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.

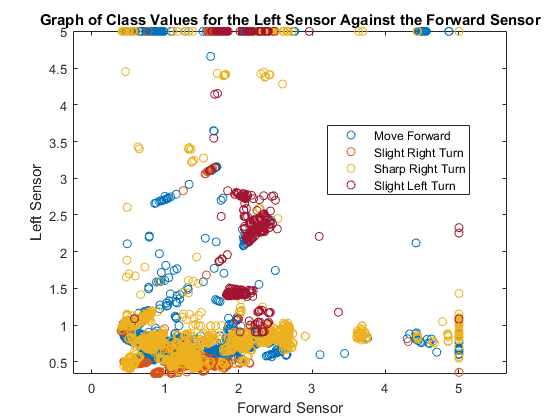


Figure 1 Initial graph to assess what type of problem attempting to solve

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 perform computation; they transform the input into something the output layer can understand, and the output layer – which makes a decision or prediction. The hidden layer(s) has 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 time, and the network may not always be general, for new and unseen data it may perform poorly.

### Radial Basis Function:

A radial basis function network consists of an input layer, a hidden layer and a single 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. Furthermore, 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 MLPs.

### Recurrent Neural Network:

Recurrent neural networks are those that have connections backwards (these connections can be just one or multiple) between neurons, so that a networks output can affect its input, this acts as a form of memory network, but this memory is not infinite. Older “memory” is less usable as more inputs are used. This means that time sequence problems can be solved, problems where previous information is useful (E.g. predicting the weather using the previous day’s weather). This means networks have more accurate predictions as the previous inputs and outputs are used and not ignored.

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 past values of the output, and current and past values of the input.

### Initial Network Selection:

An initial test was performed to see how each neural network 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 (the standard deviation is on the RBF graph but was close to zero so it cannot be seen). From this initial exploration can see that the RNN and MLP outperform the RBF significantly. Therefore, I will be only using MLP and RNN from here out. This will be a test of whether for this data it will be necessary for the RNNs “memory”or if just the input is adequate. It is also a test of speed, if the MLP performs the same or slightly worse than the RNN but operates faster, it may be better to use the MLP.

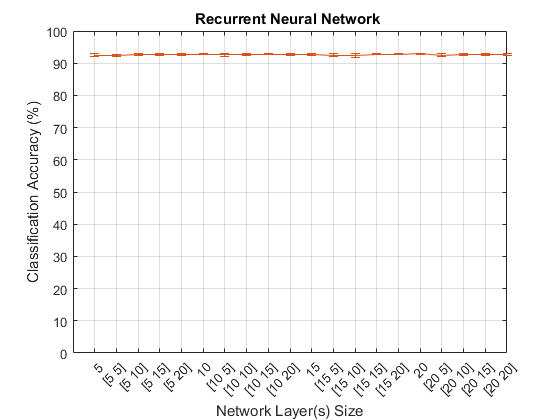
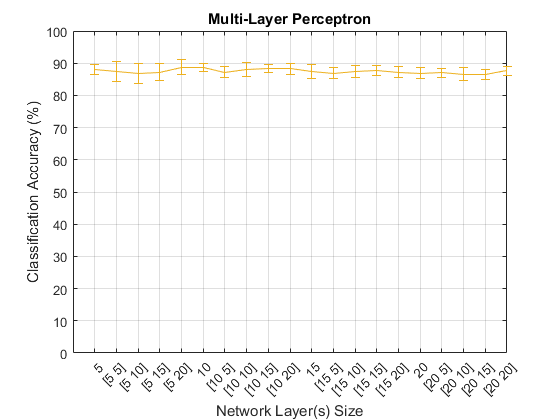
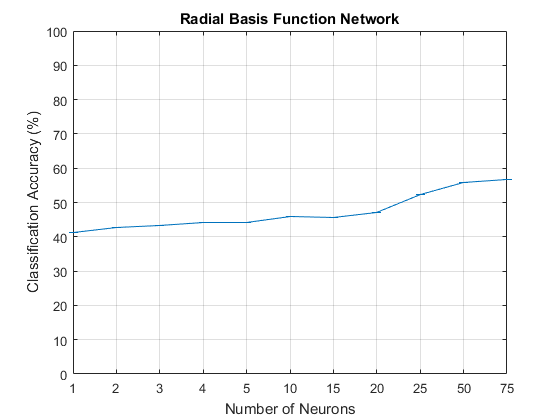


Figure 2 Graphs to assess the initial performance of the networks

## Data

The input data is 5456 rows of 24 columns of sensor data. The sensor data is all in double form. As the meta-data accompanying the data set describes each sensor data is valid. The data was then verified again according to the meta-data. To take an example 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. Therefore, it will be assumed that the rounds are of equal size, 1364 time steps per round. E.g. round one will be the rows 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 data.

The output data was initially a string – the class labels described above, 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. The table shows the conversion.

|  |  |  |
| --- | --- | --- |
| 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 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 created which are displayed below. From these it can be seen that no sensors are correlated

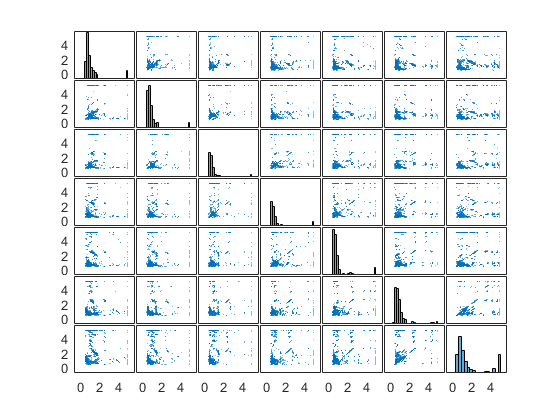
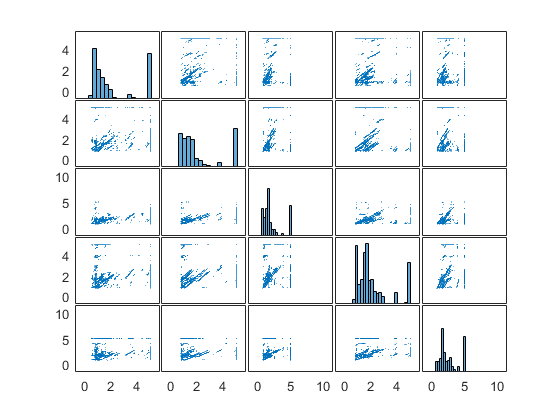
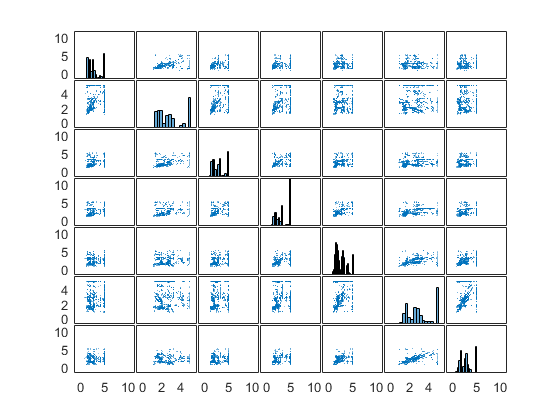
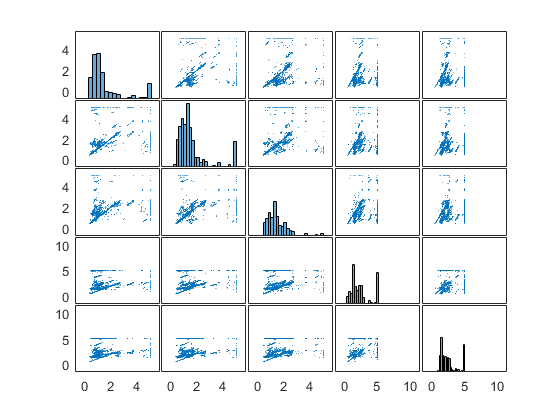


Figure 4 The plots are arranged from right to left by: the sensors to the front -US23, US24, US1, US2, US3, the sensors to the right - US4, US5, U6, U7, US8, US9, U10, the sensors to the back - US11, US12, US13, US14, US15, the sensors to the left - US16, US17, US18, US19, US20, US

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, those blocks with class values of *Slight-Right-Turn* and *Slight-Left-Turn* are then oversampled, and all blocks are then 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 | Training, validation and test | Test |
| 1 | 2205 |  |  |
| 2 | 826 |  |  |
| 3 | 2097 |  |  |
| 4 | 328 |  |  |
| Total | 5456 | 6803 |  |

Figure 5 Amount of data after oversampling

Principal component analysis (referred to as PCA from now on) was performed. First, each rows mean and standard deviation was mapped to zero and one respectively. PCA was then performed which produced a matrix with the rows in order in which they contribute to total variance. Rows were then removed if they contributed less than 0.02 to total variation. This matrix then contained sixteen rows with **X** columns. 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.

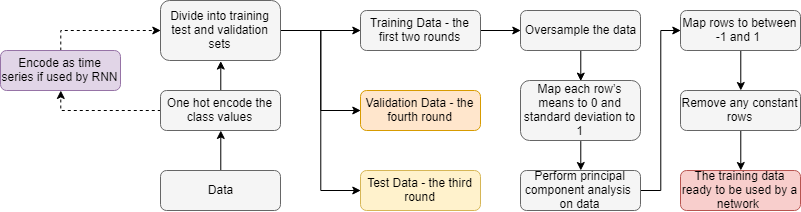


Figure 6 Operations performed on data

## 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 7 Training parameters

*Note: These are the Matlab defaults for trainlm [ADD REF]*

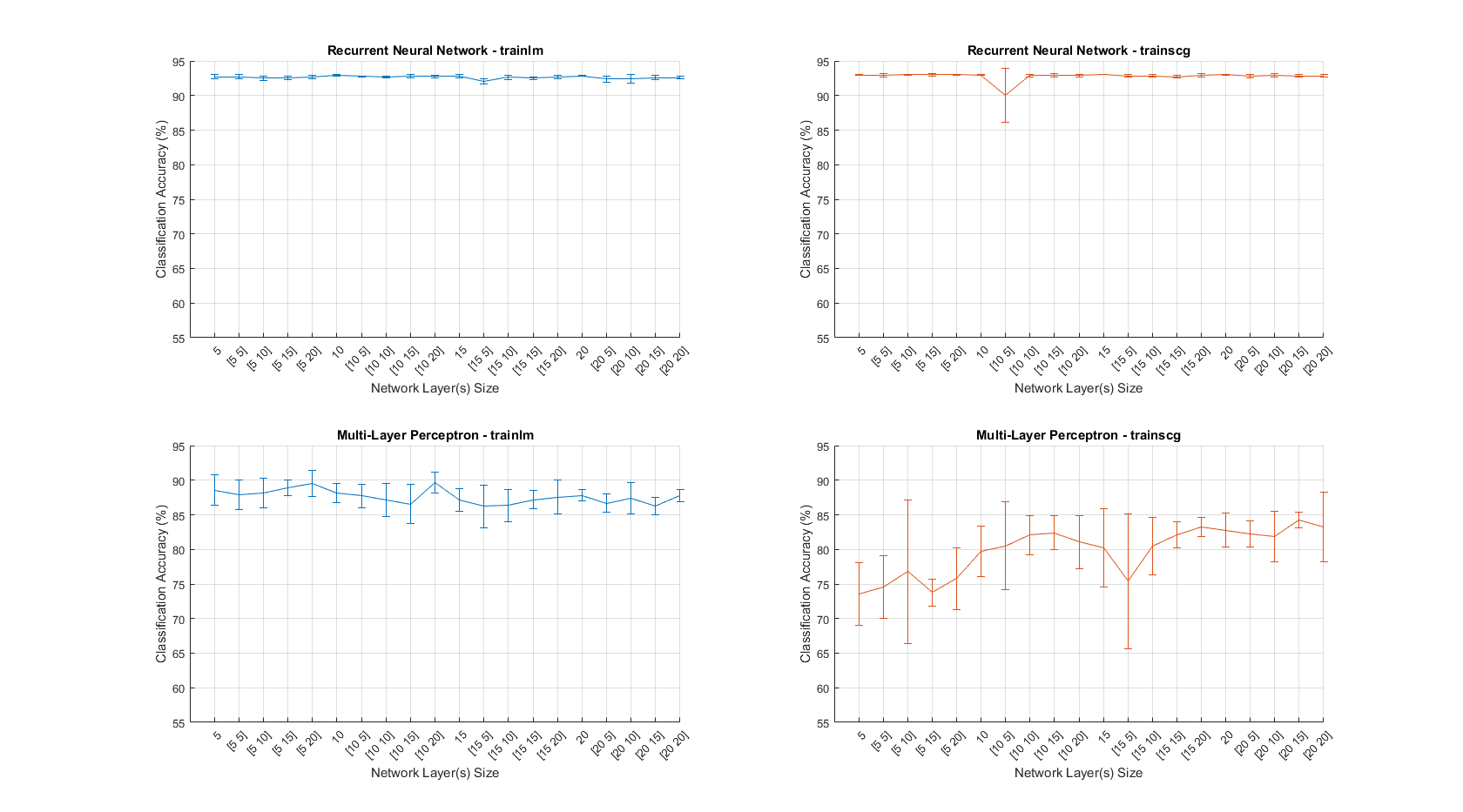
However, the downside of using *trainlm* over *trainscg* was that *trainlm* did take longer to run than *trainscg*. For example, the time taken to train the graph *Multi-layer Perceptron –* *trainlm* was 3.2988 secondsand the time taken to train *Multi-Layer Perceptron – trainscg* was 0.3192 seconds**.**

Figure 8 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 rows 1:4075 - this was the first two rounds oversampled, the validation set contained rows 4076:5440 -the third round, and the test set contained 4077:6803-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. A criterion for a good network was a high mean classification accuracy with a low standard deviation.

GRAPH EVALUATED MLP VS NARX

VARIETY OF LAYERS NUM OF NEURONS

CAN SEE THIS OVERALL NARX PERFORMED BEST

SO THEN CHANGED FEEDBACK AND DELAY SIZES

TO GET FINAL GRAPH BEST AS …

## Network:

PLOT OF BEST FOR ALL VALUES

CONFUSION MATRIX

DESCRIBE BEST NETWORK ARCHITECTURE

ALL STRUCTURAL INFO, PARAM

DIAGRAM

HOW IMPLEMENTED (MATLAB DEFAULTS)

## Results:

*Give a synopsis of the results obtained from the final selected network.*

*Relate these results back to the problem as stated – a MSE on its own is not helpful in judging how well something works. [6 marks}*

*Identify anything of interest in the results, such as areas of particularly good or poor performance, or variation between different training runs. [4 marks]*

# Section B:

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

This paper proposes a novel monocular visual odometry based on deep recurrent convolutional neural networks, where visual odometry is the process of determining the position and orientation of a robot by analysing camera images. The reason, for use with proposed active, remotely controllable, robotic capsule endoscope prototypes, which need precise and reliable real time pose estimation functionality. They are equipped with functionalities such as local drug delivery, biopsy and other medical functions, used in hospitals for screening the gastrointestinal tract and diagnosing diseases such as the inflammatory bowel disease, the ulcerative colitis and colorectal cancer.

### Main features of the architecture:

The paper proposes a novel RCNN architecture, which can successfully model sequential dependence and complex motion dynamics across endoscopic video frames. The architecture makes use of inception modules for feature extraction and RNN for sequential modelling of motion dynamics to regress the robot’s orientation and position in real time. The proposed DL network consists of three inception layers and two LSTM layers concatenated sequentially. The inception layers are extracting multi-level features the final inception layer passes the feature representation into the RNN modules.

RNNs are suitable for modelling the dependencies across image sequences and for creating a temporal motion model since it has a memory of hidden states over time and has directed cycles among hidden units, enabling the current hidden state to be a function of arbitrary sequences of inputs, using RNN, the pose estimation of the current frame benefits from information encapsulated in previous frames. Thus, the deep RNN consists of two LSTM layers with the output sequence of the first one forming the input sequence of the second one each containing 1000 hidden units

### Training the network:

The proposed system, which learns translational and rotational motions simultaneously to regress the 6-DoF pose, is trained on Euclidean loss using Adam optimization method.

The back-propagation 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. In addition to saving exponentially decaying average of past squared gradients, *v t* , Adam optimization keeps exponentially decaying average of past gradients, *m t* that is similar to momentum.

Architecture was trained using Caffe library and NVIDIA Tesla K40 GPU. Using back-propagation-through-time method, the weights of hidden units were trained for up to 200 epochs with an initial learning rate of 0.001. Overfitting meaning that the noise or random fluctuations in the training data are picked up and learned as concepts by the model, whereas these concepts do not apply to a new data and negatively affect the ability of the model to make generalizations, was prevented using dropout and early stop- ping techniques

Dropout regularization technique introduced is an extremely effective and simple method to avoid overfitting. It samples a part of the whole network and up- dates its parameters based on the input data. Early stopping is an- other widely used technique to prevent overfitting of a complex neural network architecture which was optimized by a gradient- based method. The approach is executed by splitting the dataset into a training and a validation set to evaluate the generalization capability of the model.

### Data:

There were two groups of training datasets. The first dataset was recorded on five different real pig stomachs whereby the second dataset which was only used for trai ing purposes, was captured using a non-rigid open GI tract model EGD (esophagus gastro duodenoscopy) surgical simulator LM-103.

To ensure that our algorithm is not tuned to a specific camera model, four different commercial endoscopic cameras were employed. For each pig stomach-camera combination, 2000 frames were acquired which makes for four cameras and five pig stomachs 40000 frames, in total.

As a second training dataset, for each of four cameras, we captured 10000 frames on an EGD human stomach simulator making 40 , 0 0 0 frames, in total. During video recording, Optitrack motion tracking system consisting of eight Prime-13 cameras and a tracking software was utilized to obtain 6-DoF localization ground truth data in a sub-millimetre precision (see Fig. 2 ) which was used as a gold standard for the evaluations of the pose estimation accuracy.

We created a testing dataset recorded using five different real pig stomachs, which were not used for the training section. For each pig stomach-camera combination, 2000 frames are acquired making 4000 0 frames, in total. We did not capture any synthetic dataset for the testing session since it is less realistic due to obvious patterns of such artificial simulators. For all of the video records, again Optitrack motion tracking system was utilized to obtain 6-DoF localization ground truth.

### Effectiveness:

To the best of our knowledge, this is the first monocular VO approach through deep learning techniques developed for the endoscopic capsule robot and hand-held standard endoscope localization. Neither prior knowledge nor parameter tuning is needed to recover the absolute trajectory scale contrary to monocular traditional VO approach. The proposed method solves several issues faced by typical visual odometry pipelines.

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

The results depicted indicate, 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 crispy motions, contrary to realEndoVO; GoogLeNet and ResNet50 path estimations which 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.

Solving the scale ambiguity for monocular camera, based VO makes our proposed DL based method more beneficial than traditional VO approach. As opposed to the traditional VO pipeline the DL-based VO do not require any explicit feature extraction, matching, outlier detection or multi-scale bundle adjustment-like parameter tuning requiring operations, which can be seen as further benefits of the proposed approach

We compare the performance of the proposed deep EndoVO with two of the widely used state-of-the-art SLAM methods. LSD SLAM is a direct image alignment-based method, which optimizes the geometry using all of the image intensities. In addition to higher accuracy and robustness particularly in environments with little key points. We believe that our 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. 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 the robots end of arm tooling (EOAT).

### Main features of the architecture:

Our 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.

Use ResNet We also introduce a multi-modal model which extracts features from both RGB and Depth images to predict the grasp configuration. We propose a single step prediction technique feed the entire image directly into DCNN to make grasp prediction on complete RGB-D image of the object. This solution is simpler and has less overhead.

Use ResNet-50, a fifty layer deep residual model, to solve this grasp detection problem. The ResNet architecture uses the simple concept of residual learning to overcome the challenge of learning an identity mapping. A standard feed-forward CNN is modified to incorporate skip connections that bypass a few layers at a time. Each of these skip connections gives rise to a residual block, and the convolution layers predict a residual that is added to the block’s input. The key idea is to bypass the convolution layers and the non-linear activation layers in kth residual block, and let through only the identity of the input feature in the skip connection.

We introduce two different architectures for robotic grasp prediction: uni-modal grasp predictor and 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, where as the multi-modal grasp predictor is a 3-D Grasp Predictor that uses multi-modal (e.g., RGB and Depth) information.

A ResNet-50 model that is pretrained on ImageNet is used to extract features from the RGB channels of the image. For a baseline model, we use a linear SVM as classifier to predict the grasp configuration for the object using the features extracted from the last hidden layer of ResNet-50.

In our 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. We use SGD to optimize our training loss and mean squared error (MSE) as our loss function.

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. Last fully connected layer is the output layer, which predicts the grasp configuration for the object in the image.

The ResNet-50 layers work as feature extractors for both the images. 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 feed into a shallow convolutional neural network with three fully connected layers. The fully connected layers use ReLU activation functions. We added a dropout layer after 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. Fig. 5 shows the complete architecture of our multi-modal grasp predictor.

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.

### Training the network:

Our 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 our uni-modal grasp predictor, we used SGD to optimize the model with hyper parameters in first stage. For fine-tuning the network in the second phase, we use a much lower learning rate and plateau the learning rate if the training loss does not decreases

Theoretically, a DCNN should have better performance with increased depth because it provides increased representational capacity. However, our current optimization method, stochastic gradient decent (SGD) is not an ideal optimizer. In experiments, researchers found that increased depth brought increased training error, which is not in-line with the theory. The increased training error indicates that the ultradeep network is very hard to optimize. This means that identity map is very hard to obtain in a convolutional neural network by end-to-end training using SGD. Therefore, we use residual layers as in ResNet [31], which reformulates the mapping function between layers, using the function given by

Large-scale image classification datasets have only RGB images. Therefore, we can pre-train our deep convolutional neural networks with 3-channels only. We introduce a unimodal grasp predictor model which is designed to detect grasp using only three channels (RGB or RGD) of the raw 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.

### 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:

* 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.
* Object-wise: splitting splits all the object instances randomly and all images of an object are put in 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 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 image that have a NaN value as they were occluded in the original stereo image. These pixels with NaN value were replaced by zeros. Pre-training was necessary when the domain-specific data available is limited as in the Cornell grasp dataset. 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 network is 6 times deeper as compared to the previous work by Lenz et al. An improvement of 14.94% for image-wise split and 13.36% for object-wise was made. 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 were able to achieve an accuracy of 93.4%, which is at par with the current state-of-the-art.

## Bibliography: