Posenet++: A CNN Framework for Online Pose Regression and Robot Re-Localization

Ray Zhang, Zongtai Luo, Sahib Dhanjal, Christopher Schmotzer, Snigdhaa Hasija
University of Michigan, Ann Arbor, MI
https://posenet-mobile-robot.github.io/

***Abstract*—In this project, we develop a novel re-localization algorithm which addresses the global localization problem. We modify PoseNet, a robust and real-time monocular six degree of freedom re-localization system, to solve the purpose of smoothing and mapping in conjunction with GTSAM. Our system trains a convolutional neural network to regress the 6-DOF camera pose from a single RGB image in an end-to-end manner without the requirement of additional feature detection. The algorithm operates in real time, taking approximately 45ms per frame to compute. We use VGG-16 network to achieve solutions to complicated out of image plane regression problems. We leveraged transfer learning from large scale classification data. We also demonstrate the viability of the algorithm to localize in situations with difficult lighting, motion blur and different camera intrinsic where point based SIFT registration fails.**

***Index Terms*—PoseNet, GTSAM, CNN, Pose Regression, re-localization**

I. Introduction

In the scenario of an arbitrary change of location, commonly referred to as the **kidnapped robot problem**, a robust and real-time re-localization is necessary in robot perception. To approach this problem, we want to explore the advances in Deep Learning, specifically a Convolutional Neural Network for pose regression. This project outlines our implementation of a full robot re-localization pipeline using PoseNet [4] as the sensor model, the odometry of the robot as the action model, and Georgia Tech Smoothing and Mapping (GTSAM)[18] as our backend for factor graph optimization.

The motive behind this project is to demonstrate a robust method for localization of a robot in places where SIFT based feature registration fails: difficult lighting, different environmental scenes and motion blur to name a few. We've demonstrated our pipeline capable of re-localizing a robot to within 2m and 3° of its true position in mid-scale environments, and that convolutional neural networks, used on mono camera systems, can be a viable solution to the kidnapped robot problem.

Convnets help build an end-to-end model to regress camera's orientation and position but they are unable to be generalized for different data distributions. Training convolutional networks is usually dependent on very large labeled image datasets, which are costly to assemble. Examples include the ImageNet and Places datasets, with 14 million and 7 million hand-labeled images, respectively. Therefore, we use transfer learning which trains a pose regressor, pre-trained as a classifier, on immense image recognition datasets. This converges to a lower error in less time, even with a very sparse training set, as compared to training from scratch. We have pre-trained our model on the Places dataset [19].

II. Related Work

The localization problem is solved through two approaches: Metric-based and Appearance-based. Metric SLAM localizes a mobile robot by focusing on creating a sparse or dense map of the environment. M. Cummins and P. Newman[14] propose scalable appearance-based localizers which use SIFT features in a bag of words approach to probabilistically recognize previously viewed scenery. Convolutional neural networks have also been used to classify a scene into one of several location labels [15].

A method of Scene Coordinate Regression Forest for re-localization is proposed by D.P. Kingma and Jimmy Ba [10], where they use depth images to create scene coordinate labels which map each pixel from camera coordinates to global scene coordinates. This mapping was then used to train a regression forest to regress these labels and localize the camera. PoseNet is closely related to this algorithm.

A robust and real-time monocular six degree of freedom visual re-localization system is also provided by Alex Kendall and Roberto Cipolla[11]. They use a Bayesian convolutional neural network to regress the 6-DOF camera pose from a single RGB image. Their algorithm can operate indoors as well as outdoors in real time, taking under 6ms to compute.

The *"learning to see by moving"* approach learns to do visual odometry with a deep CNN given a couple of nearby frames [12]. Although it doesn't work very well yet, but one can imagine hooking this into something like the LSD-SLAM pipeline to obtain much more robust associations between neighboring frames or a frame with its nearest keyframe, than edges alone can provide.

CNN-SLAM presents a CNN-based depth prediction for monocular SLAM and semantic mapping [13]. Although convolutional neural networks classify spatio-temporal data really well, they are only just beginning to be used for regression. They have advanced the state of the art in object detection [16] and human pose regression [17]. However their regression targets are limited to lie in the 2-D image plane. PoseNet regresses the full 6-DOF camera pose transform including depth and out-of-plane rotation. Furthermore, it's able to learn regression as opposed to being a very fine resolution classifier. It has been shown that convolutional network representations

trained on classification problems generalize well to other tasks. These representations of classification can be applied to 6-DOF regression problems. Using these pre-learned representations allows convolutional networks to be used on smaller datasets without over-fitting.

III. Methodology

The methodology employed in the project deploys the pipeline shown in *Figure 1*.

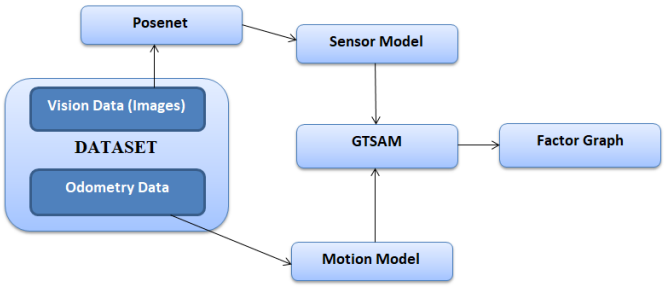

Fig. 1: Project Pipeline

The individual components of the above figure are described in detail below:

A. PoseNet

In our project, we provide PoseNet with vision data *(images)* as input and get the camera pose in a 6 DOF frame of reference as it's output. PoseNet [4] is a robust and real-time monocular six degree of freedom re-localization system which deploys a convolutional neural network (convnet) trained end-to-end to regress the cameras orientation and position. It leverages transfer learning from very large scale classification datasets which helps it converge to a lower error in substantially lesser time, even with a very sparse training set, as compared to training the whole network from scratch.

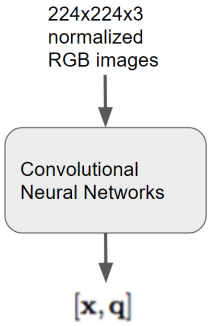

Fig. 2: PoseNet Workflow

Appearance-based relocalization has had success [1, 2] in coarsely locating the camera among a limited, discretized set of place labels, leaving the pose estimation to a separate system. PoseNet presents a means of computing continuous pose directly from appearance. The scene may include multiple objects and need not be viewed under consistent conditions. It estimates the camera pose directly from a single monocular RGB image. The output of the network is a pose p given by a 3D camera position **x** and orientation represented by a quaternion **q**:

$$p = [x, q] \tag{1}$$

Pose **p** is defined relative to an arbitrary global reference frame which is defined by the training set used. Quaternions are the chosen form of orientation representation mainly because arbitrary 4-D values are easily mapped to legitimate rotations by normalizing them to unit length. To regress this pose, the convolutional neural network is trained on Euclidean loss using stochastic gradient descent with the following objective loss function:

$$loss(I) = ||\hat{x} - x||_2^2 + \beta||\hat{q} - \frac{q}{||q||}||_2^2 \tag{2}$$

The parameter β is a scale factor chosen to keep the expected value of position and orientation errors to be approximately equal. In our loss function, a balance(β) must be struck between the orientation and translation penalties as they are highly coupled. This is mainly due to the fact that they are regressed from the same model weights. The optimal β is given by the ratio between expected error of position and orientation at the end of training, not the beginning. Based on thorough experimentation, we found that β is greater for outdoor scenes *(when compared to indoor scenes)*, as positional errors tend to be larger *(0.5m indoors versus 2m outdoors on average)*.

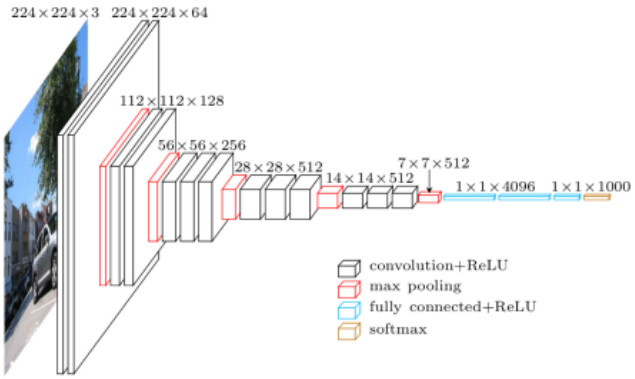

Fig. 3: VGG-16 Architecture

The original implementation of PoseNet uses the GoogLeNet for classification. We, however, used another CNN architecture for classification: VGG-16 [6]. It serves as a basis for developing our pose regression network. The architecture is shown in *Figure 3*. It is characterized by its simplicity, using only 3x3 convolutional layers stacked on top of each other in increasing depth. With a given receptive field(the effective area size of input image on which output depends), multiple stacked smaller size kernels are better than the ones with a larger size kernel because multiple non-linear

layers increases the depth of the network which enables it to learn more complex features, and that too at a lower cost.

The VGG convolutional layers are followed by 3 fully connected layers. The width of the network starts at a small value of 64 and increases by a factor of 2 after every sub-sampling/pooling layer. It achieves the top-5 accuracy of 92.3 percent on ImageNet. Volume size reduction is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier.

B. Georgia Tech Smoothing and Mapping

Georgia Tech Smoothing and Mapping *(GTSAM)* [8] is a library of C++ classes that implements smoothing and mapping (SAM) in robotics and vision, using factor graphs and Bayes networks as the underlying computing paradigm rather than sparse matrices.

Factor graphs are graphical models [3] that are well suited to modeling complex estimation problems, such as Simultaneous Localization and Mapping (SLAM) or Structure from Motion (SfM). They are bipartite graphs consisting of factors connected to variables. The variables represent the unknown random variables in the estimation problem, whereas the factors represent probabilistic information on those variables, derived from measurements or prior knowledge. Bayes networks are another type of graphical models which are directed acyclic graphs.

GTSAM provides state of the art solutions to the SLAM and SfM problems, but can also be used to model and solve both simpler and more complex estimation problems. It exploits sparsity to be computationally efficient. Typically, measurements only provide information on the relationship between a handful of variables, and hence the resulting factor graph will be sparsely connected. This is exploited by the algorithms implemented in GTSAM to reduce computational complexity. Even when graphs are too dense to be handled efficiently by direct methods, GTSAM provides iterative methods that are quite efficient regardless.

In our project we use incremental smoothing and mapping using the Bayes tree *(iSAM2)* [7] as our back-end optimization tool. Based on Bayes Tree structure, the first order least square matrix can be updated efficiently according to the factor dependencies.

C. Sensor Fusion

In our project, we build the factor graph based on robot states (*2D pose* (x, y, θ)) and fuse the PoseNet prediction by adding it's factor to the current vertices. Referring to *Figure 4*, we define X_t as the estimation of the current robot 2D pose. Each robot pose is constrained by the odometry and its covariance, which is shown as the **black dot** factor in the figure. The odometry can be directly calculated based on the measurement from wheel encoders and inertial measurement unit (IMU). The covariance of the odometry is based on the sensor specifications and varies from robot to robot.

We assume each odometry factor to be a Gaussian random variable, and independent of each other. The odometry measurement is generally higher in frequency, hence we do not add the odometry factor immediately. Instead, we accumulate the odometry's mean and covariance, according to the the sum of Gaussian independent random variables, till we get the ground-truth pose generated by PoseNet. Using this approach synchronizes the frequency of PoseNet measurement and odometry measurement efficiently without building a large factor graph, while maintaining good accuracy.

To fuse the PoseNet measurement/prediction, an extra factor similar to the GPS measurement is connected to the current vertex of the robot. This can be seen in *Figure 4*, where we add the PoseNet factor in purple color by matching the time-stamp. This measurement is independent of the existing graph and added to the current vertex as a prior factor. The covariance of the PoseNet measurement is obtained from running the PoseNet on the test dataset. In cases where the test covariance cannot be obtained, an approximation is used, which is suggested by the training standard deviation. An assumption while computing this approximation is that the test and train images are obtained from a similar distribution.

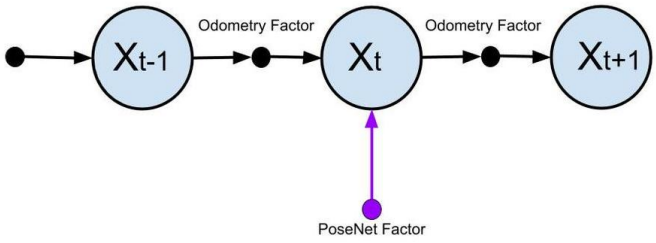

Fig. 4: Factor Graph

IV. Experiments

In this section, we evaluate the implemented methods, including PoseNet, sensor fusion, and localization with GTSAM. They are evaluated on 3 datasets encompassing both indoor and outdoor scenes. For all these datasets, we need the robot poses as ground truth to train PoseNet. If the odometry data is also available, we use GTSAM to fuse them with PoseNet's outputs and provide a final prediction of the robot pose.

A. Datasets

Our experiments were conducted using the following datasets as input to compare how well our pipeline performed in different environments.

1) Outdoor Dataset: King's College: The King's College dataset is a subset of the Cambridge Landmarks Dataset and is also used in the original PoseNet paper [4]. The rationale behind using this dataset is to benchmark our implementation with respect to the original implementation.

2) Outdoor Dataset: Shop Facade: This dataset is similar to the King's College Dataset as it originates from the Cambridge Landmarks Dataset as well. The reason behind using this is to test our implementation on a very sparse dataset and compare it to the performance of the original PoseNet paper [4].

Fig. 5: Example Images: The Cambridge Landmarks Dataset

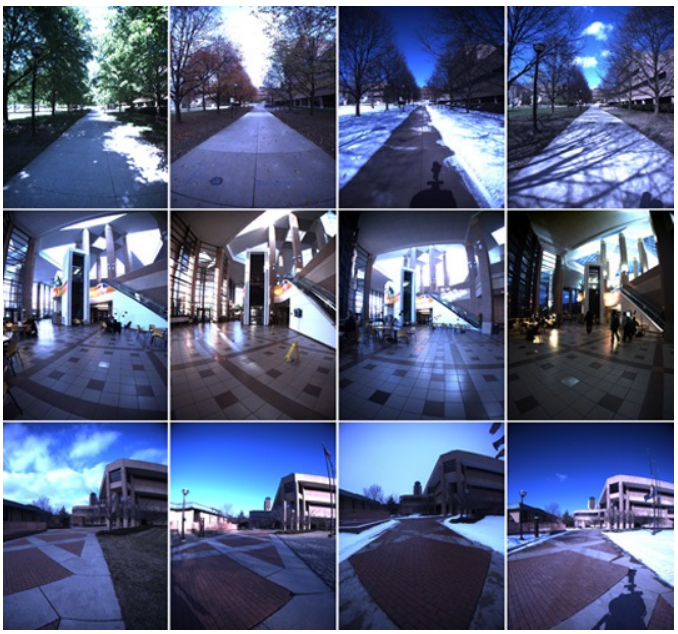

Fig. 6: Example Images: NCLT Dataset

3) Outdoor Dataset: North Campus Long Term: The North Campus Long-Term Vision and LIDAR Dataset, or NCLT Dataset, is an outdoor dataset that contains the data gathered spanning multiple sessions of a Segway robot exploring the University of Michigan's North Campus. Approximately 5.5km of the campus was covered during each of the 15 sessions.

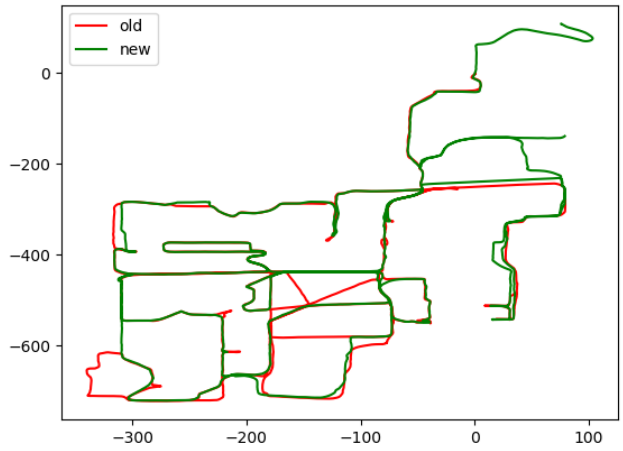

Fig. 7: The Path encompassed in the sessions used: (i) **old** (dataset from **2012-01-08**), (ii) **new** (dataset from **2012-03-17**)

B. Training

We implemented VGG-16 in Tensorflow that was pre-trained on the *Places2 dataset* [9]. We slightly modified the architecture by replacing the last softmax classification layer with one fully connected layer with 2048 neurons. On top of this layer, we add an output layer with either 6 or 7 output regression units. The number depends on the representation of the training data. 6 neurons are used when the orientation is represented as Euler angles, whereas 7 are used when it's represented using quaternions. On the Shop Facade, King's College and Fetch datasets, the ground truth for rotations are represented as quaternions, so the pose vector returned by PoseNet is of 7 dimensions. The NCLT dataset, however, uses Euler angles, so the pose vector output is of 6 dimensions instead.

We use the ADAM optimizer [10] that uses the first and second order moments to control the step size of the gradients updates. The learning rate is set to $1e^{-5}$ and $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e^{-8}$. For the loss function, the $\frac{q}{||q||}$ is modified to q, suggested by author's implementation.

Each of the training images are pre-processed as follows before being fed to the network:

1) Scale down the original image till the smallest dimension is of size 256 pixels. In our case, the input images are of various sizes, which we then scale down to a size of 455 x 256 px, and rotate them accordingly
2) Generate 128 random crops of size 224 x 224 from the scaled down image, along with a center-cropped region.
3) Normalize each of the generated samples by subtracting the mean and dividing it by the standard deviation.
4) Associate the generated samples to their corresponding labels and send them to the network for training.

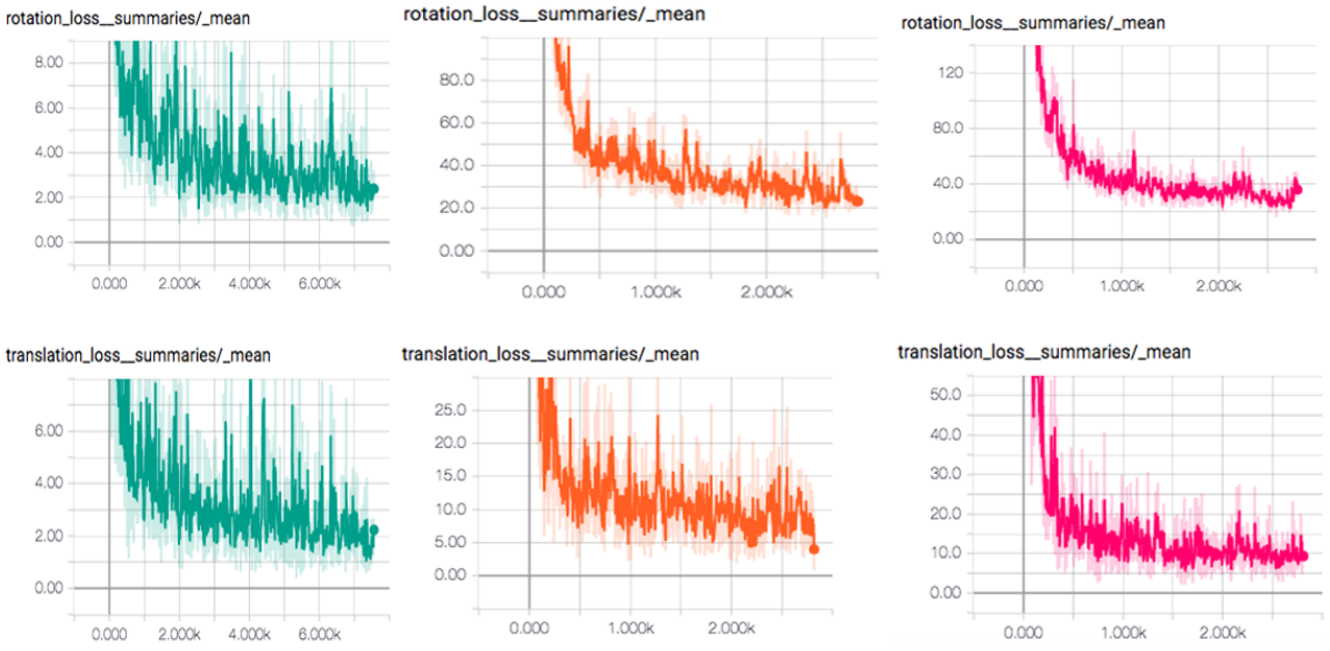

Fig. 8: Training loss on the following datasets: (i) ShopFacade, (ii) King's College, and (iii) NCLT

To validate our algorithm we use four sequences of NCLT dataset to train our PoseNet neural network and then use it on different datasets to test our results. Since the size of each sequence is very large and the trajectory is not identical, we picked the overlapping trajectories that appeared in the four sequences that we chose. For the training sequence, they are: 2012-01-08, 2012-03-17, 2012-10-28 and 2012-11-04 and for testing, we use 2012-03-31. We sampled around 650 images from each of the training sequences and 605 images from the testing sequence in the overlapped area. From inspection, this area contains buildings but the features are not rich enough, since all the buildings look similar.

We found out the following weighting factor,s β in the loss function to work out best for the respective datasets:

1) **Shop Facade**: 100
2) **NCLT**: 100
3) **King's College**: 500

The training of ShopFacade and King's College converges in 2 hours on average using two $GTX1070$ GPUs, while NCLT takes 4-5 hours. The translation and rotational loss obtained for each dataset is shown in *Figure 8*.

C. Testing Results

1) PoseNet Regression Results: We test PoseNet predictions on the Shop Facade and King's College dataset and plot the ground-truth versus predictions (*refer Figure 9*). Note that to speed up the testing process, we only center crop a 224×224 region and feed it into the network. We just take into consideration the (x, y, θ) locations from the predicted pose vector instead of the 6-DOF pose. The mean error and standard deviation obtained from regression can be seen in Table I. The distance is in meters, and θ is in rad. These errors are close to those in the original PoseNet paper.

On large scale outdoor scenes such as NCLT, the testing error is around 12 meters, much larger than the errors of King's College. We have singled out some potential reasons for the same:

1) The use of a super wide angle lens in the NCLT Dataset
2) PoseNet works by mapping features in an image to regress the camera pose. The images from the NCLT Dataset however are taken from far out locations, laying very less stress on the features (such as buildings, etc)
3) To localize, PoseNet generally requires images from different viewing angles, rather than a continuous stream of images, as was the case with the NCLT dataset.

To arrive at the above conclusions, we tested PoseNet on (i) a small area of the map in Figure 7, and (ii) the whole map of the session.

2) PoseNet fused with Odometry: With the optimization from GTSAM as back-end and PoseNet measurement as a front-end sensor solution, the localization result can be found in *Figure 11*. The measurement mean error has been greatly reduced and the trajectory has been smoothed with the information of the odometry data. The error statistics can be found in [citation to nclt table]. The PoseNet prediction and iSAM optimization frequency is 22 Hz.

V. Discussion

A. PoseNet Pose Regression

We noticed that PoseNet performs well when the input images contain unique features, such as the geometry of

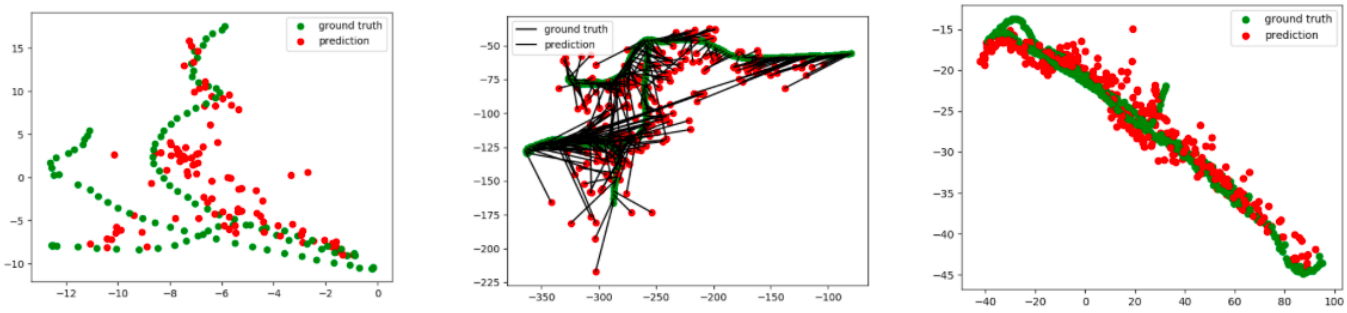

Fig. 9: Test Result of PoseNet predictions on (i) Shop Facade, (ii) NCLT and (iii) King's College

DataSet	Train Frames	Test Frames	Spatial Extent	PoseNet Mean(VGG-16)	PoseNet Std Dev(VGG-16)	PoseNet Mean (GoogLeNet)
NCLT (parital)	2622	605	300m x 150m	-12.370m, 1.146°	37.121m, 20.340°	NA
King's College	1220	343	140m x 40m	1.587m, 2.38°	6.02m, 3.684°	1.92m, 2.70°
Shop Facade	231	103	35m x 25m	-1.108m, 5.615°	3.00m, 15.069°	2.1m, 5.20°

TABLE I: Dataset details and Test Results, Compared with the PoseNet paper's result

the buildings or windows. However, it doesn't perform well when the images of different locations share similar features. For instance, in *Figure 10a* and *Figure 10b*, whose majority regions are grass/pathway/sky. Test results on these images yield large errors.

(a) Error: 54.439m, $-35.795°$ (b) Error: 8.127m, $-14.438°$

Fig. 10: Images with unique features like buildings will have better predictions than those with common features such as pathways/trees

Moreover, the test result indicates that when the spatial extent of the dataset grows, the errors of the predictions increase. Especially when the movement of robot includes more rotations, the rotation errors will increase. NCLT dataset has larger scale with many turn arounds, and accordingly it has large prediction error. Our intuition is that in a large scene, the network is closer to its capacity.

B. PoseNet fused with GTSAM

The trajectory optimized from the GTSAM-iSAM2 matches closely with the ground truth which is a result of a SLAM sensor fused with camera at 22 Hz frequency. The mean error and standard deviation are reasonable, since the dataset is an outdoor dataset and the scale is large. As for the running speed, we write our codes in python, but the running speed is still very high, which is about 0.045 frames per second. The performance can be further improved by C++ implementation. These results show the potential of real-time performance on a robot that only requires an inexpensive mono camera. Meanwhile, it needs a powerful GPU for the neural network prediction.

C. Global Localization

In this project, we initialized our robot with an arbitrary starting pose and tried to localize the robot with a bad prior. From the test results, if the scene has been seen by PoseNet, the robot can be localized globally quickly, even if it is kept at an arbitrary location abruptly. However, if the surroundings are new to the robot, the robot will fail to localize. This is due to the fact that the PoseNet regressor is a re-localization algorithm that solves the kidnapped robot problem. We have used some sequences of the scene for training and some for testing as already mentioned in the Training subsection before.

From Figure 11, we can see our algorithm localize the robot very well disregard of the initial guess of its location. We have tested the different prior influence on the final average error and standard deviation. From the table and figure, The prior has a minor influence on the estimation.

VI. Conclusion

In this project, we implemented a re-localization pipeline using PoseNet as the front-end and GTSAM as the back-end. PoseNet was implemented using the VGG-16 architecture in Tensorflow and trained/tested on the following datasets:

- Kings College Dataset
- Shop Facade Dataset
- NCLT Dataset

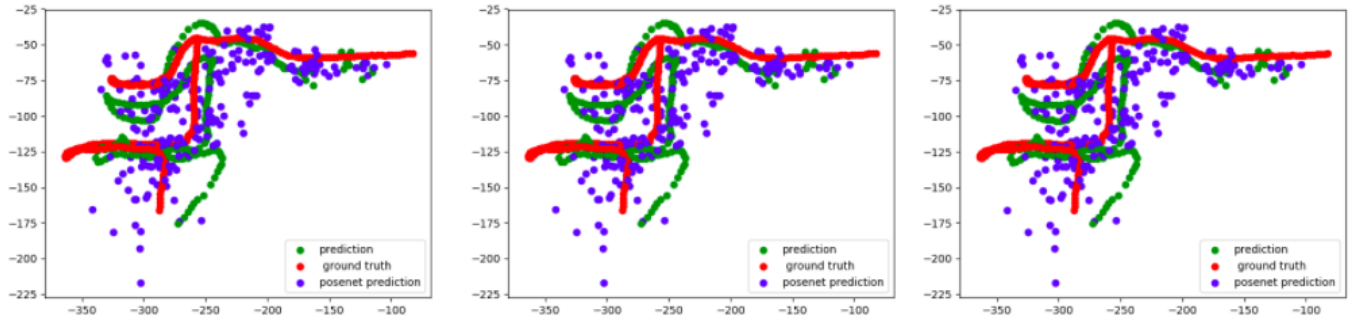

Fig. 11: Localization test with the respective priors on a partial area of NCLT dataset: (i) Bad Prior (ii) Medium Prior, (iii) Good Prior

GTSAM was implemented for factor graph optimization where PoseNet served as our sensor model and the odometry data as our motion model. These two were fused together through GTSAM to give a final factor graph representing the robot's trajectory.

Prior	Mean Error	Standard Deviation
Bad Prior	20.40912m, 0.205°	12.6454m,0.555°
Medium Prior	20.41747m, 0.205°	12.64738m, 0.555°
Good Prior	20.42789m 0.205°	12.65145m, 0.555°

TABLE II: Comparison of PoseNet's Performance with different location priors on the NCLT dataset

Through our work, we established that CNNs have the potential of being used in the domain of SLAM (*as a sensor model in our use case*). Thought it is an end-to-end model, one of it's limitations is that it doesn't generalize well on datasets with different distributions.

Through our experiments on the large scale outdoor NCLT dataset, we realized that there are some scenarios where our PoseNet implementation cannot do well. Most of the places in the dataset contain similar features which makes it extremely difficult to regress the actual location of the robot in the scene. For example, if two of the images contain just trees and a pedestrian pathway, it would be extremely difficult for the robot to localize. Currently, PoseNet only takes one frame to regress the robot's location. A possible extension to solve this problem can be to take a sequential stream of images as input to PoseNet rather than just the current frame. In that way, a relation between the robot's actual location in the scene can be regressed and matched.

Secondly, due to the limited size and quality of the datasets, we weren't able to train the regressor with a deeper network, like ResNet which may improve the accuracy significantly. Finally, a 360deg view of the scene provides a better estimate than just one image. We realized this from the fact that, if we use the scenes on the right side of the robot to train, and the left side of the robot *(on the same path)* to test, it leads to a significant inaccuracy in the robot's re-localization algorithm.

REFERENCES

[1] M. Cummins and P. Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. The International Journal of Robotics Research, 27(6):647665, 2008.

[2] N. Sunderhauf, F. Dayoub, S. Shirazi, B. Upcroft, and M. Milford. On the performance of convnet features for place recognition. arXiv preprint arXiv:1501.04158, 2015.

[3] Koller, D. and Friedman, N. Probabilistic Graphical Models: Principles and Techniques.The MIT Press, 2009.

[4] Kendall, Alex et al. PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization. 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 2938-2946.

[5] Changchang Wu. Towards Linear-time Incremental Structure from Motion, 3DV 2013.

[6] arXiv:1409.1556 [cs.CV]

[7] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, Frank Dellaert.iSAM2: Incremental smoothing and mapping using the Bayes tree. The International Journal of Robotics Research Vol 31, Issue 2, pp. 216 - 235.

[8] Carlone, Luca and Kira, Zsolt and Beall, Chris and Indelman, Vadim and Dellaert, Frank. (2014). Eliminating Conditionally Independent Sets in Factor Graphs: A Unifying Perspective based on Smart Factors. Proceedings - IEEE International Conference on Robotics and Automation. 10.1109/ICRA.2014.6907483.

[9] Places: A 10 million Image Database for Scene Recognition B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.

[10] ADAM: A Method for Stochastic Optimization, Kingma, D.P. and Ba, Jimmy Lei, https://arxiv.org/pdf/1412.6980.pdf

[11] Modelling Uncertainty in Deep Learning for Camera Relocalization, Alex Kendall and Roberto Cipolla, arXiv:1509.05909 [cs.CV]

[12] Learning to See by Moving, Pulkit Agrawal,Jitendra

Malik and Joao Carreira, arXiv:1505.01596 [cs.CV]
[13] CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction, Keisuke Tateno et al, arXiv:1704.03489 [cs.CV]
[14] M. Cummins and P. Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. The International Journal of Robotics Research, 27(6):647665, 2008.
[15] N. Sunderhauf, F. Dayoub, S. Shirazi, B. Upcroft, and M. Milford. On the performance of convnet features for place recognition. arXiv preprint arXiv:1501.04158, 2015.
[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. arXiv preprint arXiv:1409.4842, 2014.
[17] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 16531660. IEEE, 2014.
[18] Georgia Tech Smoothing and Mapping Tool, https://borg.cc.gatech.edu/
[19] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In Advances in Neural Information Processing Systems, pages 487495, 2014.