation model. We shall provide an overview of key terms, current industry standards, a review of State of the Art (SoA) systems, and a potential gap to explore.

2 LITERATURE REVIEW

Pose estimation algorithms can be separated into two broad categories- Traditional methods and Deep Learning Methods. Traditional methods usually consist of using markers, templates or global/local features to estimate pose [2]. These methods are usually computationally expensive, and are greatly affected by occlusion and low texture images [23]. Traditional methods outperformed Deep Learning methods at first- but recent comparisons show that deep Learning methods outperform traditional methods, and are more robust in their application [7].

Deep Learning Methods (DLM) are currently considered State of the Art (SoA) [15] due to their impressive performance and robustness. A major limiting factor for DLM is the availability of large quality datasets - owing to their cost and complexity in developing. However the use of synthetic data solutions have worked to close this gap[15]. DLM can be classified into two categories-

- Regression Methods
- Perspective-n-Point (Indirect Deep Neural Network)

For the sake of brevity, this paper will only report on these two Deep Learning Methods, as they are considered more relevant and state of the art. However, traditional methods can be reviewed for a better understanding of pose estimation.

2.1 *Regression* Methods of Pose estimation

Regression methods train Neural Networks to directly predict the pose from the input RGB image. Regression based methods are simpler to implement, and generally outperform traditional methods [2].

- **PoseCNN** [19] is an important regression model for 6D pose estimation. PoseCNN introduced the 'patched' method- it attaches an object detection model before using a rotational regression network [2]. The object detection model predicts a 2D bounding box around an object, which is used to crop and resize the region of interest before being fed into a rotation regression network. This method inspired many other works to follow a similar method- as it allows for multi-pose estimation with only a simple single-pose estimation model [2].
- **EfficientPose** [1] is built on EfficientNet [16] and makes use of a bi-directional feature pyramid network [2][1]. EfficentNET allows for efficient scaling of the depth/width/resolution of Convolution Neural Networks' (CNN) [16]. EfficientPose thus build multiple subnets on EfficientNET [2], allowing for 4 different tasks - classification, detection , rotation and translation regression- to be solved for in an single forward path [2].
- **6D-VNET** extends MASK R-CNN [4] by adding custom heads to predict a vehicle's class, rotation and translation [18]. MASK R-CNN is an object image segmentation built on FASTER R-CNN [18]. MASK R-CNN allows for parallel object detection via 2D bounding box and segmentation[4] This is a well-performing end-to-end model, and has placed

consistently high in the Peking University Kaggle Vehicle 6D pose estimation competition.

Regression methods offer lightweight frameworks for 6D pose estimation, but generally struggle with accuracy. A key issue is the discontinuity of rotations in representations such as Euler angles or quaternions. [2]. Neural networks may diverge when a prediction is close to a discontinuous region. [22][2]. Regression methods may also struggle with symmetrical objects in an image, this is an important consideration for ground vehicles- where the front and the back of a car may be symmetrical

2.2 PnP methods of pose estimation

PnP is a well-known problem in computer vision, and is used in estimating the position and orientation of an object in 3D space from its 2D projection. The n in 'PnP' refers to the number of 3D that are required to solve a problem. There are different methods to solving a PnP problem.

PnP methods are powerful keypoint based models. PnP models may use keypoint detectors to extract keypoints from an image and match it to predefined keypoints- thus generating a 2D-3D correspondence [2]. A PnP solver can then be used to solve for object pose. PnP models have dominated recent Pose estimation benchmarking competitions [15].

- **BB8** referring to the 8 points of a bounding box. A CNN is used to predict the 2D locations of a 3D bounding box around an object [14]. These 2D points are used to generate a transformation matrix which provides the 3D coordinates of the object, which can then be used to estimate for objects pose [14]. BB8 was an early implementation of the PnP model, and thus struggles with occlusion and complex geometries. [2]
- **YOLO-6D** is a single-shot method that extends the work done in the BB8 method[2] It uses CNN to regress over a bounding box, and uses a PnP solver to solve for pose. [2]

2.3 Current state of 6D Pose estimation

2.3.1 ***Two-staged vs end-to-end models.*** Early Deep Learning Pose estimation models relied on a two-stage process to determine pose. The first stage involved feature extraction and object detection using a Convolution Neural Network, and then passing the Proposed Regions or cropped images to a seperate Pose Estimation model that is built on PnP/RANSAC[13]. This process requires the resampling an image for each stage, and makes geometric projection error prone. [18]

Recent SoA models implement an end-to-end approach, where object detection and pose estimation occurs through a single pass-through a Neural Network. This results in resampling an image only once, which should provide an increase in performance. [13]

2.3.2 ***Current State of the Art (SoA).*** The BOP challenge is a yearly challenge that aims to evaluate 6D pose estimation models, and benchmark them against each other. The highest-performing RGB-only pose estimation models from 2022 were built on [15] GDR-NET[17]. GDR-NET [17] takes in a cropped RGB output and densely predicts 2D-3D correspondences, and builds a mask of the visible object part[15]. Pose is then estimated using a CNN instead of PnP [15] This model showed strong performance in Frames-per-second(fps), anti-occlusion and with symmetrical objects.[15]

Table 1: Datasets for 6D pose estimation of vehicles [8]

Data	Format	Features
KITTI[3]	RGB	100Gb+ of high resolution RGB images
ApolloCar3D (Kaggle)	RGB	5Gb annotated data for pose estimation
NYC3DCars [11]	RGB	2000+ Annotated photos
EPFL Cars [12]	RGB	2000 RGB images from a car show
ShapeNET[20]	3D Models	3D models of Vehicles

Table 2: Other Popular RGB Datasets for general 3D object Detection and Pose estimation[8]

Data	Format	Features
COCO [9]	RGB	2,500,000 labeled instances
BOP [15]	RGB	Collation of multiple datasets. PBR data available
LINEMOD [5]	RGB	7481 trained and 80256 labeled objects

2.4 Datasets

Deep learning Methods rely on large, quality datasets to produce accurate and robust models. Currently, there are limited options for datasets for training pose estimation models for vehicles. Obtaining real training datasets is expensive and time-consuming, more-so for vehicles, as the dataset has to be annotated and ground truth must be determined. A workaround is to utilise synthetic data developed using 3D renders. This offers larger, cheaper datasets- however models trained on these datasets are often inaccurate when used on real data. [7][2] This domain gap between real and synthetic data can be reduced by creating datasets that utilise a mixture of both- real and synthetic data. There is also work being done on implementing Physically Based Rendering (PBR) techniques for synthetic data. PBR utilises tools such as ray-tracing and rasterization, to create more realistic synthetic data. This approach is greatly reducing the gap between synthetic and real data. [15]

Freezing the top layers of a network is another method of maximizing dataset use. Initially proposed by DPOD [21][2], the first few layers of a neural network learn very low-level features of an image. Thus, they are prone to over-fitting when trained on perfect synthetic images. We overcome this by training the first 5 layers of a network using pre-trained weights. [2]

Table 1 provides an overview of current publically available RGB datasets available for 3D object detection and pose estimation specifically for vehicle object detection and pose estimation. Table 2 provides other popular datasets that are used for general-object and bin picking object detection and pose-estimation.

2.5 Evaluation metric

2.5.1 . **Mean Average Precision (mAP)** is a commmon metric used in Object detection models. mAP is built on the following sub-metrics:

- **Confusion Matrix** - predictions are assigned a label based on their correctness- True Positive, True Negative, False Positive, False Negative
- **Intersection Over Union (IoU)**- measures overlap between predicted bounding box and ground truth bounding box
- **Recall** - Ratio of True Positives out of all Ground Truths
- **Precision** - Ratio of True Positives out of Total Detections

The above sub-metrics help calculate the Average Precision (AP) for each class. The mAP can thus be found by calculating average AP over all classes.

Definition 2.1. Mean Average Precision (mAP) is the average of AP calculated across all detected classes in an image:

$$MeanAveragePrecision(mAP) = 1/N \sum_{i=1}^{N} AveragePrecision(AP)$$

mAP is a standard metric and is commonly used with a IoU of 0.5 in object detection metrics such as YOLO and imageNET.

2.5.2 . **ADD** [6] is a popular metric for evaluating pose estimation models.

ADD/S calculates the average point difference between the 3D model points transformed by the ground truth rotation R and translation T, and the estimated rotation and . [6][1]

ADD is used for assymetrical objects, ADD-S is used for symmetrical objects. [6]

$$ADD = 1/m \sum_{x \in M} ||(Rx + t) - (\hat{R}x + \hat{T})||_2. \quad (1)$$

[6]

For symmetrical objects,

$$ADD - S = 1/m \sum_{x_1 \in M} min_{x_2 \in M} ||(Rx + t) - (\hat{R}x + \hat{T})||_2. \quad (2)$$

A pose estimation is considered 'correct' if the average point difference is 10% of target object diameter. [1]

3 METHODOLOGY

The methodology for this project can be seperated into three distinct stages-

- Requirement Gathering with Stakeholders- Swordfish Computing
- Open-Source model discovery and evaluation
- Environment configuration to run inference and training of models

3.1 Requirement Gathering

Through interactions with Swordfish Computing, a rubric of requirements was developed to assess the suitability of open-source models. These requirements were consistently reviewed and updated through weekly meetings.

- **Speed** Speed refers to the time taken to perform inference on provided input. Speed can be measured in milli-seconds or in frames-per-second (FPS). Real-time inference speed is considered to be 24 FPS. The model must be able to infer at 24FPS or above to be considered real-time.
- **Accuracy** ADD is a key metric, as it measures the accuracy of the predicted output of the model. [1] states that a model is 'correct' if it's between 10% of object diameter. Hence an ADD value between 5-10% of object diameter is desired.
- **Occlusion** Occlusion is not a key focus in this project. Occlusion is when objects in an image are obstructed from view- either partially or completely. We aim that our model is able to be resistant to up to 20% occluded images.
- **Model-architecture** An end-to-end model is an important requirement given its lightweight nature, and end-to-end trainable. This was decided according to the stakeholders needs in regards to future works with the model.
- **Symmetric Objects** The chosen model must show strong performance on symmetric objects, as a car is considered to be a symmetric object (along the y-axis).
- **Custom Data** The model must be customizable to be able to train on custom data and annotation passed in.

Features	Unsatisfactory	Satisfactory	Nice-to-Have
Speed	<24FPS	>=24FPS	>24FPS
Accuracy	>10% ADD	ADD <10% ADD	<5% ADD
Occlusion	no-resistance	mild-resistance	Fully-resistant
Model Architecture	Two-staged	End-to-end	-
Symmetric Objects	Does not perform	Performs	Performs well
Custom-data	Non-trainable	Trainable	Customisable

Table 3: Model-selection requirement matrix

3.2 Model Discovery and Evaluation

The discovery process involved scoping through various survey papers and conference proceedings papers, which helped highlight the current state of pose estimation, future trends, and recent work.

Using the developed requirement matrix, and findings from the literature review process, the following models were chosen for consideration

- 6DVnet
- YOLO3D
- YOLOv8 Pose Estimation
- PoET Pose Estimation
- EfficientPose-Main

Efficient-Pose was ultimately chosen given it's strong performance and speed, and it's continued support through updates. The use of efficient-net, and it's end-to-end architecture made it a promising choice.

3.3 Environment configuration

Setting up a working environment to install and run pose estimation libraries proved to be a time-consuming process. Incompatibilities with operating systems and CUDA versions, meant that virtual environments such as Anaconda and Docker were used.

The following tools were used to set up a working environment

-

- **Nvidia CUDA** allows for GPU pass-through to an application for training and running Machine Learning models.

EfficientPose requires CUDA version 10.0 to run GPU training and and inference.
- **Anaconda** Anaconda was used to build a conda virtual environment with the required python version (3.7) to run EfficientPose.
- **Tensorflow** was installed and run within the created Anaconda Virtual environment.
- **EfficientPose** Download and install EfficientPose using the provided installation instructions. A few changes were made to the 'requirements.txt' file to resolve conflicts with using a pip installer within a conda environment.

Other solutions that were explored but not used included running EfficientPose within a docker container-

- **Docker** A docker file was built to configure a docker image with the specific version requirements. A major advantage of using docker is the ability to use different CUDA versions in a Docker image, without having to redownload different Nvidia driver and CUDA versions on the host machine - which could lead to OS runtime errors.

4 EXPERIMENTAL SETUP

4.1 Setting-up Custom Dataset

Working with our project stakeholders- Swordfish Computing, a dataset consisting of synthetic images and annotations was obtained. The main advantage of a synthetic dataset is-

- Full control of lighting conditions and object pose
- Easy annotation of ground truth

The synthetic dataset consists of 2,500 images of a 3D model of a vehicle rendered in various real-life backgrounds. The object is transposed and rotated and placed in a camera frame, with a real-life 2D image rendered as the background- to improve the performance of the model in real-life conditions. The scene is created and rendered using Blender.

The dataset was in the COCO dataset format, with the images split into a 'Train' and a 'Test' split, with all ground truth annotations in one JSON file. EfficientPose requires the dataset used, to be in the BOP specified Linemod format [15]. Hence, converting our custom dataset to suit the required format involved the following steps- - which is outlined as below-

4.1.1 Re-configuring file-structure. A series of python scripts were written to help in automating the conversion from COCO format [**?**] to linemod. The BOP dataset toolkit library [15] was a valuable resource for scripts in converting datasets into the LineMod format.

4.1.2 Configuring Ground Truth annotations to match required format. The ground-truth annotations provided in the custom-dataset were

- Camera and Vehicle Location in blender world coordinates (x,y,z)
- Camera and Vehicle rotation represented using Euler angle rotations.
- 2D bounding-box values of the vehicle in (x,y,width,height) format.

The Linemod Datset expects the following ground-truth values:

- *cam_t_m2c* 3x1 translation matrix of the object in the camera's coordinate system. (in millimetres)
- *cam_r_m2c* 3x3 rotation matrix of the object in the camera's coordinate system.
- *bbox* 2D bbox vertex values in the format (x1,y1,x2,y2).

The script developed to generate and convert the annotations can be found in *prepare_dataset.py*

We first convert the camera object rotation, stored in the custom dataset as a euler rotation vector, to a 3x3 rotation matrix.

$$rot\_mat\_bl = rot\_vec\_to\_mat(eul\_rot\_bl) \tag{3}$$

To find our *cam_t_m2c* values, we use the following equation.

$$cam\_t\_m2c = rot\_mat\_bl \times world\_cords\_bl \tag{4}$$

where rot_mat_bl is our camera object rotation in the blender co-ordinate system, represented as a 3x3 rotation matrix, and $world\_cords\_bl$ is our camera object world-coordinates in the blender coordinate system, represented as a 3x1 matrix. This equation converts our location matrix from the Blender world coordinate system to the camera coordinate system.

We calculate *cam_r_m2c* by converting our rotation_matrix from it's rotation matrix representation to it's axis angle representation, since that is the only accepted rotation representation for EfficientPose. Axis-angle representations, also prevents the issue of Gimbal lock that occurs with Euler rotations.

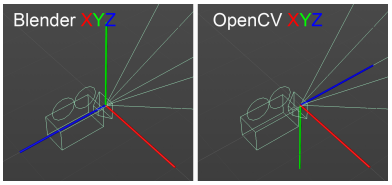
$$axis\_angle\_bl = rot\_mat\_to\_axis\_angle(rot\_mat\_bl) \tag{5}$$

where $axis\_angle\_bl$ is our camera object rotation matrix converted to an axis-angle rotation representation.

We then multiply Y and Z values of $axis\_angle\_bl$ by -1, to align the Y and Z axis with the OpenCV coordinate system.

$$cam\_r\_m2c = \begin{bmatrix} X & -Y & -Z \end{bmatrix}_{axis\_angle\_bl} \tag{6}$$

An important note to consider, is the different coordinate-systems used in Blender and OpenCV. The Blender co-ordinate system, which denotes X,Y,Z as follows-



[**?**]

Hence to go from blender coordinate system to OpenCV coordinate system, we multiply the Y and Z-axis -1 to denote positive Y-values below the origin/camera, and positive Z-values away from the origin/camera.

4.1.3 Generating a 3D mesh of vehicle object. A 3D mesh of the vehicle object is required to be passed in as a .ply file. A .ply file stores the location of the vertices making up a 3D mesh object. These vertices are used in calculating the Average Point Distance values to measure the accuracy and loss of the regression model. The vehicle 3D object was exported from blender to Meshlabs, with the origin of the 3D object, at the origin of the world. The object was left at it's default scale. The model was then exported from Meshlabs as a ṗly file.

generate_model_info.py script was then used to read in the 3D ply file, and return the following values to create *model_info.py*

- *obj_diameter* is the largest distance between any two vertices in the 3D mesh.
- *min_x* is the minimum x-value of the 8 vertices of the object's 3D bounding box.
- *min_y* is the minimum y-value of the 8 vertices of the object's 3D bounding box.
- *min_z* is the minimum z-value of the 8 vertices of the object's 3D bounding box.
- *size_x* is the distance between max_x and min_x 3D bounding box values.
- *size_y* is the distance between max_y and min_y 3D bounding box values.
- *size_z* is the distance between max_z and min_z 3D bounding box values.

A key challenge that was faced while preparing the 3D object, was ensuring the scale of the ground truth annotations, the model dimensions and EfficientPose's expected dimensions aligned.

The model was left in it's default scale, with the 3D object model values in *model_info.py* values multiplied by 1000 to convert from metre (blender's standard unit of measurement) to millimetres (expected unit by EfficientPose). The world coordinates of the camera object $world\_cords\_bl$ was also multiplied by 1000 to convert to millimetres.

4.1.4 ***Validating ground truth values with debug.py.*** Before beginning training, it is important to use the provided *debug.py* script to annotate the given ground truth values on training images, to see if our model's ground truth and 3D object bounding box values are in the right coordinate system and scale.

A key challenge we faced, was that our vehicle 3D object and our blender world were in two different scales- this created issues with drawing the 3D bounding box at the right scale, as the OpenCV function responsible for projecting the 3D bbox vertices to 2D and drawing them on the image- *cv2.projectPoints()*, expects the translation_vector and the 3Dbbox points to be in the same scale. To get over this, we had to manually fine-tune our 3D bounding box values using trial-and-error to fit the bounding box around the object with the right scale-

- Reduce *min_y* value by 1000 and reduce *size_y* value by 2000.
- Reduce *size_z* value by 600.

4.2 Training

Training was performed using the following command-

```
python train.py --phi 0 --weights
./weights/Occlusion/phi_0_occlusion_best_ADD(-S).h5
-no-6dof-augmentation --epochs 100 --compute-val-loss
 --validation-image-save-path
./Validation/ linemod ./custom_dataset/ --object-id 1
```

The inputted arguments are described below.

- *–phi 0* 'phi' is a value used to uniformly scale the width, depth and scale of the EfficientNet backbone [16] used. A value of phi-0 was chosen as it was shown to have the best tradeoff between accuracy and speed [1].
- *–weights* Training EfficientPose on a new dataset from scratch would be a time-consuming and resource expensive process- as we have to train the classifier and object detector as well. Given that our dataset is relatively small (2500 images) we chose to employ transfer learning by using weights of the model trained on the LineMod Occlusion dataset- which can be found on the official EfficientPose repo.There are other pre-trained weights including COCO pretrained which may be of interest for future work.
- *–no6dof-augmentation* EfficientPose has a novel 6DoF augmentation process [1], which augments the image and its respective ground truth value by a certain degree. We chose to forego this feature, due to the issues we were facing with model and ground-truth scale (see section 4.14).
- *–epochs 100* This argument controls the number of epochs to train the model on. We chose a value of 100, as any value higher may lead to overfitting.
- *validation arguments* Arguments to store validation loss data, and to store validation images.

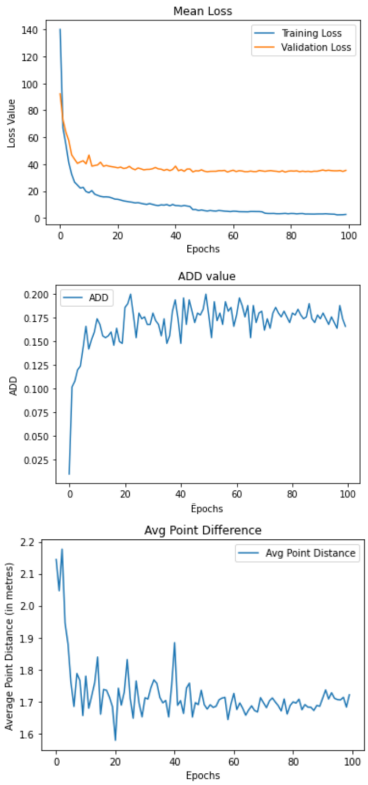
The model was trained for 100 epochs at the default learning rate of 0.0001, with the default batch size of 1, using the ADAM optimiser.

4.3 Results

4.4 Model Loss values

We extract the loss values using tensorboard, and plot the values using matplotlib.

Looking at our ADD and APD curves, we notice that the model becomes increasingly accurate through more epochs- this was verified manually by inspecting the validation images during training, and seeing the model was able to infer pose fairly accurately. However, we notice that the APD- or average point Difference at it's lowest value is around 1.6, which is still above 10% of the diameter of the 3D object (0.528m). This could be attributed to an error with the scale of the 3D object. Thus, by using our metric of Avg Point Difference being between 10% of object diameter [1], our model fails to accurately predict pose to a acceptable range of error.



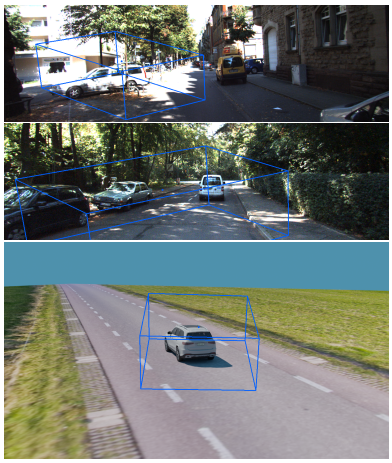
A key takeaway, is that the training and Validation Loss curves diverge after only 10 epochs, and the curves continue to diverge. This indicates that the model has overfit for the inputted data. Overfitting was an expected concern for this project- given the small size of the dataset.

4.5 Inference

We ran inference using our trained model, and by editing our *inference.py* model to fit our custom data. When testing inference, we chose to use validation images the model had never seen before, including images from real-life scenes from KITTI.

Below are some validation images generated after inference-

Below is the inference performed on validation images the model has never seen before-



As seen, the model performs well on inferencing pose on validation images the model has seen before, while it fails to predict pose reliably on new images. This indicates that our model has overfit to our dataset.

4.6 Discussion

As seen in our Loss Curves and in our inference section, our model has overfit for the data- thus producing good performance for the given dataset, but it is unable to make meaningful predictions for new data. Possible ways to combat this include-

- Reduce the learning rate, to allow for the model to move to our optimal weights at a slower rate.
- Split training into multiple short epoch counts, and evaluate loss curves in between. This allows us to stop training as soon as it reaches convergence.

- Make use of the 6DoF augmentation tool provided in EfficientPose [1]. Augmentation is a powerful tool in resisting overfitting.However, care must be taken that the 6DoF augmentation tool augments the ground truth by the correct value.
- Increase the phi value to 3 or beyond. EfficientNet[16] scales the parameters of the network by a uniform scale. Phi 0 was used in this experiment for it's performance as cited in [1], however since our dataset image size is higher (1920x1080) as opposed to the (640 x 480) images used in the LineMod dataset. The higher input image size might require more network layers, to allow for the model to capture more 'fine-grained' patterns on the bigger image [16]
- Increase the size of the custom dataset. This can be achieved by introducing real-life images of vehicles from KITTI [**?**], or by improving the size of the synthetic dataset used.

5 CODE

All code for this paper can be found Here* as 6DoF Pose Estimation Model.

Follow the instructions on the README.txt to install a customised version of EfficientPose.

5.1 Conclusion

This paper explored the pose estimation of ground vehicles using a single monocular RGB image. The current state of Pose Estimation is increasingly moving towards end-to-end Deep Learning Methods, although a performance deficit may still exist between traditional PnP solver methods. EfficientPose [1] was chosen to be used to try and perform pose estimation of vehicles using a custom dataset generated by Swordfish Computing. The results showed the EfficientPose model, built on EfficientNet, learns well, but overfit to our dataset given the listed parameters. However, the model still showed significant promise. Proposed further work includes- fine-tuning the model and improving the size of the custom synthetic dataset.

REFERENCES

[1] Yannick Bukschat and Marcus Vetter. 2020. EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach. (2020). arXiv:cs.CV/2011.04307

[2] Jiale Chen. 2022-11. *Texture Optimization for 6 DoF Pose Estimation.* Master Thesis. ETH Zurich, Zurich. https://doi.org/10.3929/ethz-b-000583168

[3] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

[4] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2018. Mask R-CNN. (2018). arXiv:cs.CV/1703.06870

[5] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. 2011. Multimodal Templates for Real-Time Detection of Texture-less Objects in Heavily Cluttered Scenes. *Proceedings of the IEEE International Conference on Computer Vision*, 858–865. https://doi.org/10.1109/ICCV.2011.6126326

[6] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. 2012. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. *Proc. Asian Conf. Computer Vision* 7724. https://doi.org/10.1007/978-3-642-33885-4_60

[7] Tomas Hodan, Martin Sundermeyer, Bertram Drost, Yann Labbe, Eric Brachmann, Frank Michel, Carsten Rother, and Jiri Matas. 2020. BOP Challenge 2020 on 6D Object Localization. (2020). arXiv:cs.CV/2009.07378

[8] Sabera Hoque, Md. Yasir Arafat, Shuxiang Xu, Ananda Maiti, and Yuchen Wei. 2021. A Comprehensive Review on 3D Object Detection and 6D Pose Estimation With Deep Learning. *IEEE Access* 9 (2021), 143746–143770. https://doi.org/10.1109/ACCESS.2021.3114399

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2015. Microsoft COCO: Common Objects in Context. (2015). arXiv:cs.CV/1405.0312

[10] Yangxintong Lyu, Remco Royen, and Adrian Munteanu. 2022. MONO6D: Monocular Vehicle 6D Pose Estimation with 3D Priors. In *2022 IEEE International Conference on Image Processing (ICIP).* 2187–2191. https://doi.org/10.1109/ICIP46576.2022.9897311

[11] Kevin Matzen and Noah Snavely. 2013. NYC3DCars: A Dataset of 3D Vehicles in Geographic Context. In *Proc. Int. Conf. on Computer Vision.*

[12] Mustafa Ozuysal, Vincent Lepetit, and Pascal Fua. 2009. Pose estimation for category specific multiview object localization. In *2009 IEEE Conference on Computer Vision and Pattern Recognition.* 778–785. https://doi.org/10.1109/CVPR.2009.5206633

[13] Patrick Poirson, Phil Ammirato, Cheng-Yang Fu, Wei Liu, Jana Košecká, and Alexander Berg. 2016. Fast Single Shot Detection and Pose Estimation. (09 2016).

[14] Mahdi Rad and Vincent Lepetit. 2018. BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. (2018). arXiv:cs.CV/1703.10896

[15] Martin Sundermeyer, Tomas Hodan, Yann Labbe, Gu Wang, Eric Brachmann, Bertram Drost, Carsten Rother, and Jiri Matas. 2023. BOP Challenge 2022 on Detection, Segmentation and Pose Estimation of Specific Rigid Objects. (2023). arXiv:cs.CV/2302.13075

[16] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 6105–6114. https://proceedings.mlr.press/v97/tan19a.html

[17] Gu Wang, Fabian Manhardt, Federico Tombari, and Xiangyang Ji. 2021. GDR-Net: Geometry-Guided Direct Regression Network for Monocular 6D Object Pose Estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 16611–16621.

[18] Di Wu, Zhaoyong Zhuang, Canqun Xiang, Wenbin Zou, and Xia Li. 2019. 6D-VNet: End-To-End 6DoF Vehicle Pose Estimation From Monocular RGB Images. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).* 1238–1247. https://doi.org/10.1109/CVPRW.2019.00163

[19] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. 2018. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. (2018). arXiv:cs.CV/1711.00199

[20] Li Yi, Vladimir G Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, ARCewu Lu, Qixing Huang, Alla Sheffer, Leonidas Guibas, et al. 2016. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 210.

[21] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. 2019. DPOD: 6D Pose Object Detector and Refiner. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV).* 1941–1950. https://doi.org/10.1109/ICCV.2019.00203

[22] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. 2019. On the Continuity of Rotation Representations in Neural Networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 5738–5746. https://doi.org/10.1109/CVPR.2019.00589

[23] Yingzhao Zhu, Man Li, Wensheng Yao, and Chunhua Chen. 2022. A Review of 6D Object Pose Estimation. In *2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, Vol. 10. 1647–1655. https://doi.org/10.1109/ITAIC54216.2022.9836663