Computer Vision Coursework

Student ID: 11340063

Introduction

The goal of this coursework is to improve computer vision algorithmic skills with an emphasis on robotics object recognition. The performance of Convolutional Neural Networks (CNNs) for deep learning and conventional techniques are compared. Specifically, K-Nearest Neighbours (KNN) and Support Vector Machines (SVM) are used as classifiers, and the Scale-Invariant Feature Transform (SIFT) algorithm is used for local feature extraction. Extensive investigation and assessment show that although CNNs are usually very accurate, they require larger amounts of data and processing power to train. On the other hand, conventional techniques such as SIFT provide robustness to variations in the scene and computational efficiency, but they may not work well in complex scenes. Examining these methods offers important information about the trade-offs involved and how well they apply to robotics research.

Dataset Description

One popular benchmark dataset in computer vision is CIFAR-10. It is divided into 10 classes and includes 60,000 colour images, each measuring 32 by 32 pixels. Commonplace items like cars, trucks, ships, frogs, dogs, cats, deer, and aeroplanes are included in these classes. With an equal number of images in each class, the dataset is divided into 50,000 training and 10,000 test classes. For image classification tasks, CIFAR-10 is a standard dataset that can be used to train and assess different machine learning and deep learning models.

CNN for Image Classification

This approach used the CIFAR10 dataset to classify images using convolution neural networks (CNNs). CNNs are a particular kind of deep learning neural network that are used to perform a variety of visual imagery tasks, such as object detection, image segmentation, and image classification. The CNN architecture was modified from *(Çalik and Demirci (2018))*, where 4 convolution layers and 2 fully connected layers were proposed for the model. The model's learning rate is first set at 0.1 and is decreased by 10% every 350 epochs. To train the model, 100000 epochs are utilised in total. These parameters were adjusted to fit the available resources .

The final architecture of the image classification model comprised 4 convolution layers and 2 fully connected layers. Each of the four convolution layers contained 32, 64, 128, and 256 filters. The network is able to recognise ever-more complex patterns and structures in the images because each layer has more filters than the previous one. A 3x3 kernel is used in all convolution layers to capture fine details in input

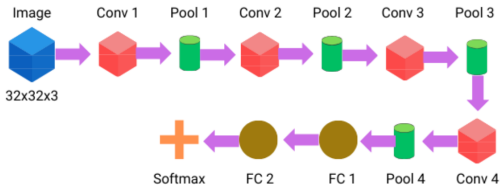

Fig. 1. Architecture of CNN model

images while limiting the number of parameters. All convolution layers had a stride of 1, which maximised the image's coverage and allowed for the detection of features at various spatial locations. Every convolutional layer is followed by max-pooling layers with a pool size of (2, 2). By downsampling the feature maps, these layers preserve the most crucial information while decreasing the spatial dimensions of the maps. The process of downsampling reduces computational complexity and avoids overfitting by concentrating on the most significant features. After every pooling layer, dropout layers (0.2, 0.3, 0.4, and 0.5) are added with increasing dropout rates. Dropout is a regularisation technique that forces the network to learn more robust features and lowers the risk of overfitting by randomly dropping a portion of connections during training. The dense layer, which has 512 units, and the output layer, which has softmax activation, are the two fully connected layers. In the classification task, the softmax output layer generates probabilities for each class, while the dense layer uses the flattened feature maps to learn high-level features. Figure 1 depicts the architecture of the CNN model used.

Subsequently, the model is assembled utilising the Adam optimizer, which is popular for its effectiveness in training deep neural networks and has a learning rate of 0.001. The CIFAR-10 dataset's multi-class classification feature is consistent with the selection of the categorical cross-entropy loss function. In order to balance computational efficiency with effective parameter updates, the model processes training data in batches of 128 samples per gradient update during training. The model is trained for 200 epochs, going through several iterations over the whole dataset to progressively improve its internal representations and classification accuracy.

The performance of the CNN model for image classification on the CIFAR-10 dataset was optimised through manual hyperparameter tuning. The hyperparameters 'batch_size', 'epochs', 'learning_rate', 'dense_units', and 'optimizer' have been selected for tuning.

The number of training examples used in each training cycle iteration is set by the 'batch_size' hyperparameter. Testing various batch sizes, including 64, 128 and 256, allows us to assess the trade-offs between model convergence and computational efficiency.

The amount of complete passes through the training dataset

that are made during the training process is indicated by the 'epochs' hyperparameter values. Through experimenting with various values, like 50, 100 and 200, we evaluate how training time affects model performance. Increasing the number of epochs may improve convergence and accuracy, but there is a risk of overfitting if the model becomes overly adept at memorization of the training set.

During training, the step size at which the model parameters are updated is managed by the 'learning_rate' hyperparameter. We explore the effect of the learning rate on the rate of convergence and the stability of the training process by experimenting with various values, such as 0.001 and 0.01. While a higher learning rate might hasten convergence, it also runs the risk of overshooting the ideal solution or creating instability during the training phase.

The number of neurons in the CNN model's fully connected dense layer is set by the 'dense_units' hyperparameter. We investigate the model's ability to extract complex patterns from the extracted features and learn them by experimenting with various values, such as 256 and 512. Increasing the number of dense units could help the model learn more complex representations, but it could also make overfitting more likely if the model's capacity is greater than the dataset's complexity.

The optimisation algorithm that updates the model parameters during training is specified by the 'optimizer' hyperparameter. We examine options such as 'adam' and'sgd' (Stochastic Gradient Descent) in order to compare the convergence properties and performance of various optimisation methods. The Adam optimizer is renowned for its flexible learning rate characteristics and effective convergence across diverse datasets. In contrast, SGD is a conventional optimisation technique that may necessitate meticulous adjustment of learning rate and momentum to achieve peak efficiency.

All things considered, manual hyperparameter tuning enables a methodical investigation of various configurations to determine which combination produces the best performance for the specified task and dataset. We can improve the CNN model's capacity to efficiently learn from the data and generate precise predictions on the CIFAR-10 dataset by carefully choosing and assessing hyperparameters.

The model can be trained on the remaining 80% of the dataset while keeping a sizeable portion for validation by designating 20% of the original dataset as a validation set. This allows the model to evaluate its performance on unseen data during training while also learning from the training set. In order to reliably assess the model's performance and ensure that it generalises well to new data, a validation split of 0.2 strikes a compromise between having an adequate amount of training data to identify meaningful patterns and an adequate validation set. By using this technique, one can identify overfitting, adjust hyperparameters, and make well-informed choices regarding the design of the model and training methods.

The CIFAR-10 dataset's labels, which were initially represented as integers between 0 and 9, are transformed into categorical one-hot encoded vectors for data preprocessing. With the help of this encoding scheme, the model is able to learn to predict the probability distribution for each class, allowing it to generate accurate predictions for every image. In addition, by dividing each pixel value by 255, the pixel values of the input images are transformed into floating-point numbers and normalised to a range between 0 and 1. In order to prevent bias in the model's training process caused by variations in pixel intensity scales, this normalisation process normalised the pixel values across all images. By limiting large gradients and enabling more effective optimisation, normalisation also aids in stabilising the training process.

Google Collab was intended to be used for running the hyperparameter tuning portion of the code. The model's computation was accelerated by the T4 GPU. Owing to the restricted GPU resources available in Google Collab, the final models were executed locally in a Jupyter Notebook. Refer to *(APPENDIX)* to view the Python notebooks.

Analysis of CNN model

The model was tested using various parameter combinations in order to determine the ideal accuracy value. Initially, the number of units in the dense layer, learning rate, and optimizer were varied for a batch size of 64 over 100 epochs in order to test the model. In this instance, it was found that the models' accuracy was extremely poor. It was also noted that the models were overfitting. Table 1 displays the accuracy of various models. The batch size of 64 was considered inappropriate for the reasons listed above.

Dense Layer Units	Optimizer	Learning Rate	Accuracy
512	Adam	0.001	0.803
512	SGD	0.001	0.412
512	Adam	0.01	0.1
512	SGD	0.01	0.737
256	Adam	0.001	0.802
256	SGD	0.001	0.409
256	Adam	0.01	0.1
256	SGD	0.01	0.728

Table 1. Accuracy of Models with batch size 64 and epochs 100

After the analysis, 256 batch sizes with 100 epochs were taken into consideration. While the accuracy in this scenario was higher than when using a batch size of 64, overfitting was clearly occurring. Additionally, it was noted that 256 was a computationally demanding batch size. For the reasons listed above, batch size of 256 was disregarded. Table 2 displays the obtained accuracy values.

The aforementioned analysis also revealed that Adam outperforms the model and operates optimally at a learning rate of 0.001. Nevertheless, SGD optimizer performs poorly for the specified model, and it is found that SGD yields significantly better results when the learning rate is 0.01. Therefore, we only take into account two scenarios for batch sizes of 128: Adam optimizer, which has a learning rate of 0.001, and SGD optimizer, which has a learning rate of 0.01. After 500 epochs, the SGD optimizer with a learning rate of 0.01 produced an accuracy of 0.813. Even though it provides greater

Dense Layer Units	Optimizer	Learning Rate	Accuracy
512	Adam	0.001	0.804
512	SGD	0.001	0.227
512	Adam	0.01	0.1
512	SGD	0.01	0.574
256	Adam	0.001	0.806
256	SGD	0.001	0.223
256	Adam	0.01	0.373
256	SGD	0.01	0.576

Table 2. Accuracy of Models with batch size 256 and epochs 100

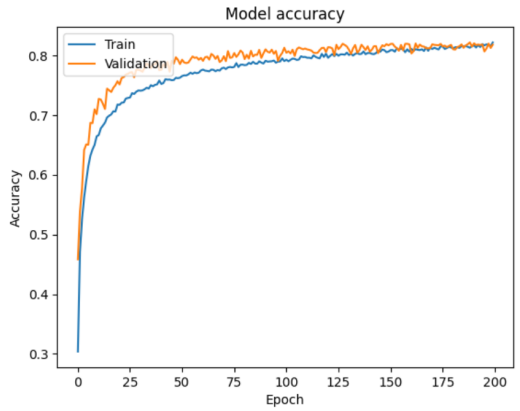

Fig. 2. Final Model Accuracy

accuracy, there are numerous variations on the model accuracy graph, which could be a sign of overfitting. Hence this model is not preferred.With a batch size of 128 and an Adam optimizer learning rate of 0.001, the accuracy is 0.809, and the model accuracy graph is likewise fairly stable, suggesting that overfitting is not present. Therefore, 128 batches with the Adam optimizer and a 0.001 learning rate are thought to be the ideal set of values.

We run the model with 512 and 256 dense units for 200 epochs each in order to compare the number of dense units in this instance. According to the results, a dense layer with 512 units performed better, suggesting that increasing the number of features extracted contributed to an increase in accuracy. Compared to the model with 256 units, the graph obtained for 512 units was more stable. Figure 2 and 3 displays the graphs obtained for 512 units. Table 3 displays the accuracy that each of the two models achieved.

Model	Accuracy
Baseline Model	**0.858**
Batch size = 128, Adam optimizer (Learning rate of 0.01), Dense layer units = 512, Epochs= 200	**0.813**
Batch size = 128, Adam optimizer (Learning rate of 0.01), Dense layer units = 256, Epochs= 200	0.809

Table 3. Accuracy of Final Models

The baseline model provides an accuracy of 0.858 based on Table 3. The model's efficiency is demonstrated by the accuracy of 0.813 it produced, even with different parameters. Hence, it is possible to achieve at least the baseline accuracy with improved resources and hyperparameter tuning.

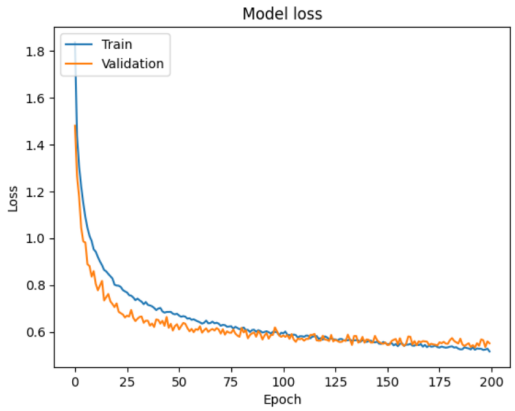

Fig. 3. Final Model Loss

Object Recognition using Local Features and Computer Vision Algorithm

First, the CIFAR-10 dataset is loaded. This dataset consists of training and test images with labels. OpenCV is used to instantiate the SIFT detector, which makes it possible to extract keypoints and the associated descriptors from each image. The KMeans algorithm is then used to cluster these descriptors—which identify characteristic local image features like edges, corners, and textures—into a visual vocabulary. In this case, the number of clusters, or visual words, is predefined and usually comes from computational limitations or empirical knowledge.

The SIFT descriptors are quantized to the closest visual word for every image, thereby mapping every local feature to a particular cluster centroid. The process produces a histogram representation called the Bag-of-Words (BoW), in which each bin of the histogram represents a visual word and the value of the bin indicates how frequently that word appears in the image. The BoW representation offers a concise yet informative summary of the image content by condensing the rich information captured by SIFT descriptors into a fixed-length vector, which makes it easier to perform subsequent analysis and classification tasks. This BoW encoding process is applied to both the training and test images, yielding BoW representations for every image in the dataset that are subsequently used for machine learning model evaluation and training.

After the training images' Bag-of-Words (BoW) representations are created, the code trains two different classifiers using these representations. The K-Nearest Neighbours (KNN) classifier, a straightforward yet powerful algorithm for classification tasks, is trained first. Based on the majority class among its k closest neighbours in the training dataset, the KNN classifier predicts the class labels of test instances. The model's bias-variance trade-off is influenced by the number of neighbours (k); smaller values may result in more flexible decision boundaries, but they also raise the possibility of

overfitting.

A more advanced Support Vector Machine (SVM) classifier is then trained after that. In this implementation, the SVC classifier and feature standardisation are combined via a pipeline that is made using scikit-learn's make_pipeline() function. To guarantee that every feature contributes equally to learning and to keep features with larger scales from controlling the optimisation process, the StandardScaler component makes sure that the input features are standardised. Effective model training is made possible by standardisation, which eliminates the mean and scales the data to unit variance across dimensions.

The primary classifier used for training and prediction is known as the Support Vector Classifier (SVC). Using kernel functions such as the Radial Basis Function (RBF) kernel in this implementation, SVC is a potent supervised learning algorithm that can handle non-linearly separable data by transforming input features into a higher-dimensional space. By doing this, SVC is able to create non-linear decision boundaries between various classes and capture intricate relationships between visual features that are extracted from images. Because of this ability, support vector machines (SVMs) are especially well-suited for image classification tasks where complex and nonlinear underlying relationships may exist between features and classes.

Through the process of training two classifiers—a KNN and an SVM—on the BoW representations of the training images, the code investigates various modelling strategies, spanning from elementary to advanced algorithms. This method enables the classification performance of these two models to be compared and sheds light on how successful each method is for the particular image classification task at hand.

Analysis of Object Recognition using Local Features and Computer Vision Algorithm

Using K-means and a cluster size of 100, the process starts with first clustering the SIFT descriptors. These cluster centroids are then used as features by K-Nearest Neighbours (KNN) and Support Vector Machine (SVM) classifiers. KNN classifier using 100 neighbours, yields an accuracy of 0.154. Several kernel functions are investigated for SVM, and Table 4 displays the findings for each kernel.

Model	Accuracy
SVM (kernel=rbf)	0.277
SVM (kernel=poly)	0.242
SVM (kernel=sigmoid)	0.224
SVM (kernel=linear)	0.268
KNN (100 neighbours)	0.154

Table 4. Accuracy of Models with cluster size 100

The cluster size is raised to 200 in order to improve the classification accuracy even more. Applying KNN with this increased cluster size results in an accuracy of 0.124. Thus, the analysis proceeds, concentrating on the 100-cluster case's most successful scenario: SVM with the 'rbf' kernel. The 'rbf' kernel SVM achieves an improved accuracy of 0.289 with the larger cluster size.

The pipeline incorporates cross-validation in an attempt to further hone the outcomes. Accuracy values of 0.289 and 0.291 are obtained by performing cross-validation with 5 and 10 folds, respectively. The dependability of the model's performance is demonstrated by the consistency of the results across various cross-validation folds. Table 5 displays the findings for 200 cluster sizes.

Model	Accuracy
SVM (kernel=rbf)	0.289
SVM (kernel=rbf) (5 cross fold)	0.290
SVM (kernel=rbf) (10 cross fold)	**0.291**
KNN (100 neighbours)	0.124

Table 5. Accuracy of Models with cluster size 200

It is clear from the thorough analysis of these data that better classification performance is obtained when the cluster size is increased to 200. Cross-validating the training set can also be noted as a way to increase accuracy. This result emphasises how crucial it is to experiment and fine-tune parameters when optimising machine learning models for particular datasets and tasks.

Comparison of the 2 Models

In computer vision, object recognition is essential for a variety of applications, including augmented reality and autonomous driving. Here, we compare a traditional Computer Vision (CV) method that uses Scale-Invariant Feature Transform (SIFT) with a deep learning-based approach that uses Convolutional Neural Networks (CNNs).

CNNs outperform SIFT-based techniques in terms of accuracy; they achieve 81.3% accuracy as opposed to SIFT's 29.1%. SIFT depends on manually created features and more basic classifiers, whereas CNNs are superior at deriving complex patterns directly from pixel values.

CNNs require more resources in terms of computational efficiency, particularly when they are being trained. Still, they are fast image processors during inference, especially when hardware acceleration is used. SIFT requires less computing power when extracting features, but for larger datasets, it can still consume a substantial amount of resources.

Because CNNs can learn hierarchical representations, they are generally more robust than SIFT and can provide better generalisation to unseen data. SIFT features might not have the same semantic understanding as CNNs, despite being intended to be invariant to variations.

The requirements of the task determine which of the two approaches to use. When real-time performance is critical and computational resources are scarce, SIFT may be adequate for simpler tasks. CNNs, on the other hand, are best suited for tasks that prioritise high accuracy and a large amount of labelled data.

State-of-the-art in computer vision for robotics

One type of computer vision task is object detection, which is locating and identifying objects within an image or video frame. Due to its ability to allow robots to sense and interact with their surroundings, this task is essential to robotics. Convolutional neural networks (CNNs), in particular, are deep learning techniques that have revolutionised object detection by offering incredibly accurate and effective solutions. YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) are two popular architectures for real-time object detection.

SSD was first presented by *(Wei Liu et al. in 2016)*, and it is intended to detect objects in real time using only one network forward pass. SSD makes direct predictions of object bounding boxes and class scores at multiple feature maps with varying resolutions, in contrast to Faster Region-CNN, which makes use of a different region proposal network. This allows SSD to maintain real-time performance while achieving high detection accuracy across a range of object scales. Because of its architecture, SSDs can effectively manage objects with varying sizes and aspect ratios inside of a single network. Proposed by *(Shaoqing Ren et al. in 2015)*, Faster R-CNN is a groundbreaking deep learning-based object detection framework. It is made up of two primary parts: a detection network that categorises and refines the region proposals that are generated by the region proposal network (RPN). Faster R-CNN performs better in real time on platforms with limited resources, but it can be computationally demanding and still achieve high accuracy. Using both of the aforementioned techniques, *(Kumar, A. and Srivastava, S. (2020))* seeks to attain high accuracy, surpassing 75%, with a 5–6 hour training duration. The algorithm effectively extracts feature information by using multi-scale feature maps and distinct filters with distinct default boxes. The creation of a robot to gather bottom sediments on tidal flats is depicted in *(Hatano et al., 2023)*. The robot must steer clear of debris, driftwood, and other obstacles while traversing muddy terrain. Next, image recognition was used to identify objects using the SSD.

YOLO creates a grid out of the input image and uses the grid cells to predict bounding boxes and class probabilities. YOLO is able to detect multiple objects per grid cell because each grid cell predicts multiple bounding boxes and associated class probabilities. In comparison to techniques like Faster R-CNN and SSD, YOLO sacrifices a little bit of accuracy but makes up for it in speed, which makes it ideal for real-time applications like robotics and driverless cars. *(Lahoti et al., 2024)* entails using a robotic arm for segregation in conjunction with an advanced YOLOv5 model for waste management on SoC IoT devices like Raspberry Pi. The main sensor is a webcam, which records real-time video feeds of the waste materials that are fed into the system. Real-time processing splits the video stream into its component frames. The model locates and categorises objects in each frame as it is analysed, assigning them to the relevant waste category. Serial communication is used to send the classification result produced by the SoC to an Arduino microcontroller. The data is interpreted by this microcontroller, which then transforms it into commands that the robotic arm's DC motors can follow. Using the classification input as a guide, the robotic arm physically sorts the waste by picking up objects and putting them in bins that are appropriate for each category. The model can perform even better if it is trained for more epochs in a better GPU environment. Its maximum performance was 0.65 precision and 0.5 recall.

Simultaneous Localization and Mapping, or SLAM, is another method that leverages deep learning. It describes a robot's or an autonomous system's ability to map its surroundings and figure out where it is in relation to them at the same time. RecSLAM, a multirobot laser SLAM system operating within a hierarchical robot-edge-cloud architecture, is covered in *(Huang et al., 2021)*. RecSLAM reduces robot computing stress and communication overhead by using edge servers to handle map fusion and SLAM computation. By optimising data flow between robots and edge servers, it can reduce latency by up to 39.31% when compared to current solutions. The system's modules for data collection, grouping, and offloading to edge servers for processing are designed to efficiently distribute SLAM workloads. The study discusses issues with multi-robot SLAM processing and suggests edge computing-based solutions to increase productivity and lower latency.

Convolutional neural networks (CNNs) with U-Net's architecture were created especially for semantic segmentation tasks. It has become widely used and successfully applied to numerous other computer vision tasks in addition to biomedical image segmentation. The contracting path of the U-Net architecture involves gradually downsampling the input image through convolutional and max-pooling layers to extract hierarchical features. This is followed by an expansive path that upsamples the features and combines them with skip connections to recover spatial information and produce a high-resolution segmentation map. The last layer generates a pixel-wise probability distribution over the class labels using a convolutional layer with a softmax activation function. Because of its compact design and effective use of context, U-Net's U-shaped architecture has made it a popular framework for semantic segmentation tasks. *(A. Attanasio et al)* segments soft tissues in endo-scopic images using spatiotemporal neural networks based on the U-Net. The networks can extract the correlation between consecutive frames in an endoscopic video stream because they are outfitted with Long Short-Term Memory and Attention Gate cells. This improves the accuracy of segmentation when compared to the standard U-Net. Using both Attention Gate blocks and Long Short Term Memory (LSTM) convolutional layers yields an accuracy of 83.77% and a precision of 78.42%. The findings could be useful for numerous independent tasks, including ablation, suturing, and debridement, even though they were first obtained in the context of surgical tissue retraction.

Conclusion

In conclusion, our project highlights the trade-offs between accuracy and computational efficiency by comparing deep learning-based CNNs with conventional CV methods using SIFT for object recognition. CNNs need a lot of processing power, even though they provide better accuracy and resilience. On the other hand, SIFT provides computational efficiency at the expense of accuracy, especially in complex scenarios. The decision between the two techniques ultimately comes down to the specific needs of the project; CNNs are preferred for high-accuracy tasks with plenty of resources, while SIFT is appropriate for environments with limited resources where real-time performance is crucial.

References

[1] Çalik, R.C. and Demirci, M.F. (2018). Cifar-10 Image Classification with Convolutional Neural Networks for Embedded Systems. [online] IEEE Xplore. doi:https://doi.org/10.1109/AICCSA.2018.8612873.

[2] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. and Berg, A.C. (2016). SSD: Single Shot MultiBox Detector. Computer Vision – ECCV 2016, [online] 9905, pp.21–37. doi:https://doi.org/10.1007/978-3-319-46448-0_2.

[3] Ren, S., He, K., Girshick, R. and Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. [online] arXiv.org. Available at: https://arxiv.org/abs/1506.01497.

[4] Kumar, A. and Srivastava, S. (2020). Object Detection System Based on Convolution Neural Networks Using Single Shot Multi-Box Detector. Procedia Computer Science, 171, pp.2610–2617. doi:https://doi.org/10.1016/j.procs.2020.04.283.

[5] Hatano, M., Manami Senzaki, Kawasaki, H., Chiaki Takasu, Yamazaki, M. and Yukiyoshi Hoshigami (2023). Robots traveling on muddy terrain for sampling bottom sediment in tidal flats. Artificial life and robotics, 29(1), pp.155–160. doi:https://doi.org/10.1007/s10015-023-00920-9.

[6] Lahoti, J., Sn, J., Krishna, M.V., Prasad, M., Bs, R., Mysore, N. and Nayak, J.S. (2024). Multi-class waste segregation using computer vision and robotic arm. PeerJ Computer Science, [online] 10, p.e1957. doi:https://doi.org/10.7717/peerj-cs.1957.

[7] Huang, P., Zeng, L., Chen, X., Luo, K., Zhou, Z. and Yu, S. (2021). Edge Robotics: Edge-Computing-Accelerated Multi-Robot Simultaneous Localization and Mapping. [online] arXiv.org. Available at: https://arxiv.org/abs/2112.13222.

[8] A. Attanasio et al., "A Comparative Study of Spatio-Temporal U-Nets for Tissue Segmentation in Surgical Robotics," in IEEE Transactions on Medical Robotics and Bionics, vol. 3, no. 1, pp. 53-63, Feb. 2021, doi: 10.1109/TMRB.2021.3054326. keywords: Image segmentation;Surgery;Streaming media;Robots;Task analysis;Biological tissues;Biomedical imaging;Medical robotics;computer assisted interventions;minimally invasive surgery;surgical vision

Appendix

The following drive link has the python notebooks used for coding along with the dataset and other necessary files:

https://drive.google.com/drive/folders/1GMt-Z3gECU7pQV4NIa62bA1hlKnhk6Nk?usp=sharing