Modeling SLAM Performance Online using Pre-trained Convolutional Neural Networks

Adam Burhan, Supervised by Dr. Samer Nashed and Professor Liam Paull

August 9, 2024

Abstract

Autonomous robots must navigate safely and reliably in a wide variety of unpredictable environments, which is critical for their effective operation. A key challenge in achieving this is ensuring the robustness of Simultaneous Localization and Mapping (SLAM) algorithms, which are essential for navigation. However, predicting SLAM performance in diverse scenarios is difficult due to varying environmental conditions, sensor inputs and the inherent complexity of SLAM systems. To address this, we propose the development of a prediction module that can estimate SLAM performance accross different situations, potentially even generalizing to various sensor types. Our approach involves leveraging a pretrained AlexNet model, converting the regression problem of predicting Robot Pose Error into a classification task and employing the Earth Mover's Distance loss to account for the ordinal nature of the prediction targets. Preliminary results show that our model achieves better than random performance on a classification task using the KITTI dataset. Future work will focus on further optimizing the model through extensive hyperparameter tuning and exploring the addition of metadata and alternative architectures.

1 Introduction

Autonomous robots rely heavily on Simultaneous Localization and Mapping (SLAM) algorithms to navigate complex environments safely and reliably. Evaluating SLAM performance is crucial for enabling robots to assess their capabilities and make informed decisions, ensuring robust operation. Without accurate performance evaluation, robots may struggle to self-assess their perception capabilities, potentially leading to unsafe and suboptimal decisions, such as navigating into hazardous areas or failing to correctly map their surroundings.

Modeling SLAM performance is a complex task due to several inherent challenges. One significant difficulty is that small changes in the environment can lead to large variations in SLAM algorithm performance. Factors such as lighting, surface properties, and the presence of dynamic objects can drastically affect the accuracy of SLAM. Traditionally, SLAM performance evaluation relies on the availability of ground truth data, which is essential for benchmarking and developing SLAM algorithms. However, given the volatile nature of real-world environments where robots are deployed, it is crucial to also have an online mechanism that can assess whether a given situation is within the SLAM algorithm's operational limits.

An ideal solution would be a model that can assess SLAM performance in real-time, as the algorithm operates, to detect when the algorithm may struggle in a given situation. Such a model would enable robots to continuously evaluate and adapt their SLAM strategies based on current performance predictions, enhancing their ability to navigate safely and effectively in diverse environments.

Our approach leverages convolutional neural networks (CNNs) to create an adaptable performance model for SLAM. CNNs are particularly well-suited for this task because they excel at automatically learning hierarchical features from raw data, which is crucial given the high-dimensional and complex nature of sensor data. By using CNNs, we can avoid the use of hand-engineered features and directly utilize sensor data and metadata to predict SLAM performance. Although we train the CNN on popular benchmark datasets like KITTI [5], the resulting performance model is designed to make online predictions in real-time, assessing SLAM performance as the algorithm operates. Our hypothesis is that the features learned from the model provide a good scene understanding and can potentially generalize to other environments.

To implement this model, we used 11 sequences from the KITTI dataset, focusing on the Relative Pose Error (RPE) as our metric [11]. Given the dataset's relatively small size, we employed transfer learning with AlexNet [10], fine-tuning the model for our specific task. Our primary focus was on developing a performance model tailored to ORB-SLAM2 in a stereo setting [14]. We chose this approach because transfer learning allows us to leverage pre-trained networks that have been trained on much larger datasets and potentially have knowledge that can be useful to our specific task [21].

Our experiments included both regression and classification approaches, using quantiles for discretizing RPE data. We explored various configurations, including different loss functions and learning rates, and techniques like weight freezing. These efforts have provided valuable insights into how CNNs can be used to model SLAM performance, highlighting areas for further refinement. While our current focus is on optimizing the model and improving its accuracy for ORB-SLAM2, future work will involve exploring additional metadata and testing the model's generalizability to ensure its applicability across various SLAM systems.

Our preliminary results show better-than-random accuracy while only using raw sensor data, which encourages us to further refine our model and experiment with other techniques such as adding metadata and exploring other CNN architectures.

2 Related Work

The idea of an autonomous robot having a self-assessment component is not new. It was first introduced as introspection in robotics by Morris et al. [13], who provided a framework for self-assessment to handle operational uncertainties in field-capable robots. This concept has been further explored and specifically applied to perception systems.

For SLAM in particular, approaches for developing an introspection component, or in other words, modeling the SLAM system's performance, generally vary along four primary axes. First, are the predictions made offline, before deploying the robot and using prior knowledge of the operating environment [15, 12], or online, as the robot is operating and receiving sensor data [3, 2, 1, 16, 4, 8]? Second, how is the performance modeled: as a binary "failure" and "nominal" classification problem [3] or as a regression problem [15, 12, 2, 1, 16, 8]? Third, are the predictions made regarding the entire algorithm [15, 12, 3, 2, 1, 8] or specific subsets of the input [16, 9]? Finally, do the performance models use hand-engineered features [15, 12, 3, 1], process raw data [3, 16, 9], or a mix of both [8]?

While offline approaches [15, 12] work well, they require both environment and system features. This limits their applicability since acquiring the environment features requires prior knowledge about the environment or even the planned route, which is not always possible.

Regarding online prediction approaches, they differ mainly based on how they define the notion of performance. For instance, IV-SLAM [16] performs feature extraction from the regions of the image expected to yield less noise by developing a feature reliability model using introspective learning. Another example is Muca-SLAM [9], which predicts which camera from a set of multiple cameras will provide the best features. Some approaches treat performance as a binary variable, predicting either failure or nominal behavior in a binary classification task, such as using support vector machines [3]. These approaches work well; however, they are restricted by the limitations of visual SLAM systems.

A more general notion of performance that can be applied to any SLAM algorithm is the magnitude of localization error for a given pose. In [2], the authors propose estimating this error using learning and then correcting the pose estimates by the predicted error. This approach, however, is very challenging since you not only need to predict the magnitude of the error but also the correct direction.

Two approaches attempt to solve an easier version of the problem, closest to ours, by only predicting the magnitude of the localization error, not the direction. In [1], the authors propose predicting pose error for entire trajectories using random forest regression trained on a set of global features generated from raw camera sensor data passed through 1-D pooling. In [8], they attempt to solve the same problem using a DNN-based model that takes both extracted features from the visual SLAM pipeline and its visual input. The system we aim to build is one with the ability to generalize to other types of sensor data and that does not need custom features to predict error frame-by-frame.

2.1 Broader Connetion to OOD

Out-of-Distribution (OOD) detection is an important concept in machine learning, used to identify when a model encounters data that differs significantly from what is was trained on. OOD detection is essential for maintaining the reliability of systems in unfamiliar or unpredictable scenarios, such as autonomous driving or medical diagnosis [19]. In the context of SLAM performance modeling, our approach can be viewed as a type of OOD detection. By predicting SLAM performance, the model identifies situations where the environmental conditions or sensor inputs fall outside the "space" of scenarios for which the algorithm was optimized. We think that this is an interesting connection that can be made and it might be worth potentially drawing from advanced OOD detection techniques and investigating whether they can be useful to us.

3 Methods

To evaluate the performance of SLAM algorithms and develop a robust prediction model, we followed a systematic approach encompassing data collection, performance metric definition, dataset generation, model setup and experimental validation. This section details each step of our methodology, providing an overview of how we implemented and tested our hypothesis using CNNs on sensor data.

Firstly, why even use CNNs for SLAM performance modeling? Well, there is the nature of the sensor data. In the case of ORB-SLAM2 [14], the input data consists of images captured by cameras. CNNs are particularly compatible with image data due to their ability to efficiently process and extract meaningful features from high-dimensional inputs. Convolutional layers in CNNs apply filters to local regions of an image, detecting patterns such as edges, textures, and shapes. This process reduces the dimensionality of the data while preserving critical spatial information, making CNNs potentially effective for image analysis tasks. CNNs typically learn hierarchical features from images, starting with low-level features such as edges and corners, progressing to mid-level features like textures and shapes, and eventually identifying high-level features such as objects and complex structures. This hierarchical learning allows CNNs to capture the essential characteristics of the visual data that are crucial for various tasks, including object recognition and scene understanding [21]. Our hypothesis is that the features learned by CNNs from images could be directly applicable to predicting SLAM performance. SLAM algorithms rely on accurately identifying and tracking features in the environment to construct maps and localize the robot. We propose that the same hierarchical features that CNNs learn from images are fundamental to this process. By training a CNN to predict performance metrics based on these features, we hypothesize that it may be possible to develop a model that forecasts the effectiveness of a SLAM algorithm in different environments. This approach aims to leverage CNNs' strengths in feature extraction and pattern recognition to potentially enhance the reliability and robustness of SLAM performance predictions. This is also why we considered doing transfer learning. Even though our task differs from image classification, we believe a pretrained network on a very large dataset like AlexNet or ResNet [6] has learned a great deal regarding characteristics of visual data.

3.1 Data Sources

Since we are using a supervised learning approach for our model, we need to have a labeled dataset. We utilized publicly available datasets and open-source SLAM implementations. The primary data source for our study was the KITTI visual odometry/SLAM benchmark, which provides a comprehensive dataset for evaluating SLAM algorithms. This dataset includes 11 sequences with ground truth trajectories, captured by cameras mounted on a vehicle navigating through urban and rural environments [5]. The ground truth data is generated using a high-precision GPS and IMU system, providing accurate pose information for each timestep.

In addition to the KITTI dataset, we employed two open-source SLAM algorithms: ORB-SLAM2 and LOAM. ORB-SLAM2 is a widely-used feature-based visual SLAM algorithm that leverages Oriented FAST and Rotated BRIEF (ORB) features for robust tracking and mapping. The source code for ORB-SLAM2 was obtained from its official GitHub repository. Similarly, LOAM (Lidar Odometry and Mapping) is a LIDAR-based SLAM algorithm known for its high accuracy and efficiency in various environments [20]. The source code for LOAM was also obtained from its official GitHub repository.

By utilizing these data sources and SLAM algorithms, we aimed to generate a comprehensive dataset of input images and corresponding performance metrics, which would serve as the foundation for training and evaluating our CNN-based SLAM performance model.

3.2 Performance Metric

In order to evaluate the performance of the SLAM algorithms, we utilized the relative pose error (RPE), a metric proposed in [11]. This method benchmarks SLAM performance by measuring the error of the correlated trajectory rather than comparing maps. This approach is advantageous as it focuses on the relative relations between poses and does not rely on a global reference frame, making it more robust to the inherent inconsistencies and scale variations that can occur in global map comparisons.

Poses and Coordinate Systems: A pose represents the position and orientation of the robot in a given coordinate frame and is typically represented as a transformation matrix $\mathbf{P}$ that includes both rotation $\mathbf{R}$ and translation $\mathbf{t}$ components:

$$\mathbf{P} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix}$$

where $\mathbf{R}$ is a 3x3 rotation matrix and $\mathbf{t}$ is a 3x1 translation vector (when in 3D). Poses are often provided in the world coordinate frame, which is a fixed reference frame. However, for certain calculations, it is useful to convert these poses into the robot's coordinate frame, where the robot is always at the origin.

Relative Motion and Poses: The relative motion between two poses $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ is the transformation that describes the change in position and orientation from i to $i+1$:

$$\mathbf{P}_{i\to i+1} = \mathbf{P}_i^{-1}\mathbf{P}_{i+1}$$

This represents how the robot moves from one timestep to the next.

Relative Pose Error (RPE): The Relative Pose Error measures the local accuracy of the SLAM algorithm by comparing the relative motion of the consecutive poses in the estimated trajectory and the corresponding motion in the ground truth trajectory. Using RPE allows for a more consistent and reliable assessment of SLAM performance, as it mitigates the effects of drift and accumulated errors over long trajectories.

To compute it, let let $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ be the ground truth poses at timesteps i and $i+1$, and $\hat{\mathbf{P}}_i$ and $\hat{\mathbf{P}}_{i+1}$ be the corresponding estimated poses. The RPE $\mathbf{E}_{i,i+1}$ is:

$$\mathbf{E}_{i,i+1} = \left(\mathbf{P}_i^{-1}\mathbf{P}_{i+1}\right)^{-1}\left(\hat{\mathbf{P}}_i^{-1}\hat{\mathbf{P}}_{i+1}\right)$$

This error matrix $\mathbf{E}_{i,i+1}$ captures both translational and rotational errors between the estimated and ground truth relative motions.

Translational and Rotational Errors: The translational component of the RPE $\mathbf{t}_{i,i+1}$ is the Euclidean distance between the translation vectors of $\mathbf{E}_{i,i+1}$ and the rotational component of the RPE $\mathbf{r}_{i,i+1}$ is the angle of the rotation matrix of $\mathbf{E}_{i,i+1}$. This can be computed as the angle-axis representation of the rotation matrix.

With this as our performance metric, we now have a way to generate a label per frame for our training dataset.

3.3 Dataset Generation Process

After downloading the grayscale images for the 11 sequences of the KITTI dataset for ORB-SLAM2 and the velodyne point clouds for the same sequences for LOAM, we needed an automated process to run these SLAM algorithms on each sequence. This process involved several steps: running the algorithms, collecting the output poses for each frame, calculating the Relative Pose Error (RPE) for each frame, and finally organizing the data into a directory structured to be compatible with PyTorch's DataLoader.

Step 1: Setting Up SLAM Algorithms Locally

The first step was to have the SLAM algorithms available on our local system. Fortunately, the GitHub repositories for ORB-SLAM2 and LOAM provided detailed documentation on how to install the necessary dependencies and clone the repositories locally. By following these instructions, we ensured that both SLAM algorithms were correctly set up and ready to be used.

Step 2: Running the Algorithms and Collecting Output

We ran ORB-SLAM2 and LOAM on all 11 sequences from the KITTI dataset. For each sequence, the algorithms produced an output trajectory file (`trajectory.txt`) containing the estimated poses for each frame. With these outputs, we now had the input data (grayscale images and point clouds), the estimated trajectories, and the ground truth trajectories. This comprehensive set of data prepared us for the next phase of generating the dataset in a well-structured format.

Step 3: Calculating the RPE for Each Frame

To generate meaningful labels for our dataset, we calculated the Relative Pose Error (RPE) for each frame. This involved comparing the estimated poses from ORB-SLAM2 and LOAM with the ground truth poses from the KITTI dataset. For each sequence, we computed the translational and rotational errors as follows:

- **Translational Error**: Calculate the Euclidean distance between the translation components of the estimated and ground truth poses.
- **Rotational Error**: Determine the angular difference between the rotation matrices of the estimated and ground truth poses.

Using these calculations, we generated a performance metric for each frame, providing a label for our dataset.

Step 4: Structuring the Dataset for PyTorch's DataLoader

The final step was to organize the dataset into a directory structure that would be compatible with PyTorch's DataLoader and to also split the data in order to have a training and validation dataset (80/20 split). We structured the data as follows:

- **Images and Point Clouds**: Store the input images and point clouds in separate directories for each sequence.
- **Labels**: Save the calculated RPE values in corresponding label files.
- **Directory Layout**: Ensure that each sequence's data is organized in a manner that allows easy access by the DataLoader.

This structure facilitated efficient data loading and preprocessing during model training.

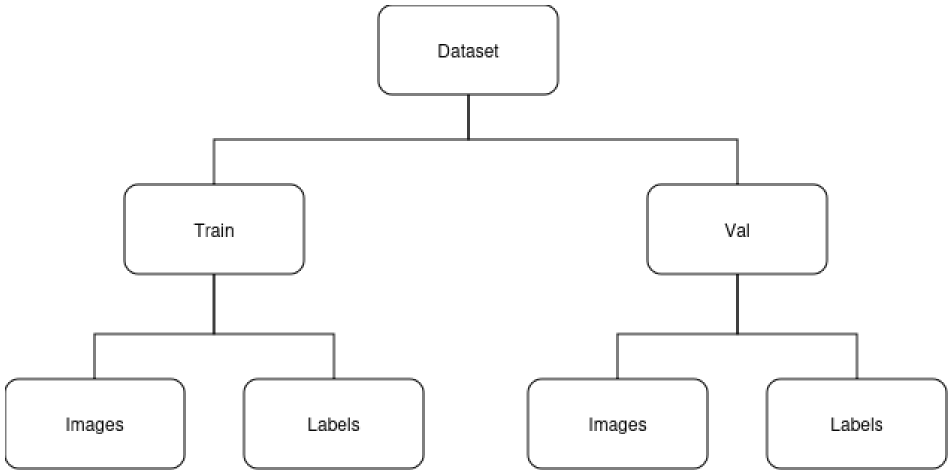

Figure 1: Directory layout for dataset

3.4 Discretizing the Data

There is a reference [18] that states that in the machine learning community, a commonly seen practice is the transformation of regression problems into classification problems since such a reformulation can often yield better performance.

At this stage, we have the data and labels for a regression problem. However, as mentioned in [18], a reformulation into a classification problem can be beneficial. Therefore, we decided to also set up our problem as a classification task. To achieve this, we quantized our data by sorting all the values for each RPE component and dividing them into nearly equal sections, or quantiles. In our case, we chose to use 5 quantiles, but in the future, we plan to explore using other numbers of quantiles. Each RPE value is now associated with a quantile from 1 to 5. This transformation allows us to treat our problem as a classification task, where instead of predicting the exact value for each RPE component, we predict the quantile to which it belongs.

By transforming the regression problem into a classification problem, we aim to leverage the potential benefits of classification algorithms, which may yield better performance in terms of accuracy and robustness.

3.5 Earth Mover's Distance and EMD-Squared Loss

For the regression formulation of the problem, we would typically use the Mean Squared Error (MSE) loss function. In the classification reformulation, the commonly used loss function is the Cross Entropy (CE) loss function. However, a limitation of the Cross Entropy loss in our context is that it does not account for the notion of distance between classes, which is present since we have quantized our data into ordered classes.

The softmax cross-entropy loss is a log-likelihood-based loss function that combines a softmax layer with a cross-entropy loss. For a single-label classification problem with C classes, a network's softmax layer outputs a probability distribution $\mathbf{p}$ of length C, with its i-th entry p_i being the predicted probability of the i-th class. The ground truth is represented as a binary vector $\mathbf{t}$ of length C. The cross-entropy loss between prediction $\mathbf{p}$ and ground truth vector $\mathbf{t}$ is defined as:

$$L_{\mathrm{CE}}(\mathbf{p}, \mathbf{t}) = -\sum_{i=1}^{C} t_i \log(p_i)$$

When the k-th class is the ground truth label ($t_k = 1$ and $t_i = 0$ for $i \neq k$), the differentiation of $L_{\mathrm{CE}}(\mathbf{p}, \mathbf{t})$ is:

$$L'_{\mathrm{CE}}(\mathbf{p}, \mathbf{t}) = -\frac{p'_k}{p_k}$$

This formulation shows that the backpropagation in a neural network using cross-entropy loss depends only on p_k, the probability of the correct class, and does not take into account the distances between incorrect classes.

To address this issue, we use the Earth Mover's Distance (EMD), also known as the Wasserstein distance, which measures the minimum amount of work required to transform one distribution into another. EMD is particularly useful in scenarios where the classes have an inherent ordering or notion of distance between them, as it considers the cost of moving probability mass between classes.

The EMD-Squared loss function is defined to incorporate the benefits of EMD into a differentiable loss function that can be used in training neural networks. In [7], it is shown that under certain conditions, EMD can be simplified to a closed-form solution, making it computationally efficient. These conditions are:

- The distributions $\mathbf{p}$ and $\mathbf{t}$ to be compared must have equal mass, which is satisfied if $\mathbf{p}$ is produced by a softmax layer.
- The ground distance matrix D must have a one-dimensional embedding, which is satisfied in ordered-class classification problems.
- The distributions $\mathbf{p}$ and $\mathbf{t}$ must be sorted vectors, which is naturally satisfied in our problem setup.

Given these conditions, the normalized EMD can be computed exactly as:

$$\mathrm{EMD}(\mathbf{p}, \mathbf{t}) = \left(\frac{1}{C}\right)^{1/l} \|\mathrm{CDF}(\mathbf{p}) - \mathrm{CDF}(\mathbf{t})\|_l$$

where $\mathrm{CDF}(\cdot)$ returns the cumulative density function of its input. Using $l = 2$ for Euclidean distance and dropping the normalization term, the final EMD-Squared loss L_{EMD} is given by:

$$L_{\mathrm{EMD}}(\mathbf{p}, \mathbf{t}) = \sum_{i=1}^{C} (\mathrm{CDF}_i(\mathbf{p}) - \mathrm{CDF}_i(\mathbf{t}))^2$$

By incorporating the EMD-Squared loss in our classification setup, we leverage the underlying distances between the quantized RPE values.

3.6 Model Architecture and Training Strategy

To develop our SLAM performance model, we leveraged a modified AlexNet architecture, adapted for stereo images and SLAM performance classification.

3.6.1 Model Architecture

AlexNet, developed by Krizhevsky et al. [10], was the winning model in the ImageNet Large Scale Visual Recognition Challenge 2012. The model consists of five convolutional layers and three fully connected layers. The first two convolutional layers are followed by max pooling and normalization layers. The fifth convolutional layer is followed by a max pooling layer. The first two fully connected layers contain 4096 neurons each and the final fully connected layer contains 1000 neurons for the target class scores. The authors used Rectified Linear Units (ReLUs) as activation functions and applied dropout as a regularization technique to reduce overfitting.

Given that our input data consists of stereo images (left and right images), we modified the initial layer to accommodate this. Specifically, the first convolutional layer of AlexNet, which originally expects 3-channel images, was modified to accept 6-channel images by stacking the stereo images along the channel dimension. Additionally, the final layer, which initially was a fully connected layer with 1000 outputs since the original data AlexNet was trained on was for a 1000-way classification task was replaced with two fully connected layers to predict the Relative Pose Error (RPE) components, with each layer outputting 5 quantiles.

3.6.2 Training Strategy

Our training strategy centered around transfer learning, leveraging the pre-trained AlexNet model to benefit from the features learned from the large ImageNet dataset. Transfer learning aims at improving the performance of target learners on target domains by transferring the knowledge contained in different but related source domains. This approach reduces the dependence on a large number of target-domain data for constructing effective models [21]. Given that our dataset is small for the model complexity we aim to use, we hypothesized that using a pre-trained model would help mitigate the risk of overfitting. Starting with pre-trained weights from AlexNet provides a better initialization point because the model has already learned a variety of general features from the large and diverse ImageNet dataset. These pre-learned features can be adapted to our specific task, allowing the model to generalize better and reducing the chance of overfitting on our limited dataset.

We used a heuristic provided by [17] regarding finding the best way to fine-tune with respect to target dataset size. It involved the following steps:

- **Layer Freezing and Unfreezing**: Initially, we froze all layers except the ones we modified or added. We gradually unfroze additional layers to fine-tune the model further, allowing more layers to adapt to our specific task.
- **Learning Rates**: We used a higher learning rate for the new or modified layers which were initialized randomly and a lower learning rate for the pre-trained layers to ensure stable training.
- **Loss Functions**: We experimented with different loss functions:
  - Cross Entropy (CE) Loss for initial classification experiments.
  - EMD-Squared Loss to capture the ordering in our quantized data.
  - Mean Squared Error (MSE) Loss for the regression formulation.
- **Problem Formulation**: We confirmed that formulating the problem as a classification task, rather than regression, yielded better performance. The quantization of RPE values into 5 classes (quantiles) allowed us to use classification techniques effectively.

3.6.3 Final Model Configuration

The best performing model was configured with:

- The first convolutional layer modified to accept 6 channels.
- The final fully connect layer replaced with two fully connected layers that output 5 quantiles for each RPE component.
- Learning rates set to 0.001 for the modified and added layers, 0.0001 for the pre-trained layers in the classifier part of AlexNet and 0.00001 for the last convolutional layer of AlexNet's feature extraction part

By systematically exploring different configurations, we identified a promising model for our SLAM performance classification task. However, due to time constraints, we were unable to test all possible configurations. In future work, we plan to use a more thorough and systematic approach to hyperparameter optimization to further improve the model's performance.

3.7 Experimental Workflow

Our experimental workflow was as follows:

- **Hypothesis Formulation**: We began by formulating a hypothesis regarding model configuration, hyperparameters, or training strategies.
- **Implementation**: The hypothesis was implemented in code, incorporating necessary changes or additions to the model and training scripts.
- **Image Creation**: Given the size of the model and the computational requirements, we created a Docker image containing the entire environment, including dependencies and the updated code.
- **High-Performance Computing**: The Docker image was then submitted to Compute Canada's high-performance computing clusters, as the model was too large to train on a local CPU. These clusters provided the necessary computational power to train and validate our model efficiently.
- **Result Collection and Analysis**: Once the training and validation were completed, we collected the results, which included performance metrics and model outputs. These results were then analyzed to evaluate the effectiveness of the hypothesis and inform subsequent experiments.

4 Experiments and Results

The primary purpose of our experiments was to test our hypothesis that using convolutional neural networks with transfer learning can effectively model the performance of SLAM algorithms. Each experiment was designed to isolate and evaluate different aspects of this hypothesis, including the effectiveness of transfer learning, the impact of different loss functions, and the generalization capabilities of the model. By systematically varying the experimental conditions and analyzing the results, we aim to gain a comprehensive understanding of the strengths and limitations of our approach.

4.1 Experiment 1: Initial Model Setup with Frozen Layers

Purpose:

The goal of this experiment was to establish a baseline performance using a modified AlexNet where only the newly added layers (the first modified convolutional layer and the two added fully connected layers for each RPE component) were trainable. All other layers from the pre-trained AlexNet were frozen.

Results:

The model's confusion matrices for this configuration for each RPE component are shown in Figure 2. This initial setup serves as the starting point for further experiments and as we can see with only the modified and added layers being trainable, the classification accuracy is close to random.

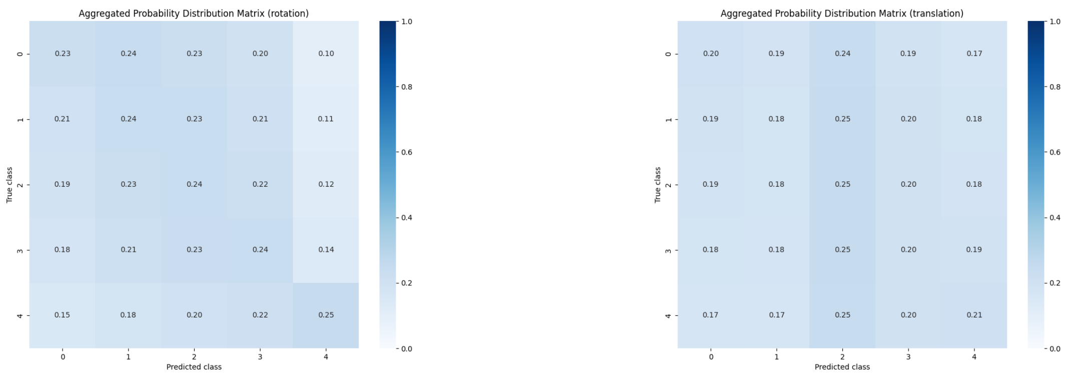

Figure 2: Aggregated probabilities matrices for the initial model setup with most layers frozen.

4.2 Experiment 2: Gradual Unfreezing of Layers

Purpose:

This experiment aimed to determine the impact of unfreezing additional layers on the model's performance. The approach was to gradually unfreeze layers, starting from the classifier part of AlexNet and moving towards the final convolutional layer of the feature extraction part.

Steps:

- **First Configuration:** Only the modified/added layers were unfrozen (initial model setup).
- **Second Configuration:** The last fully connected layer of AlexNet's classifier part was unfrozen.
- **Third Configuration:** Both fully connected layers of AlexNet's classifier part were unfrozen.
- **Final Configuration:** Same as the third configuration with the addition of the last convolutional layer of the feature extraction part unfrozen.

Results:

The gradual unfreezing resulted in improved performance, with the best results observed for the final configuration. Training and validation losses for each configuration are visualized in Figures 3 and 4 where the improvement is clearly visible. In Figure 5 we see the final configuration model's aggregated probability distribution matrices for each RPE component and although it is still not very accurate, there is a big improvement when comparing with the initial model setup and we have a better than random classification accuracy.

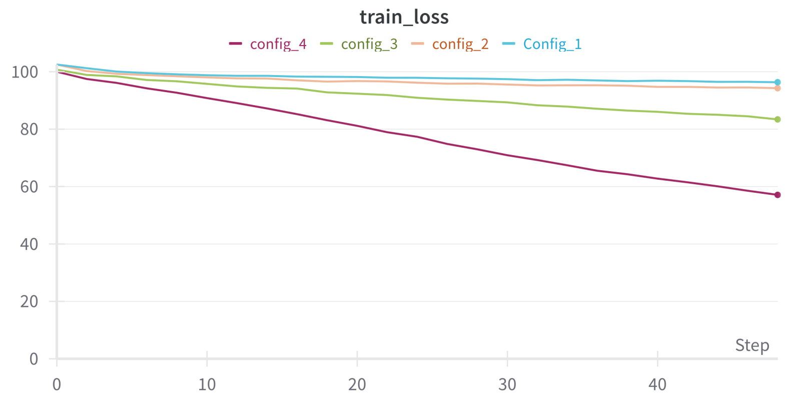

Figure 3: Training losses for each configuration.

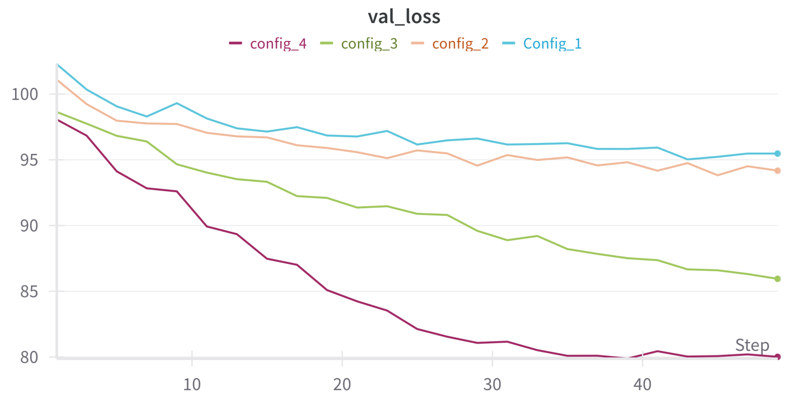

Figure 4: Validation losses for each configuration.

4.3 Experiment 3: Comparison of Loss Functions

Purpose:

This experiment was conducted to compare the performance of two loss functions: Cross Entropy loss and Earth Mover's Distance squared loss, for the best configuration obtained from Experiment 2.

Results:

Surprisingly, both loss functions yielded similar performance results, which was contrary to our initial hypothesis that EMD squared loss would outperform CE loss due to its consideration of class distance. The aggregated probability distribution matrices for each loss function are shown in Figures 5 and 6

5 Future work

Our preliminary results demonstrate that by utilizing raw sensor data and a pretrained CNN model, we can achieve performance that surpasses random chance for our classification task. Moving forward, we plan to conduct a more extensive hyperparameter search to optimize the model further. Additionally, we intend to explore the incorporation of metadata and experiment with alternative model architectures to enhance performance.

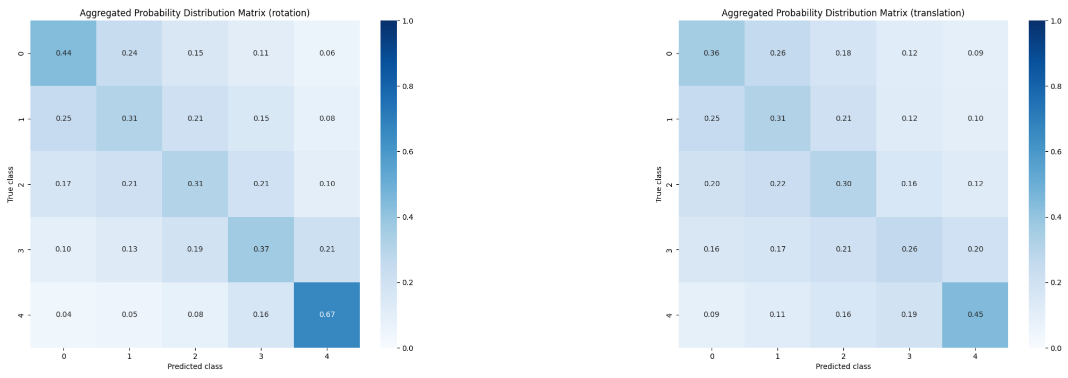

Figure 5: Aggregated probabilities matrices for the final configuration (EMD squared loss).

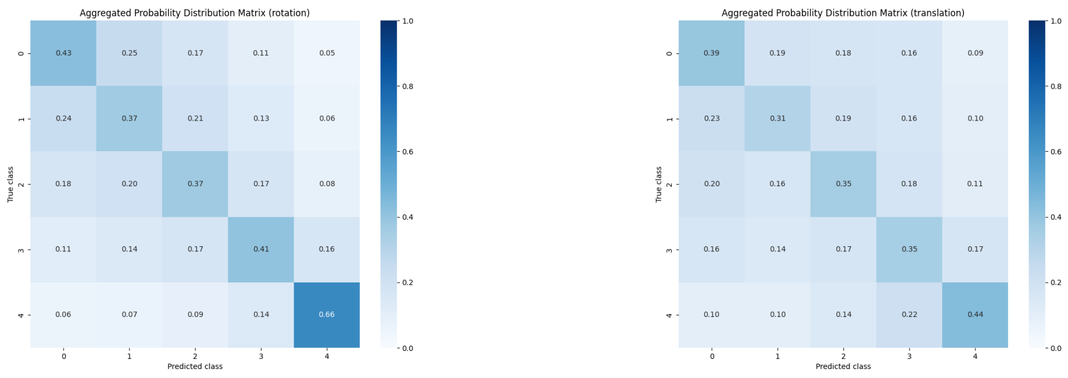

Figure 6: Aggregated probabilities matrices for the final configuration (CE loss).

Interestingly, our current results suggest that the model is best at identifying images likely to result in poor SLAM performance. This observation warrants further investigation. One potential hypothesis is that images with very few discernible features may correlate with poor SLAM performance, as they provide insufficient information for effective localization and mapping. Exploring this hypothesis could provide valuable insights into the relationship between image features and SLAM performance, guiding future improvements in our model.

References

[1] Islam Ali, Bingqing Wan, and Hong Zhang. Prediction of slam ate using an ensemble learning regression model and 1-d global pooling of data characterization. *arXiv preprint arXiv:2303.00616*, 2023.

[2] Zayed Alsayed, Guillaume Bresson, Anne Verroust-Blondet, and Fawzi Nashashibi. 2d slam correction prediction in large scale urban environments. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5167–5174. IEEE, 2018.

[3] Helen Carson, Jason J Ford, and Michael Milford. Predicting to improve: Integrity measures for assessing visual localization performance. *IEEE Robotics and Automation Letters*, 7(4):9627–9634, 2022.

[4] Shreyansh Daftry, Sam Zeng, J Andrew Bagnell, and Martial Hebert. Introspective perception: Learning to predict failures in vision systems. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1743–1750. IEEE, 2016.

[5] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Le Hou, Chen-Ping Yu, and Dimitris Samaras. Squared earth mover's distance-based loss for training deep neural networks. *arXiv preprint arXiv:1611.05916*, 2016.

[8] Tianyi Hu, Tim Scargill, Ying Chen, Guohao Lan, and Maria Gorlatova. Dnn-based slam tracking error online estimation. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–3, 2023.

[9] Pavel Karpyshev, Evgeny Kruzhkov, Evgeny Yudin, Alena Savinykh, Andrei Potapov, Mikhail Kurenkov, Anton Kolomeytsev, Ivan Kalinov, and Dzmitry Tsetserukou. Mucaslam: Cnn-based frame quality assessment for mobile robot with omnidirectional visual slam. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 368–373. IEEE, 2022.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[11] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots*, 27:387–407, 2009.

[12] Matteo Luperto, Valerio Castelli, and Francesco Amigoni. Predicting performance of slam algorithms. *arXiv preprint arXiv:2109.02329*, 2021.

[13] Aaron Christopher Morris. *Robotic introspection for exploration and mapping of subterranean environments*. Carnegie Mellon University, 2007.

[14] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[15] Enrico Piazza, Pedro U Lima, and Matteo Matteucci. Performance models in robotics with a use case on slam. *IEEE Robotics and Automation Letters*, 7(2):4646–4653, 2022.

[16] Sadegh Rabiee and Joydeep Biswas. Iv-slam: Introspective vision for simultaneous localization and mapping. In *Conference on Robot Learning*, pages 1100–1109. PMLR, 2021.

[17] Deepak Soekhoe, Peter Van Der Putten, and Aske Plaat. On the impact of data set size in transfer learning using deep neural networks. In *Advances in Intelligent Data Analysis XV: 15th International Symposium, IDA 2016, Stockholm, Sweden, October 13-15, 2016, Proceedings 15*, pages 50–60. Springer, 2016.

[18] Lawrence Stewart, Francis Bach, Quentin Berthet, and Jean-Philippe Vert. Regression as classification: Influence of task formulation on neural network features. In *International Conference on Artificial Intelligence and Statistics*, pages 11563–11582. PMLR, 2023.

[19] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *International Journal of Computer Vision*, pages 1–28, 2024.

[20] Ji Zhang, Sanjiv Singh, et al. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and systems*, volume 2, pages 1–9. Berkeley, CA, 2014.

[21] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.