INCA Summative

# Section A:

## Architectures:

The problem is to guide a mobile robot around a room in a wall following navigation task, note the assumption is that is to create a network for *this* room, not all rooms. 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 an additional column of one of four classes.

The problem is to input a set of sensor data and output a class therefore 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 indicating this is likely a non-linearly separable classification problem.

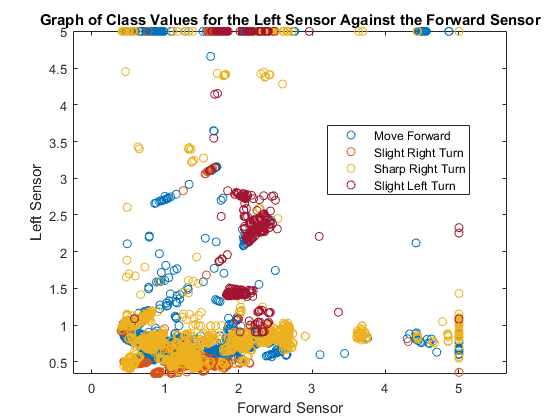


Figure 1: A graph to explore the data

I will investigate the problem, using a multi-layer perceptron (referred to as a MLP from here on), 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. They 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 [1]. However, they can need long training times, and the network may not always be generalised.

### Radial Basis Function:

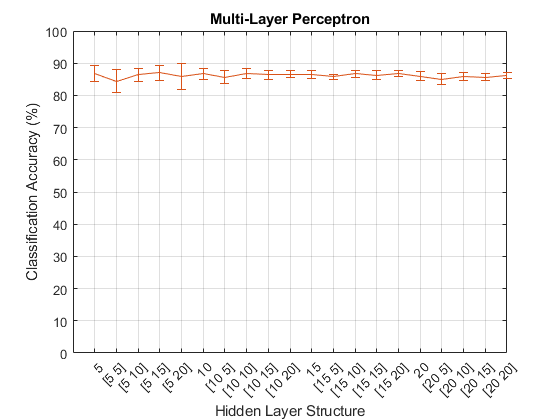
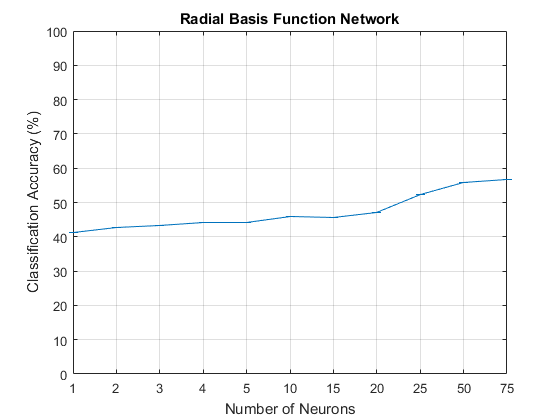
A radial basis function network consists of an input layer, a single hidden layer and an output layer. Radially symmetric basis functions are used to transform the inputs (the basis functions are usually Gaussian). These functions create local approximations by using the Euclidean distances between inputs, weights, and the Gaussian activation functions, which makes neurons more locally sensitive. RBFN 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 than other networks.

### Recurrent Neural Network:

Recurrent neural networks are those that have backward connections, that a networks previous outputs, previous inputs or previous targets can affect the current output. The RNN I will be using is a nonlinear autoregressive exogenous (NARX) network. There are two types of NARX parallel and series-parallel. With series parallel making a single step ahead prediction and parallel making multiple steps ahead. A series parallel architecture will be used. This means the network will use both past values of the target, and current and past values of the input. 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 must be able to handle time sequences.

### Initial Network Selection:

Figure 2: 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. A Hidden layer structure of [10 10] means there are 10 neurons in the first hidden layer and 10 in the second.



An initial test was performed to see how the MLP and RBF architectures performed, with a basic arrangement 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 the MLP outperform the RBF significantly. Therefore, only the MLP and RNN will be used.

## 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 and the sensor data all in double form. This was transposed to give a matrix of *25* rows and *5456* columns.

The meta-data states that the data is all valid, this was verified. For the first sensor “US 1” meta-data states, the maximum and minimum values are *5* and *0.4*, and the mean and standard deviation are *1.47162* and *0.80280*. This was calculated after importing into MATLAB and the meta-data was accurate. The data was also searched for missing values of which there was none.

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 an equal size of 1364 time steps. All pre-processing was performed on the first two rounds, the training set, so that it did not affect the validation and test sets.

The class value, the target, was initially a string, therefore this data needed to be one hot encoded. Each class label was assigned a number from 1 to 4. These integers were then one hot encoded.

|  |  |  |
| --- | --- | --- |
| 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 3 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 *24* 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. It was difficult to analyse whether the sensor data was correlated. Therefore, there is no need to remove any sensors as the networks perform well, with a high accuracy and a small training time.

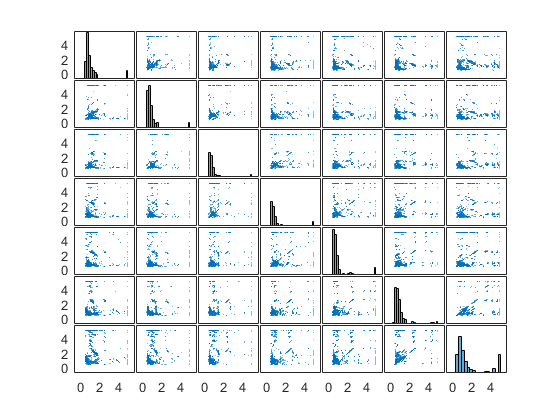
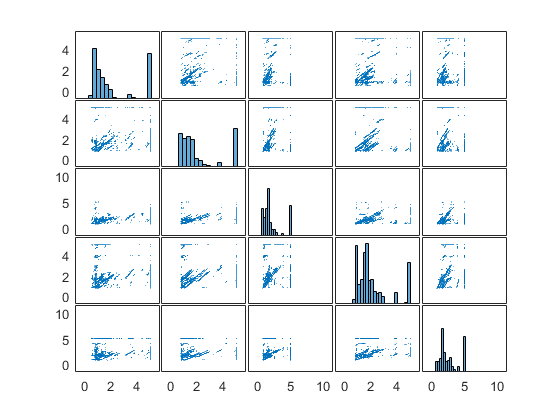
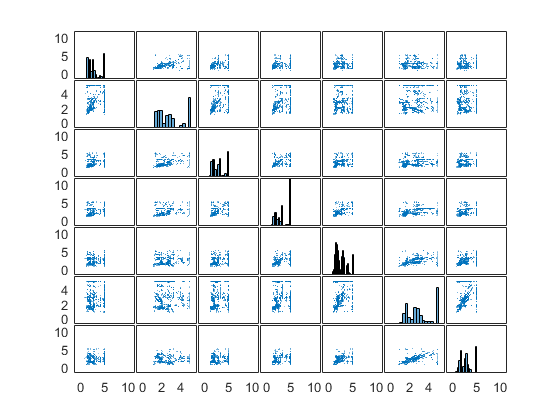
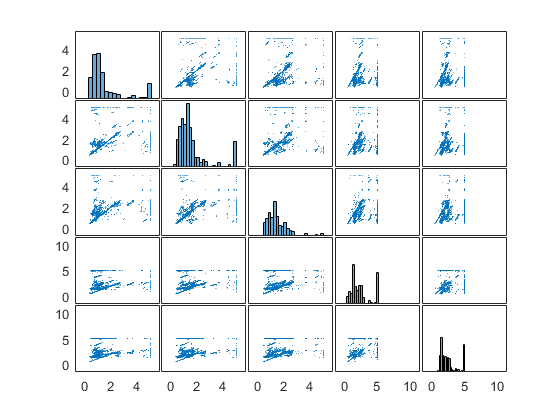


Figure 5: The plots are arranged from right to left by: the sensors to the front -US23, US24, US1, US2, US3, the sensors to the right - US4, US5, U6, U7, US8, US9, U10, the sensors to the back - US11, US12, US13, US14, US15, the sensors to the left - US16

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 *Slight-Right-Turn* has *826* samples *Sharp-Right-Turn* has *2097* samples and *Slight-Left-Turn* has *328* samples. To account for this, the data for *Slight-Right-Turn* and *Slight-Left-Turn* was resampled. *Slight-Right-Turn* was resampled by a rate of two and *Slight-Left-Turn* was oversampled by a rate of six. Resampling is performed using an antialiasing FIR low pass filter [2]. Note that the method for resampling preserves the order of the data. The method for resampling 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 4: Amount of data after oversampling

The matrix is then processed by mapping its minimum and maximum values to *1* and *-1* [3], and, constant rows are removed [4] . 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 similarly to the RNN so was not done.

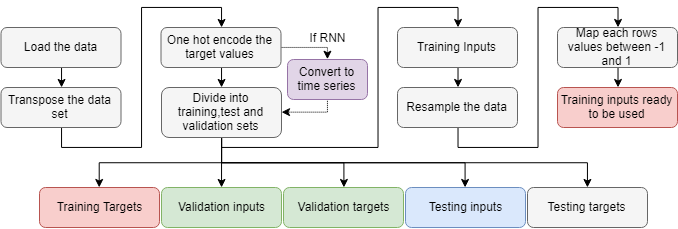


Figure 5: Steps needed to convert the data, so it is usable by the networks

## Training:

I decided between two training algorithms *trainlm* [5] and *trainscg* [6] to use in the network. The algorithm *trainlm* uses *Levenberg-Marquardt backpropagation* while *trainscg* uses scaled conjugate gradient backpropagation. To decide between the two an initial test was ran. 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 *2* hidden layers with neuron size per layer varying from *5* to *20* in steps of *5*. **Error! Reference source not found.** shows the effect of the training algorithm against the classification accuracy. *trainlm* performed better and more consistently. Therefore, *trainlm* was the training algorithm used.

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.

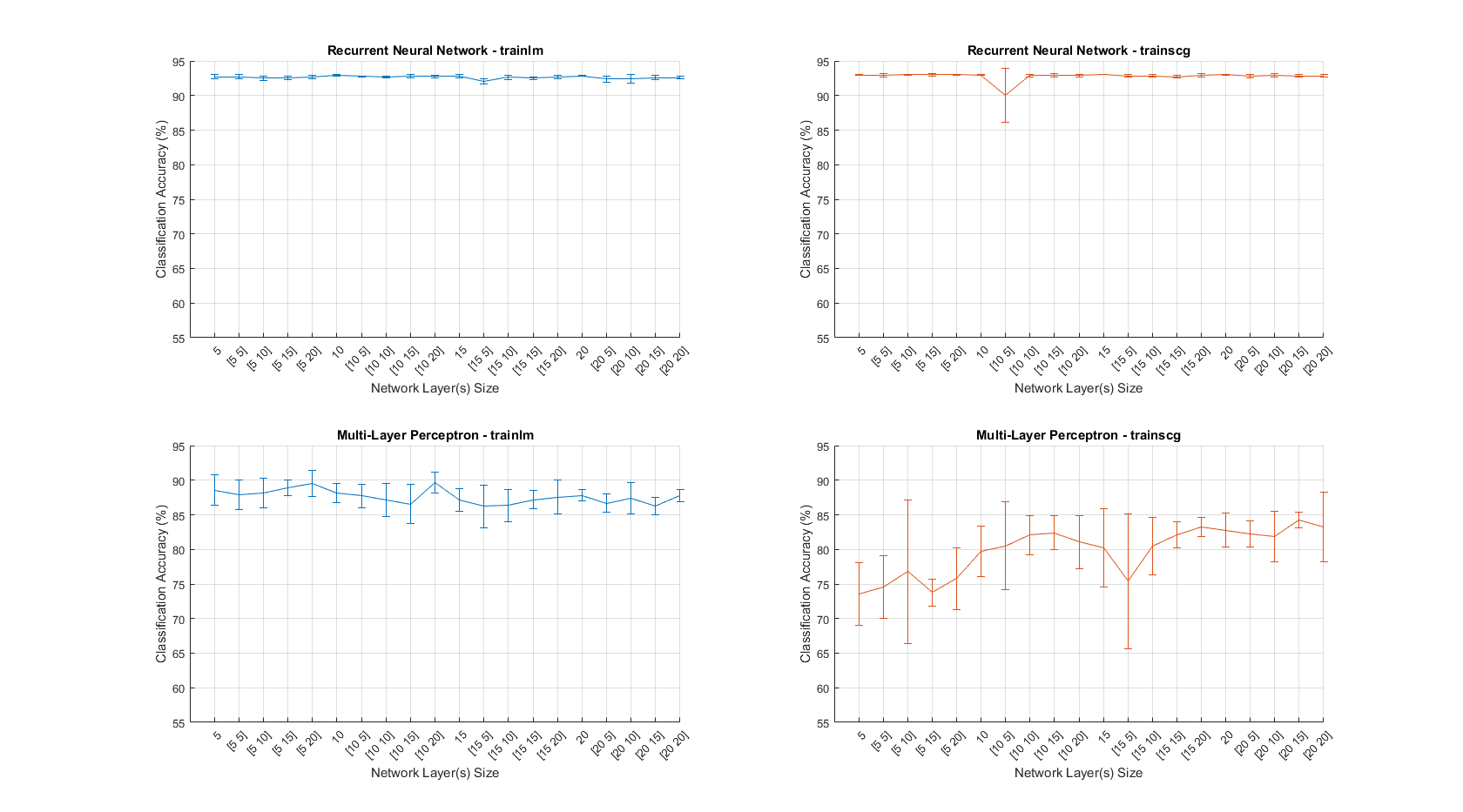


Figure 6: Performance of different training algorithms

|  |  |
| --- | --- |
| 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 7 Training parameters; note these are the Matlab defaults for trainlm [5]

## Evaluation:

The data was split into training, test and validation sets. The training set contained the columns *1:4029* – 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 involved a time series, so randomly dividing the data between the sets would invalidate the series.

To compare networks classification accuracy was used. This was the percentage of correctly identified classes, when comparing the output classes from the network with the target classes from the data. The classification accuracy was only calculated 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.

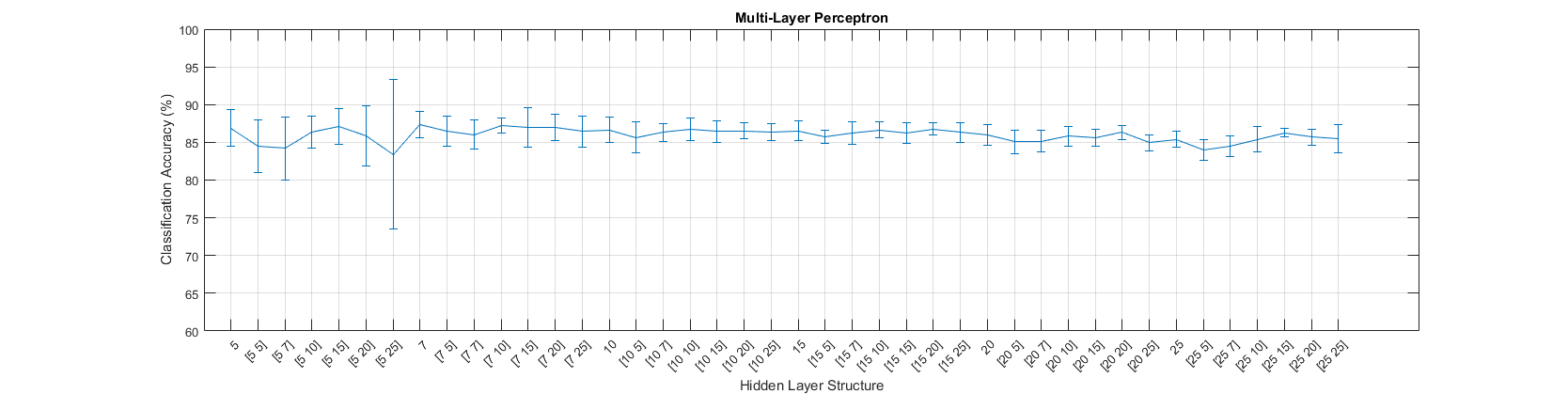
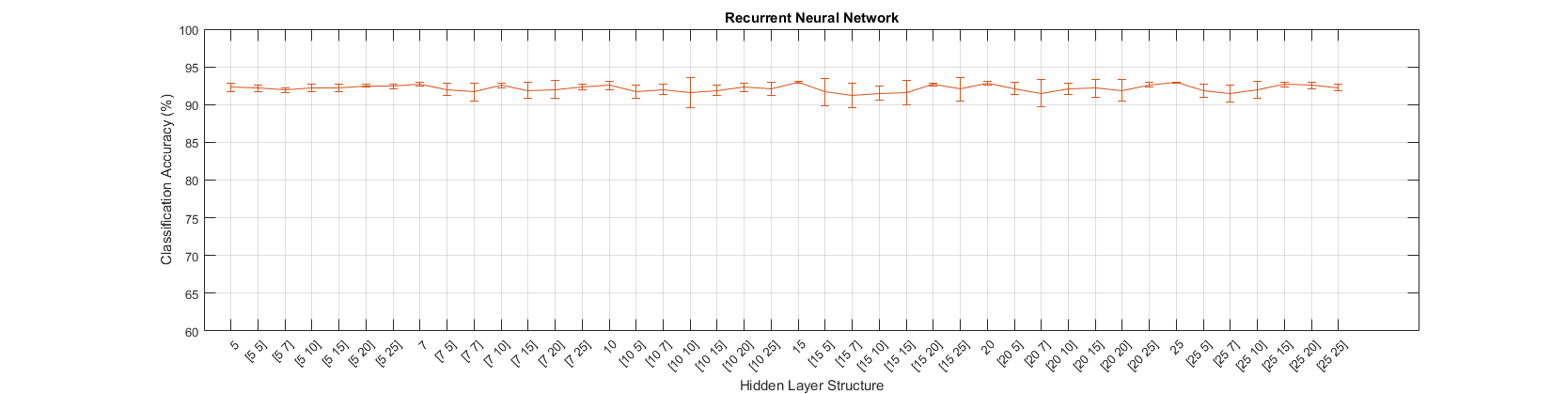


Figure 8 Classification accuracy for different networks with different network structures, for a MLP architecture and RNN architecture. A hidden layer structure of [10 10] means there are 10 neurons in the first hidden layer and 10 in the second.

A different number of hidden layers and different number of neurons in these hidden layers was then tested. From Figure 8 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 *1* hidden layer with *15* neurons with a mean accuracy of *92.93902* and standard deviation of *0.088.*

## Network:

The best network was a type of RNN, the nonlinear autoregressive exogenous network with series parallel architecture. The network takes two inputs, a timestep of the input and a timestep of the target. The hidden layer has fifteen neurons and a single layer, it has a hyperbolic tangent sigmoid transfer function[7]. It has an input and feedback delay of [1:4] meaning the previous 1 to 4 inputs and targets are used. The output layer has a linear transfer function [8] and 4 neurons. There is a single output a class value encoded in one hot form.

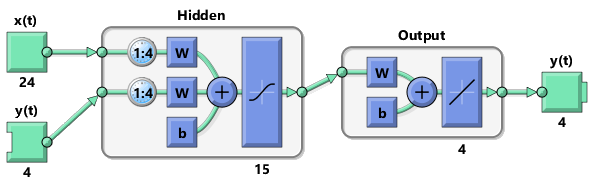


Figure 9 Diagram of the best performing neural network

The network was implemented using the MATLAB narxnet command [9]. The simplified code to create and train this is show in Figure 10. The important lines are *5* through *7*. Line *5* initialises the NARX with the specified parameters. Line 6shifts the input and target time series as many steps as are needed to fill the initial input and layer delay states [10]. Finally, line 7 trains the network and records the training session.

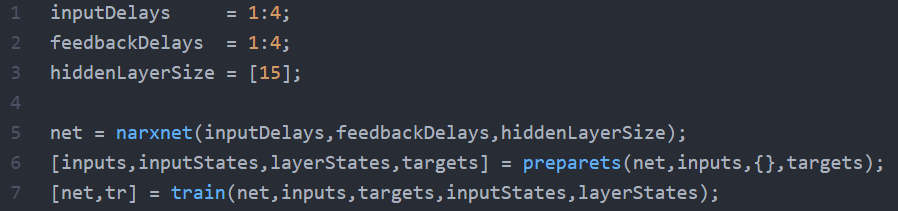


Figure 10: Code to create a simple NARX network

## Results:

This network had an average accuracy of *92.9%.* In addition, as can be seen from Figure 11 the neural network performs well on all classes, 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; potentially as the training data was resampled. 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.

However, there is the question of this networks generality. To relate this to the original problem imagine if in the original room, the furniture had moved. The MLP should be able to navigate this room relatively well as it works solely from the sensor values. The NARX network would not be able to be ran as the targets need to be known. For the NARX network to function for this changed room problem, the network would need to be converted to parallel architecture or *closed*. When testing this closed network on the original problem it scored a poor classification accuracy of *37%.*

One point of interest was the effect of varying the previous number of inputs and targets. This had a great effect on training time. When training with using a single previous input and target the network was trained with a time of *8 seconds*, when trained with up to *8* previous inputs and targets the time to train was *4 minutes 37 seconds*In addition, it had little to no effect on accuracy after increasing from *0* as can be seen from Figure 11.

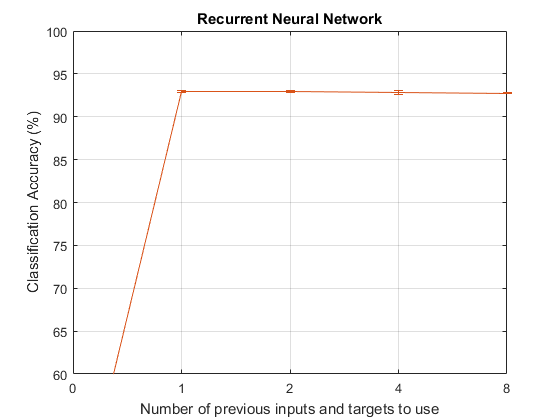
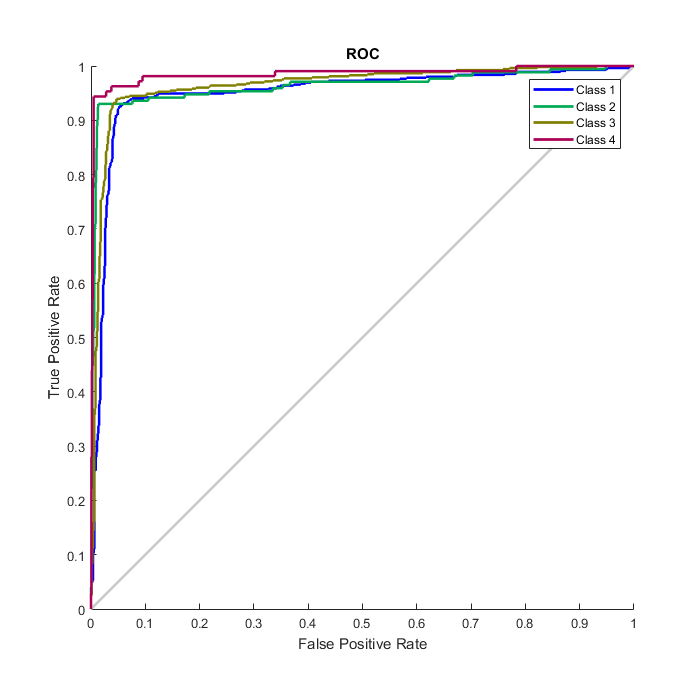
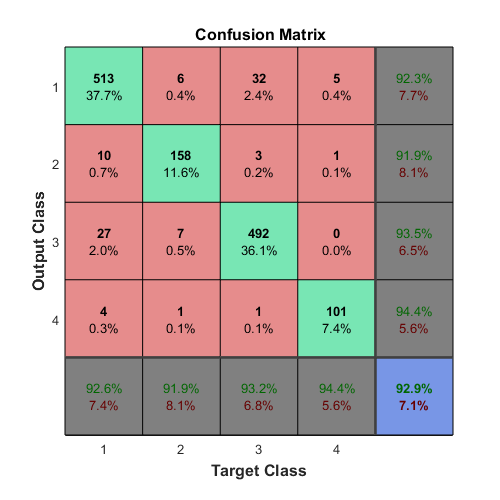


Figure 11: Performance graphs for the best performing network

# Section B:

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

This paper proposes a system based on deep recurrent convolutional neural networks for visual odometry. Where visual odometry is the process of determining the position and orientation of a robot by analysing camera images. 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 disease.

### 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. The network consists of three inception layers and two LSTM layers concatenated sequentially. The inception layers extract features with each inception layer extracting different sized detail and the final inception layer passes the feature representation into the RNN modules. 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. By using an RNN, the pose estimation of the current frame benefits from information from previous frames.

### 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 the 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 [**ADD REF**], using the 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 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 a total of *40000* frames. As a second training dataset, for each of the four cameras, *10000* frames off a human stomach simulator were captured, giving *40000* frames. During video recording ground truth data was captured, which 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 during training. For each pig-stomach camera combination, *2000* frames were acquired giving *40000* frames. There was no synthetic dataset used during testing. Again ground truths were recorded.

### Effectiveness:

The method solves several issues faced by typical visual odometry pipelines. Prior knowledge or parameter tuning is not needed to recover the absolute trajectory scale contrary to monocular traditional VO approach. In addition, deep learning-based VO did not require any explicit feature extraction, matching, outlier detection or multi-scale bundle adjustment.

The performance on the training datasets was analysed using averaged root mean square errors (RMSEs). and a comparison was made with the GoogLeNet [**ADD REF**] and ResNet50 architectures [**ADD REF**]. These were modified to regress pose values by removing a softmax layer and integrating a fully- connected layer and an affine regressor layer.

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

The performance of deep EndoVO is compared with two of the widely used state-of-the-art methods LSD and ORB SLAM. In addition to higher accuracy and robustness, the network 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 for the robotic arm.

### Main features of the architecture:

Large scale image datasets have only RGB images, therefore the network could only be trained on images with 3 channels. A unimodal grasp predictor is introduced which detects grasp using only 3-channels of the raw image. A ResNet-50 [**ADD REF**] pre-trained on ImageNet [**ADD REF**] is used to extract features from the image. As a baseline model a linear SVM is used as a classifier to predict the grasp configuration, using features from the last hidden layer of ResNet-50.Two fully connected layers replace the last fully connected layer of ResNet-50 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 the training loss and mean squared error (MSE) is used as the loss function. The last fully connected layer is the output layer, which predicts the grasp configuration for the object in the image. 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.

A multi-modal grasp predictor is introduced which uses multi-modal (RGB-D) information from the raw images to predict the grasp configuration. The raw RGBD images are converted into two images. The first is a RGB image and other is a depth image (converted into a 3-channel image). These two 3- channel images are then given as input to two independent pre-trained ResNet-50 models. The ResNet-50 layers work as feature extractors for both images. 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 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. SGD is used as the optimizer and MSE as the loss function. By using two DCNNs in parallel, the model was able to extract features from both RGB and depth images 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, a linear SVM classifier is applied to the L2-normalized RGB DCNN and depth DCNN features to predict the grasp configuration for the object in the image.

### Training the network:

The training process was divided into two stages. During the firsts stage only the shallow network is trained and during the second stage the complete network is trained end to end. For both unimodal and multimodal SGD is used to optimize the model in the first stage with learning rate set as *0.001 and 0.0006*. For fine-tuning the network in the second phase, use a much lower learning rate is used and the learning rate is plateaued if the training loss did not decreases.

### How much data:

For comparing their network, they tested the architecture on the standard Cornell Grasp Dataset [ADD REF]. This dataset consists of *885* images of *240* different objects. Each image has multiple grasp rectangles labelled as successful or failed. In total, there are *8019*-labelled grasps with *5110* successful and *2909* failed 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 the network generalized to the objects it had seen before in a different position and orientation. Secondly, object-wise splitting split all the object instances randomly and all images of an object are put into one validation set. This was helpful to test how well the network generalized 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 was first re-sized to *224\*224*, which is the input image size of the ResNet-50 model. The depth image is then rescaled to range *0* to *255*. There are some pixels in depth images that have a NaN value which were replaced with zeros.

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. Features from the last layer are fed to the shallow convolutional neural network.

### 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 unimodal 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.