INCA Summative

# Section A:

## Architectures:

The problem is to guide a mobile robot around a room in a wall following navigation task. The data was collected in four rounds from 24 sensors attached in a clockwise direction around the robot. There are 5456 total rows of data with each row of data having 24 columns of sensor data and one of four class labels.

As the problem is to input a row of sensor data and output a class label this is a classification problem. As the robot is travelling clockwise, intuitively the most important sensors will be the forward sensor and the left hand sensor. Therefore as an initial test, the left sensor value was plotted against the forward sensor value, where each point is coloured differently for each class label. There is no clear distinction between the data. Therefore, it is also a non-linearly separable classification problem.

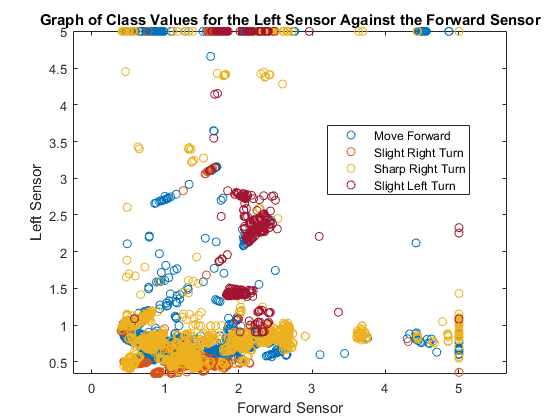


Figure Initial graph to assess what type of problem attempting to solve

**ASSUME PROBLEM IS CREATING A NN TO SOLVE FOR THIS PARTICULAR ROOM NOT A GENERIC WALL FOLLOWING ROBOT FOR EVERY ROOM**

To solve this problem I will investigate using a multi-layer perceptron (referred to as a MLP from here on). In addition to a MLP, a recurrent neural network (referred to as a RNN) and a Radial Basis Function Network (RBFN).

### Multi-Layer Perceptron:

A multi-layer can be thought of as consisting of multiple perceptron’s. Multi-layer perceptron’s train on a set of inputs and outputs and learn to model the correlation between them. Training refers to the adjustment of weights and biases to minimise the output error, which are adjusted through backpropagation. A MLP consists of an input layer – which receives the input or signal, one or more hidden layers – which transform the input into something the output layer can understand, and the output layer – which makes a decision or prediction. The hidden layer or layers have a nonlinear activation function

With one hidden layer and enough neurons in the hidden layers, an MLP is a universal approximator [**ADD REF**]. However, they can need long training times, and the network may not always be well generalised.

### Radial Basis Function:

A radial basis function network consists of an input layer, a single hidden layer and an output layer. Radially symmetric basis functions are used to transform the inputs (the basis functions are usually Gaussian). These functions create local approximations by using the Euclidean distances between inputs, weights, and the Gaussian activation functions, which makes neurons more locally sensitive. RBF neurons have maximum activation when the weights are equal to the inputs. An advantage of using a RBFN is that it is easier to grow the number of neurons during training and they are good at approximation, meaning that in general they are much more robust than other networks. However, they usually require more neurons to achieve the same result for the same task than other networks – such as a multi-layer perceptron.

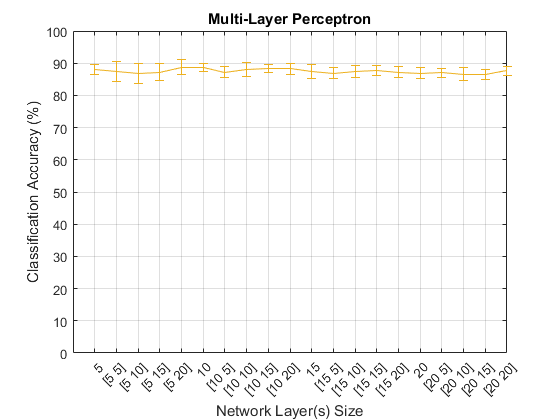
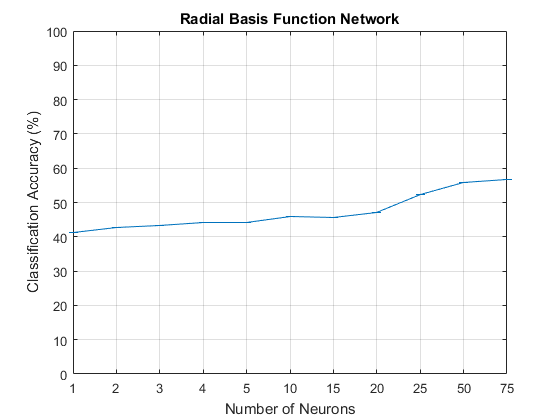
### Recurrent Neural Network:

Recurrent neural networks are those that have backward connections, which can be one or multiple, between neurons, so that a networks output or previous inputs can affect the current input, this acts as a form of memory (note that this memory is not infinite). Older memory is less usable as more inputs are used.

This means that time sequence problems can be solved. This means networks have more accurate predictions as the previous inputs and outputs are used. The problem with recurrent neural networks is that they can be difficult to train, and they have issues with convergence. One example is this is the exponential decay of the backpropagation signal. Furthermore, there is a smaller selection of learning algorithms, as these learning algorithms have to be able to handle time sequences.

The RNN I will be using is a nonlinear autoregressive exogenous (NARX) network. This means the network will use both past values of the output, and current and past values of the input.

Figure Graphs to assess the initial performance of the networks. Note the standard deviation on the RBF graph as close to zero so it cannot be seen well



### Initial Network Selection:

An initial test was performed to see how the MLP and RBF architectures performed on the data, with a basic number of neurons and hidden layers. Each network was ran 10 times and the mean plotted, with the error bars showing the standard deviation from this initial exploration can see that the MLP outperform the RBF significantly. Therefore, I will be only using MLP and RNN. This will be a test of whether the RNNs memory is necessary. It is also a test of speed, if the MLP performs the same or slightly worse than the RNN but operates or trains significantly faster, it may be superior to use the MLP.

## Data

The data is a matrix of 5456 rows of 24 columns of sensor data with a 25th column of class values in string format, the sensor data is all in double form. This was transposed to give 25 rows with the first 24 rows corresponding to a sensor, the 25th corresponding to a class values and 5456 columns of data entries.

As the meta-data accompanying the data set describes each sensor data is valid. The data was verified against the meta-data. For the first sensor “US 1” meta-data states, the maximum and minimum values are five and 0.4, and the mean and standard deviation is 1.47162 and 0.80280. This was calculated again after importing into MatLab and the meta-data was accurate.

The meta-data describes that four rounds took place but does not state if these rounds were of equal size or not. It will be assumed that the rounds are of equal size, 1364 time steps per round. E.g. round one will be the columns from 1:1364. All pre-processing was performed on the first two rounds, the training set, so that it did not affect the validation and test sets (see **Cross Ref** for more information about the division between training, validation and test sets).

The class value, the output, was initially a string therefore this data needed to be one hot encoded. Each class label was assigned a number from one to four. These integers were then one hot encoded. Each string was replaced by an integer and then each integer was replaced by a one hot encoded value.

|  |  |  |
| --- | --- | --- |
| String | Integer | One hot encoded |
| Move-Forward | 1 | 1000 |
| Slight-Right-Turn | 2 | 0100 |
| Sharp-Right-Turn | 3 | 0010 |
| Slight-Left-Turn | 4 | 0001 |

Figure One hot encoding

As a RNN was used, the matrix also needed to be turned into a time series. This involved turning the matrix into 5456 time steps with each time step being a vector of the twenty-four sensors. This was repeated for the class values - 5456 time steps with each time step being a vector of the one hot encoded values.

It was also neccesary to see if any of the input data was correlated. Four plots were to show the correlation between sensor variables, these show that no sensors are correlated. **REDO !!!**

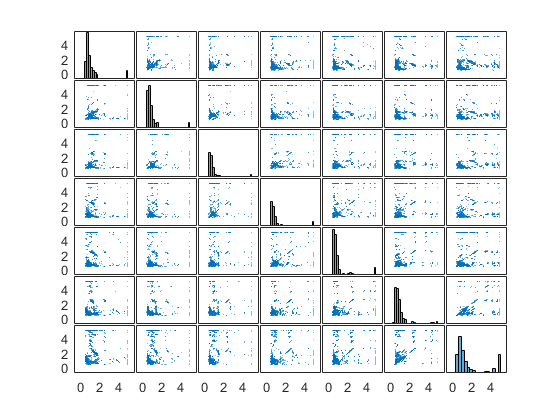
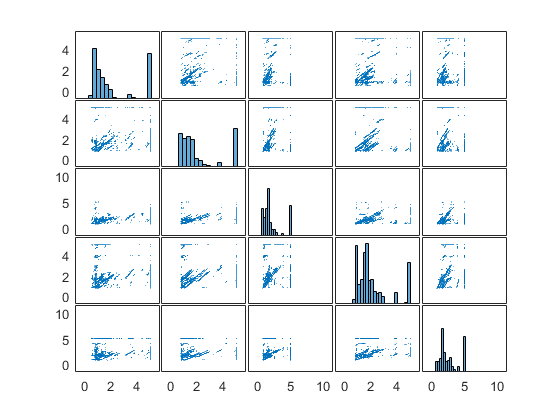
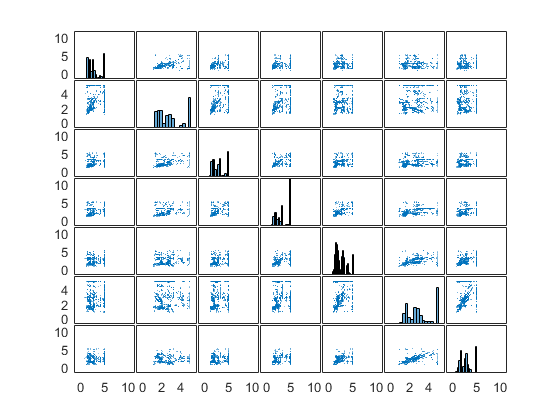
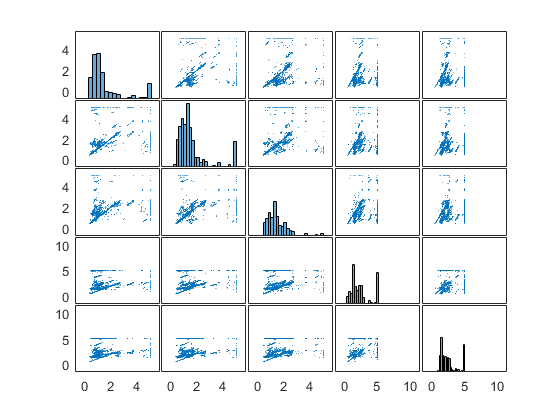


Figure 5 The plots are arranged from right to left by: the sensors to the front -US23, US24, US1, US2, US3, the sensors to the right - US4, US5, U6, U7, US8, US9, U10, the sensors to the back - US11, US12, US13, US14, US15, the sensors to the left - US16

The meta-data states that the number of samples for each class is not equal. For the entire data set *Move-Forward* has 2205 samples (40.41%), *Slight-Right-Turn* has 826 samples (15.13%), *Sharp-Right-Turn* has 2097 samples (38.43%) and *Slight-Left-Turn* has 328 samples (6.01%). To account for this the data for *Slight-Right-Turn* and *Slight-Left-Turn* was oversampled.

*Slight-Right-Turn* was oversampled by a rate of two and *Slight-Left-Turn* was oversampled by a rate of six. Sampling is performed using an antialiasing FIR low pass filter. Note that the method for oversampling preserves the order for the data. The method for oversampling divides the data into blocks of consistent class values, the blocks with class values of *Slight-Right-Turn* and *Slight-Left-Turn* are then oversampled, and then blocks are stitched back together.

*E.g. the sequence of class values [1,1,1,4,4,2,2,2,2,3,3,3,2,2,] would go to [1,1,1] 6\*[4,4]2\*[2,2,2,2][3,3,3]2\*[2,2] which would go to [1,1,1,4,4,4,4,4,4,4,4,4,4,4,4,2,2,2,2,2,2,2,2,3,3,3,2,2,2,2].*

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Original | Oversampled training set | Oversampled training set combined with validation and test |
| 1 | 2205 | 1070 | 2205 |
| 2 | 826 | 922 | 1272 |
| 3 | 2097 | 999 | 2097 |
| 4 | 328 | 1038 | 1183 |
| Total | 5456 | 4029 | 6757 |

Figure Amount of data after oversampling

The matrix is then processed by mapping its minimum and maximum values to one and negative one, and finally, constant rows are removed. Note one way of transforming the input data for use with the MLP would be for each input, also input *x* number of previous inputs. However, this would effectively perform similar to the RNN so will not be done.

The flow chart below shows the steps to get the data used in the following sections.

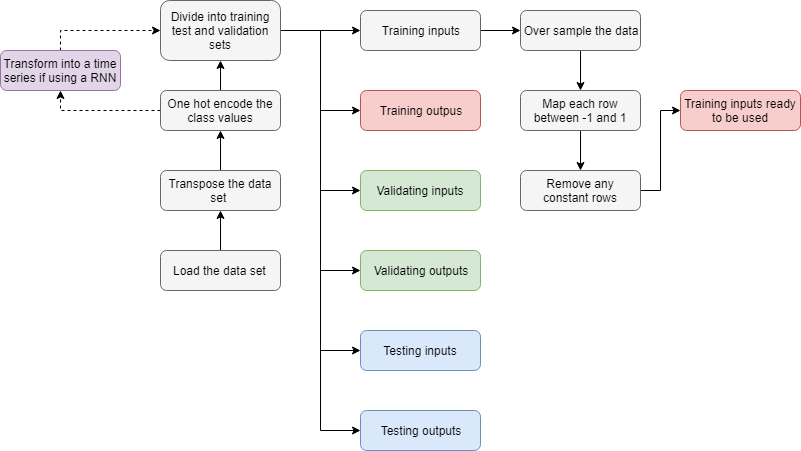


Figure Operations performed to go from the dataset to network inputs and outputs

## Training:

I decided between two training algorithms *trainlm* [ADD REF] and *trainscg* [ADD REF] to use in the network. The algorithm *trainlm* uses *Levenberg-Marquardt backpropagation* while *trainscg* uses scaled conjugate gradient backpropagation. To decide between the two I ran an initial test. Using four networks: a MLP with *trainlm*, a MLP with *trainscg,* a RNN with *trainlm* and a RNN with *trainscg.* Each network was tested with up to two hidden layers with neuron size per layer varying from five to twenty in steps of five (e.g. [5 10] means there were five neurons in the first hidden layer and ten in the second). The group of plots shows the effect of the training algorithm against the classification accuracy.

From the two *“trainlm”* plots, it shows it performed better and more consistently. Therefore, *“trainlm”* will be the training algorithm used.

|  |  |
| --- | --- |
| Parameter | Value |
| Maximum number of epochs to train | 1000 |
| Performance goal | 0 |
| Maximum validation failures | 6 |
| Minimum performance gradient | 1e-7 |
| Initial mu | 0.001 |
| Mu increase factor | 10 |
| Maximum mu | 1e10 |
| Maximum time to train in seconds | Infinite |

Figure Training parameters; note these are the Matlab defaults for trainlm [ADD REF]

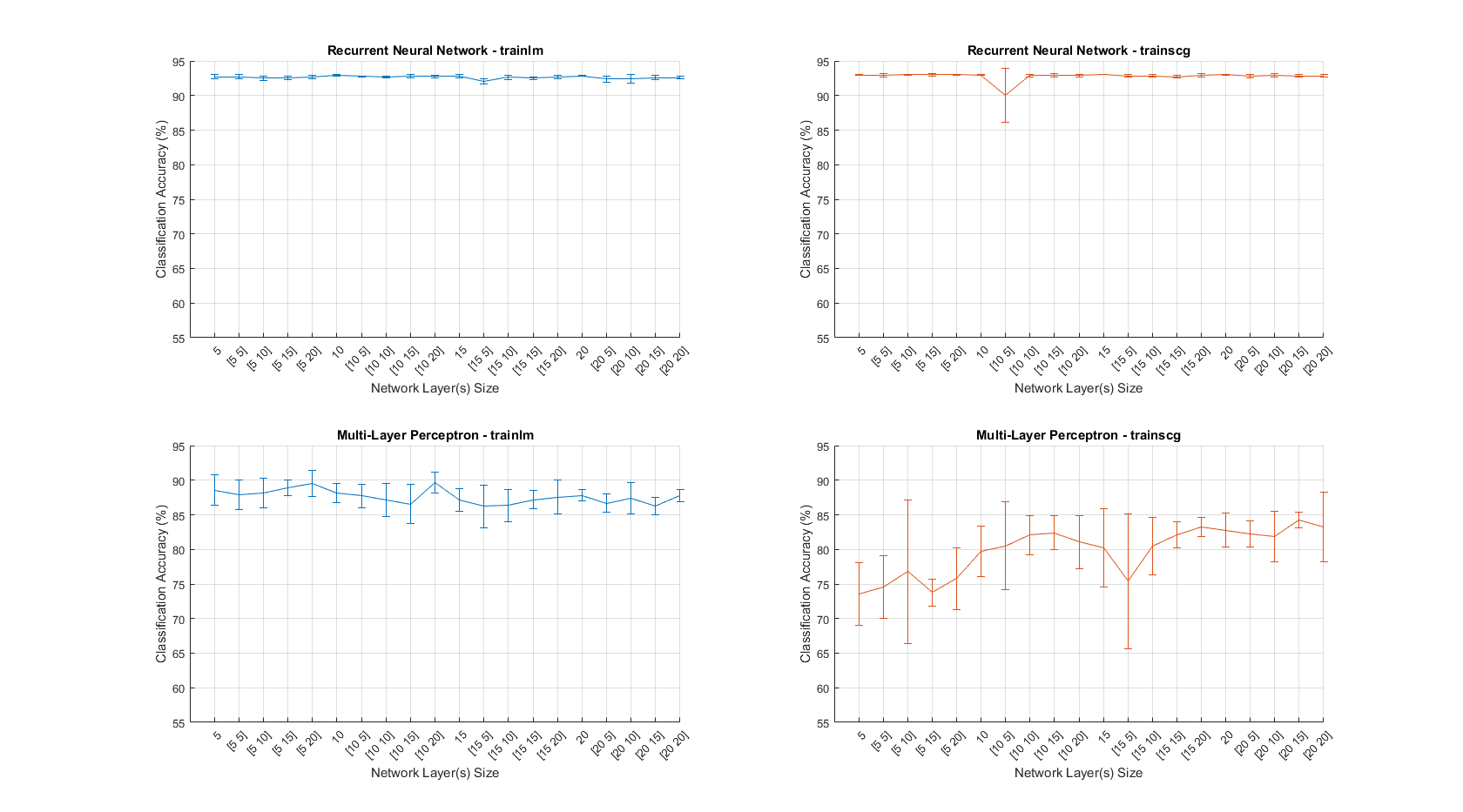
However, the downside of using *trainlm* over *trainscg* was that *trainlm* did take longer to run than *trainscg*. For example, the time taken to train the graph *Multi-layer Perceptron –* *trainlm* was 3.2988 secondsand the time taken to train *Multi-Layer Perceptron – trainscg* was 0.3192 seconds**.**

Figure Graphs to demonstrate best training algorithm

## Evaluation:

As discussed above the data was split into training, test and validation sets. The training set contained the columns 1:4029 - this was the first two rounds oversampled, the validation set contained columns 4030:5393 -the third round, and the test set contained 5394:6757-the fourth round**.** The data was divided into rounds as this problem involves a time series, so randomly dividing the data between the sets would invalidate the series.

To compare networks the classification accuracy was used. This was the percentage of correctly identified classes, when comparing the output classes from the neural network with the actual classes from the data. The classification accuracy was only worked out from the testset. To ensure this classification accuracy was accurate, each network was trained, validated and tested 10 times, and then the mean classification accuracy was calculated as well as the standard deviation for each network. A criterion for a good network was a high mean classification accuracy with a low standard deviation.

A different number of hidden layers and different number of neurons in these hidden layers was then tested. From [ADD CROSS REFERENCE] it can be seen that the recurrent neural network performed better than the multi-layer perceptron across the board, it was much more accurate and the variation in the accuracy of the networks was smaller. The best performing network had one hidden layer with 15 neurons with a mean accuracy of 92.93902 and standard deviation of 0.088.

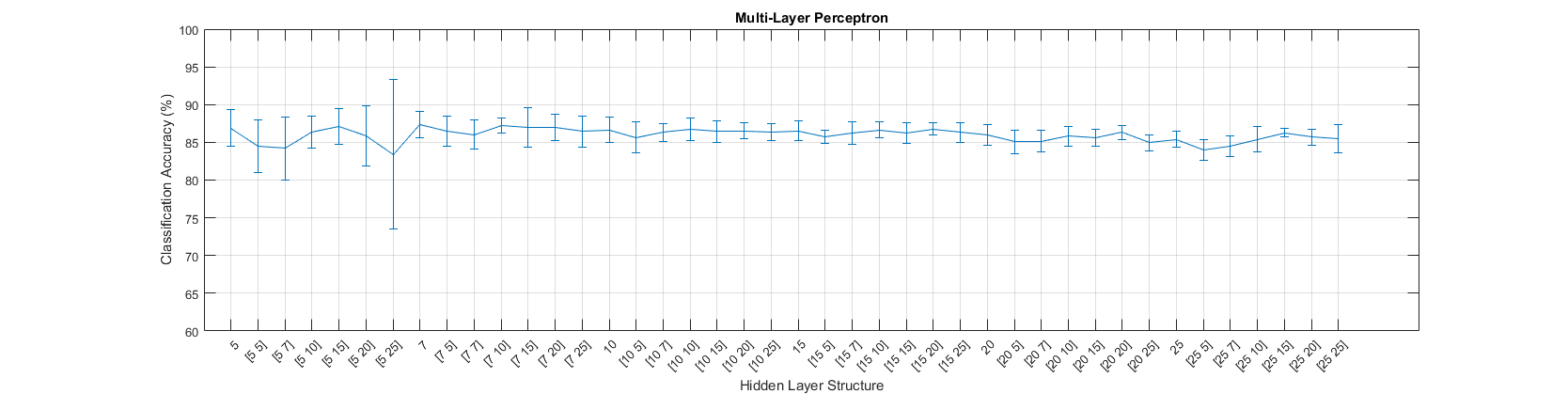
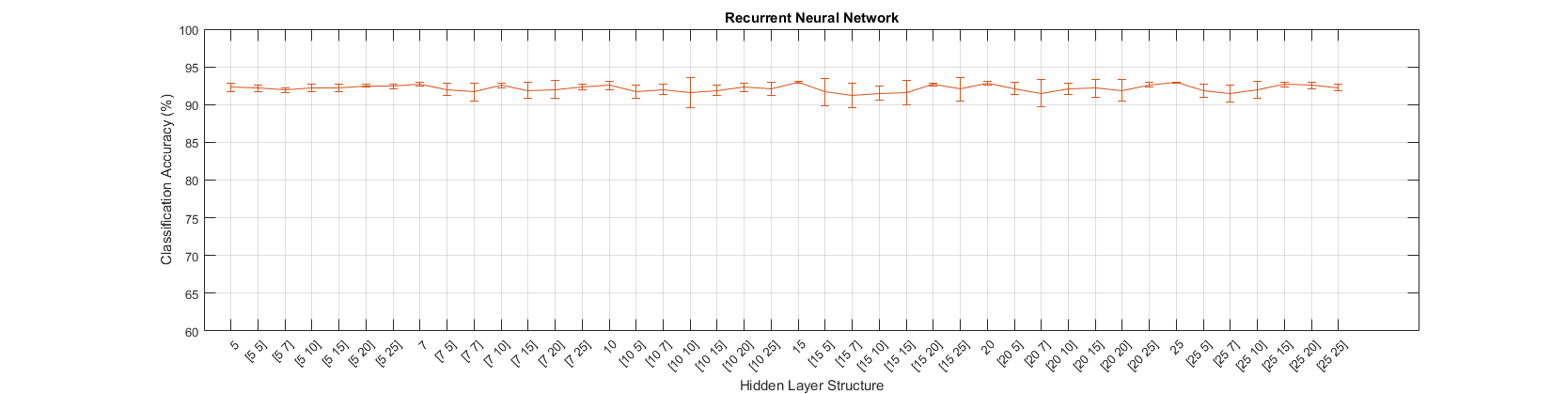


Figure Classification accuracy for different networks with different network structures, for a MLP architecture and RNN architecture.

## Network:

The best network was a NARX network with one hidden layer with twenty-five neurons in the hidden layer

ALL STRUCTURAL INFO, PARAM

HOW IMPLEMENTED (MATLAB DEFAULTS)

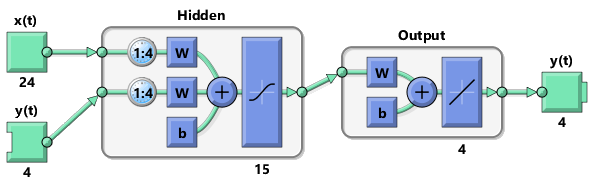


Figure NN

## Results:

This network had an average accuracy of 92.9%. In addition, as can be seen from the confusion plot and ROC **WHAT** **IS ROC** [ADD CROSS REF] the neural network performs well on all classes, including classes 2 and 4. The network did not just classify all data into classes 1 and 3, as these were the largest classes, but correctly classified classes 2 and 4 with a high accuracy as well; the most likely reason for this is from oversampling the training data. There were little false positives. The network can classify to a greater than 90% accuracy each class so the network should be able to steer the robot around the room correctly.

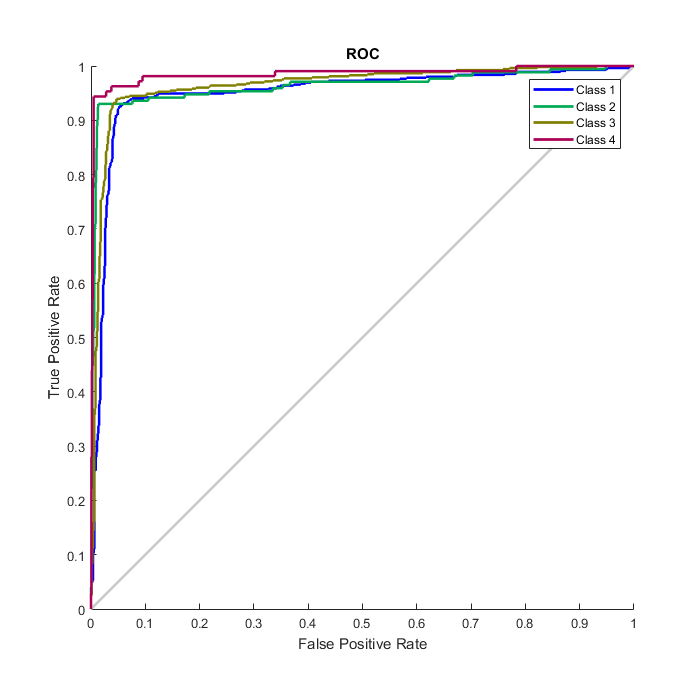
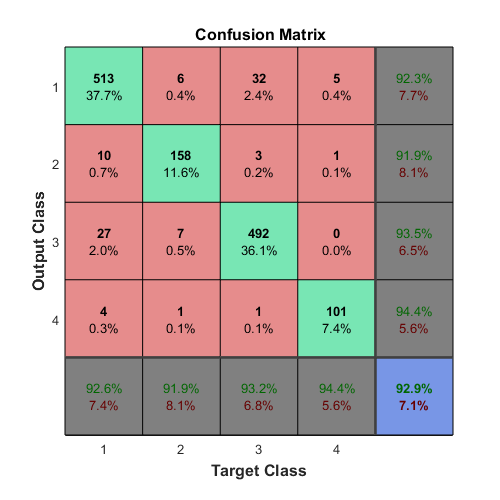


Figure CONFUSION AND ROC

One particular point of interest was the effect of varying the previous number of inputs and outputs. This had a great effect on performance. When training with using up to a single previous input and output the network was trained with a time off **X**, when trained with up to five previous inputs the time to train was **Y** and when used with ten the time to train was **Z.** In addition, it had little to no effect on accuracy as can be seen from [].

**PLOT OF FEEDBACK DELAY AGAINST ACCCURACY**

# Section B:

## Deep EndoVO: A recurrent convolutional neural network (RCNN) based visual odometry approach for endoscopic capsule robots:

This paper proposes a visual odometry; visual odometry is the process of determining the position and orientation of a robot by analysing camera images, system based on deep recurrent convolutional neural networks. This is for use with active, remote controllable, robotic capsule endoscope prototypes. These prototypes need real time pose estimation functionality and are used in hospitals for screening the gastrointestinal tract and diagnosing diseases.

### Main features of the architecture:

The paper presents a RCNN architecture, which can model sequential dependence and motion across video frames. The architecture uses inception modules for feature extraction and a RNN for sequential modelling of motion dynamics to regress the robot’s orientation and position in real time. By using an RNN, the pose estimation of the current frame benefits from information from previous frames. Therefore, the deep RNN consists of two LSTM layers with the output sequence of the first forming the input sequence of the second one each containing 1000 hidden neurons.

The proposed deep learning network consists of three inception layers and two LSTM layers concatenated sequentially. The inception layers are extracting multi-level features (with each inception layer extracting different size detail) the final inception layer passes the feature representation into the RNN modules.

### Training the network:

The system is trained on Euclidean loss using the Adam optimization method. **The backpropagation algorithm is used to calculate the gradients of RCNN weights, which are passed to the Adam optimization method to compute adaptive learning rates for each parameter employing the first-order gradient-based optimization of the stochastic objective function.**

The architecture was trained using Caffe library and NVIDIA Tesla K40 GPU. Using back-propagation-through-time method, the weights of hidden neurons were trained for up to 200 epochs with an initial learning rate of 0.001. Overfitting was prevented using dropout and early stopping techniques

### Data:

There were two training datasets. The first dataset was recorded on five different real pig stomachs. To ensure that the algorithm was not attuned to a specific camera model, four different cameras were employed. For each pig stomach camera combination, 2000 frames were acquired which gave 40000 frames, in total.

As a second training dataset, for each of four cameras 10000 frames on a human stomach simulator were captured, giving40000 frames. During video recording tracking software was utilized to obtain **6-DoF** localization ground truth data. This was used as a gold standard for the evaluations of the pose estimation accuracy.

A testing dataset was create by recording five different real pig stomachs, which were not used for the training section. For each pig stomach camera combination, 2000 frames were acquired giving 40000 frames. There was no synthetic dataset for the testing session since it was less realistic due to obvious patterns of artificial simulators. For all of the video records the 6-DoF localization ground truths were recorded again.

### Effectiveness:

The method solves several issues faced by typical visual odometry pipelines. Neither prior knowledge nor parameter tuning is needed to recover the absolute trajectory scale contrary to monocular traditional VO approach. Contrary to a traditional VO pipeline, the deep learning-based VO did not require any explicit feature extraction, matching, outlier detection or multi-scale bundle adjustment-like parameter tuning requiring operations.

The performance of the simEndoVO and realEndoVO were analysed using averaged root mean square errors (RMSEs). Testing was performed on both simEndoVO and realEndoVO. Making a comparison with GoogLeNet and ResNet50 architectures. These were modified to regress 6-DoF pose values by removing a softmax layer and integrating a fully- connected layer and an affine regressor layer.

The results indicated that realEndoVO clearly outperforms GoogLeNet and ResNet50, whereas simEndoVO slightly outperforms them. realEndoVO is able to stay close to the ground truth pose values for even sharp motions, GoogLeNet and ResNet50 path estimations deviate drastically from the ground truth path values. Even for very fast and challenge paths the deviations of realEndoVO from the ground truth remain in an acceptable range for medical operations.

**DEEP EndoVO, simEndoVO and realEndoVo, SLAM**

The performance of the proposed deep EndoVO with two of the widely used state-of-the-art SLAM methods LSD and ORB SLAM. In addition to higher accuracy and robustness particularly in environments with little key points, deep EndoVO architectures makes an optimal use of both direct and feature point information to estimate the pose. Both simEndoVO and realEndoVO clearly outperform LSD SLAM and ORB SLAM in terms of pose accuracy. EndoVO is much more robust and reliable compared to LSD SLAM and ORB SLAM. In many parts of the trajectories, ORB SLAM and LSD SLAM deviate from the ground truth trajectory drastically, whereas deep EndoVO is still able to stay close to the ground truth values even for most challenging trajectory sections

# Robotic Grasp Detection using Deep Convolutional Neural Networks

The problem of robotic grasping is unsolved, one interpretation of the task is to map pixel values from an RGB-D image to real world coordinates, and from these coordinates calculate a position and orientation for a robots end of arm tooling (EOAT) to give a grasp configuration.

### Main features of the architecture:

The approach uses two 50-layer deep convolutional residual neural networks running in parallel to extract features from RGB-D images, with one network analysing the RGB component and the other analysing the depth channel. The outputs of these networks are then merged, and fed into another convolutional network that predicts the grasp configuration.

**ResNet-50, a fifty layer deep residual model, is used to solve this grasp detection problem– DESCRIBE RESNET PLACE IN THE WORLD**

Two different architectures are used for the robotic grasp prediction, a uni-modal grasp predictor and a multi-modal grasp predictor. The uni-modal grasp predictor is a 2D grasp predictor that uses only single modality (e.g., RGB) information from the input image to predict the grasp configuration, whereas the multi-modal grasp predictor is a 3-D Grasp Predictor that uses multi-modal (e.g., RGB and Depth) information.

For a baseline model, a linear SVM is used as a classifier to predict the grasp configuration for the object using the features extracted from the last hidden layer of ResNet-50.

In the uni-modal grasp predictor, the last fully connected layer of ResNet-50 is replaced by two fully connected layers with rectified linear unit (ReLU) as activation functions. A dropout layer is also added after the first fully connected layer to reduce over-fitting. SGD is used to optimize our training loss and mean squared error (MSE). The 3-channel image is fed to the uni-modal grasp predictor, which uses the residual convolutional layers to extract features from the input image. The last fully connected layer is the output layer, which predicts the grasp configuration for the object in the image.

A ResNet-50 model that is pretrained on ImageNet is used to extract features from the RGB channels of the image. Similar to the unimodal grasp predictor, features are extracted from the second last layer of both the ResNet-50 networks. The extracted features are then normalized using L2-normalization. The normalized features are concatenated together and are fed into a shallow convolutional neural network with three fully connected layers. The fully connected layers use ReLU activation functions. A dropout layer is added after the first and second fully connected layers of the shallow network to reduce over-fitting. Similar to the uni-modal model, we used SGD as the optimizer and MSE as the loss function.

### Training the network:

The training process was divided into two stages, in the first stage, only the shallow network is trained, and in the second stage the complete network is trained end-to-end. To train the uni-modal grasp predictor, SGD was used to optimize the model with hyper parameters in first stage. For fine-tuning the network in the second phase, a much lower learning rate was used and the learning rate was plateaued if the training loss did not decrease.

Large-scale image classification datasets only have RGB images. Therefore, we can pre-train our deep convolutional neural networks with only 3-channels. During training time, weights of convolutional layers in ResNet-50 are kept fixed and only the weights of last two fully connected layers are tuned. The weights of the last two layers are initialized using Xavier weight initialization.

By using two DCNNs in parallel, the model was able to extract features from both RGB and depth images. Therefore, enabling the model to learn multimodal features from the RGB-D dataset. Weights of the two DCNNs are initialized using the pre-trained ResNet-50 models and the weights of the shallow network are initialized using Xavier weight initialization. The weights are fine-tuned during training. As a simple baseline, we also applied a linear SVM classifier to the L2-normalized RGB DCNN and depth DCNN features to predict the grasp configuration for the object in the image.

### How much data:

For comparing their method with others, they tested the architecture on the standard Cornell Grasp Dataset. This dataset consists of 885 images of 240 different objects. Each image has multiple grasp rectangles labelled as successful (positive) or failed (negative), specifically selected for parallel plate grippers. In total, there are 8019-labelled grasps with 5110 positive and 2909 negative grasps.

The data is split two ways. The first image-wise splitting splits all the images in the dataset randomly into the five folds. This is helpful to test how well did the network generalize to the objects it has seen before in a different position and orientation. Secondly object-wise splitting splits all the object instances randomly and all images of an object are put into one validation set. This is helpful to test how well did the network generalize to objects it has not seen before.

### Pre-processing:

The input to the DCNN is a patch around the grasp point extracted from a training image the patch is first re-sized to 224\*224, which is the input image size of the ResNet-50 model. The depth image is rescaled to range 0 to 255. There are some pixels in depth images that have a NaN value as they were occluded in the original stereo image. Zeros replaced these pixels with NaN value.

As the domain-specific data is limited as in the Cornell grasp dataset pre-training was necessary. Therefore, ResNet-50 is first trained on ImageNet. It was assumed that most of the filters learned are not specific to the ImageNet dataset and only the layers near the top exhibit specificity for classifying 1000 categories. The DCNN will learn universal visual features by learning millions of parameters during this pre-training process. We then grab the features from the last layer and feed it to our shallow convolutional neural network. It is important to note that the ImageNet dataset has only RGB images and thus the DCNN will learn RGB features only.

### Effectiveness:

They demonstrate that deep convolutional neural networks can be used to predict the grasp ability and for an object. The proposed architecture performs better than current state of the art systems, in both accuracy and speed. To take an example the uni-modal grasp predictor ran 800 times faster than the two-stage SAE model by Lenz [ADD REF]. Accuracy wise they were able to achieve an accuracy of 93.4%, which is at par with the current state-of-the-art.

## Bibliography:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |