HARDWARE TESTBED AND DEEP NEURAL NETWORKS FOR MULTI-MODAL SENSOR FUSION

A Thesis

by

CHENYE ZHAO

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Peng Li
Committee Members,	Weiping Shi
	Shuiwang Ji
Head of Department,	Miroslav Begovic

May 2019

Major Subject: Computer Engineering

ABSTRACT

Deep neural networks (DNN) have been widely applied in sensor fusion, providing an end-to-end solution for fusion of features extracted from multiple sensory inputs. A class of new sensor fusion networks based on DNN called gating architectures proposed in recent years improves the prediction performances over the conventional fusion mechanisms employed in convolutional neural networks (CNNs). However, experimental results show that the gating architectures are not always robust and sometimes even under-perform conventional fusion methods.

In this work, the limitations of existing gating architectures are discussed and analyzed. Through experiments, we demonstrate that gating architectures fail to learn correct fusion weights for sensory inputs, showing the inconsistency between fusion weights and corresponding qualities of sensory inputs, and hence limit the prediction performance. We propose an improved fusion architecture by introducing the auxiliary path model to regulate the fusion weights in the gating architecture. We also provide in-depth studies on the regularization mechanisms to show that the improvements on performances are achieved by the more robustly learnt fusion weights.

Evaluations are performed under two different public datasets. We generate comprehensive sensor failure schemes, where the proposed architecture significantly outperforms a baseline non-gating architecture and one existing gating architecture. We also build up a sensor fusion hardware platform: a robot car, which is equipped with

multiple sensors. The robot will be further developed and adopted as a hardware platform for evaluating the proposed sensor fusion architecture.

DEDICATION

To my families, my professors, my friends.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Peng Li, and my committee members, Dr. Weiping Shi, and Dr. Shuiwang Ji, for their guidance and support throughout the course of this research.

Thanks also go to my group members, my friends and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to my mother and father for their encouragement and to my girlfriend for her patience and love.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis (or) dissertation committee chaired by Professor Peng Li of the Department of Electrical Computer Engineering.

This work is a collaborator work. Parts of the work were done by Yang Li in the Electrical Computer Engineering Department. Section 2.2 is partially done by Mr. Amarnath Mahadevuni. Section 6.3 was partially done by Mr. Yang Li and Mr. Myung Seok Shim. All other work conducted for the thesis (or) dissertation was completed by Chenye Zhao independently.

Funding Sources

This is made possible by NPRP grant # NPRP 8-274-2-107 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION:

1.1 Sensor Fusion

Sensor fusion has been a popular topic to be studied in both industry and research. The definition of sensor fusion is to build up a more robust and dependable multi-model system by combining multiple sources of sensory data. The meaning of robustness and dependability in our work can be discussed under two different cases. Firstly, with all sensory inputs data clean, individual sensors cannot contribute enough information themselves, and the information provided through these sensors are not full mutually dependent. In this case, designing a smart fusion method can allow each sensor to provide its own useful information as complementary information for other sensors and therefore the integrated system can receive an overall boosted performance, which cannot be achieved through individual sensors. Secondly, with one or multiple sensors corrupted, a good sensor fusion technique shall to some extent compensate for the losses brought by corrupted sensors through correctly utilizing information provided by sensors that are working properly. Many complicated sensor failure situations can be studied here. For example, the numbers of corrupted sensors in different examples are not fixed. In this work, we create multiple sensor failure cases to fully evaluate the proposed sensor fusion architecture.

Sensor fusion has been studied in some works. Sensors such as Inertial measurement units (IMUs) have already been embedded into portable devices, providing the information for activity recognition. There are some sensor fusion algorithms proposed

to solve human activity recognition problems (Dehzangi, Taherisadr, & ChangalVala, 2017) (Zhao & Zhou, 2017) (Gravina, Alinia, Ghasemzadeh, & Fortino, 2017) (Yurtman & Barshan, 2017). (Ramachandram & Taylor, 2017) gives a survey on fusion architectures.

1.2 Deep Neural Networks for Sensor Fusion

Convolutional neural networks. Convolutional neural network (CNN) (LeCun & Bengio, 1995) is a class of deep neural network (DNN). Different from regular neural networks, CNN adopts the idea of weight sharing, which significantly stimulates the efficiency of neural network and reduces the number of tunable parameters within the network. The idea of convolutional neural network comes from the connectivity pattern of neurons within human brain. More specifically, the idea is inspired by the functioning of visual cortex: single neuron only responds to stimuli within a limited vision field named receptive field. The visual information are generated through multiple overlappings of such receptive fields. Therefore, CNN is widely applied in analyzing visual image, doing calculations like classification and regression. A basic CNN consists of a sequence of layers, mainly includes: convolutional layer, pooling layer, and fully-connected layer.

A convolutional layer consists of several filters, as in Fig. 1, where the input to this convolutional layer is of size $5 \times 5 \times 2$. There are two filters for this layer: w_0 and w_1 with sizes of $(2 \times 2 \times 2)$. The convolution calculation is then performed between filters and input: calculation on dot product are performed between filters and corresponding channels of inputs. As in Fig. 1, filter share weights during one convolution, which means that for one input example, there are only 16 tunable parameters.

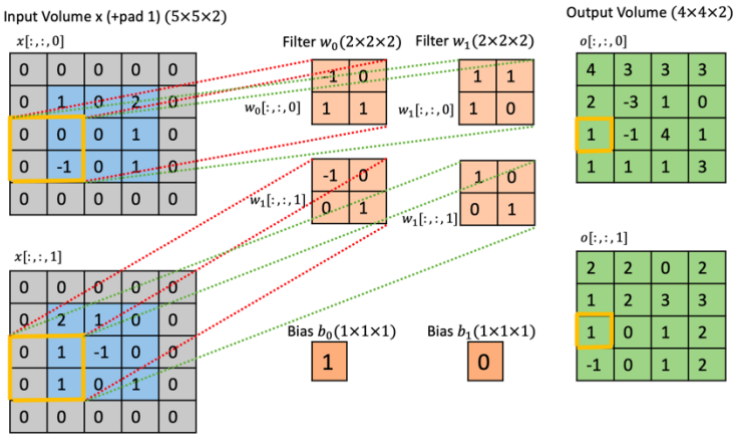

Figure 1. Filters in a convolutional neural network

The output of convolutional layer will usually be fed into an activation function to increase the learnability of non-linear functions of CNN. A pooling layer (also named as subsampling layer) is often inserted after a convolutional layer. The idea is to further reduce the sizes of features in order to reduce the number of tunable parameter and computational cost. Max-pooling is the most popular choice.

Convolutional neural network has been widely studied and applied in various fields. With its strong learning abilities, convolutional neural network is capable of dealing with many difficult problems. For example, traditional computer vision subjects such as object detection, object segmentation can be solved by convolutional neural networks with better performances than conventional methods. Since CNN saves the number of tunable parameters by weight sharing, neural networks can be built in very deep layers with still acceptable computational costs. Many classical image recognition architectures and the

resultant architectures have been developed such as AlexNet (Krizhevsky, Sutskever, & Hinton, 2012) and VGG (Simonyan & Zisserman, 2014). For 2D object detection, architectures like R-CNN (Girshick R. a., 2014), SPPNet (K, X, & S, 2014), Fast R-CNN (Girshick R. , 2015), Faster R-CNN (Ren, He, Girshick, & Sun, 2015), SSD (Liu, et al., Springer) and YOLO (Redmon, Divvala, Girshick, & Farhadi, 2016) produce good results.

Deep neural networks for sensor fusion. The applications of deep neural networks to sensor fusion provide the problem with an end-to-end solution. For example, sensory inputs such as cameras and LIDARs can be fused to produce better image recognition performances on deep neural networks (Jain, et al., 2016) (Bohez, et al., 2017) (Kim, et al., 2018) (Chen, Ma, Wan, Li, & Xia, 2017) (Wei, Cagle, Reza, Ball, & Gafford, 2018) (Garcia, Martin, De La Escalera, & Armingol, 2017) (Karpathy, et al., 2014) (Mees, Eitel, & Burgard, 2016). In (Chen, Ma, Wan, Li, & Xia, 2017), a sensor fusion architecture for 3D object detection based on deep convolutional neural network is proposed, where KITTI dataset (Geiger, Andreas, Lenz, & Urtasun., 2012) is adopted. Sensory data captured by LIDAR and camera located on testbed car consists of LIDAR bird view, LIDAR front view and RGB images. The architecture also compares three traditional fusion methods in deep neural networks: early fusion, late fusion, and deep fusion. As shown in Fig. 2. The idea of early fusion shown in Fig. 2(a) is to fuse the sensory features in early stage of DNN. Here fusion is done by simply concatenating different features. Fig. 2(b) demonstrates the late fusion, where fusion is done after sensory inputs have been pre-processed by early-stage intermediate layers. Deep fusion method splits fusion into

different stages: features are first blended through element-wise mean, then three copies of features are fed into corresponding intermediate layers, whose outputs are fused again. Repeat the fusion-and-split step for three times to get the fused data.

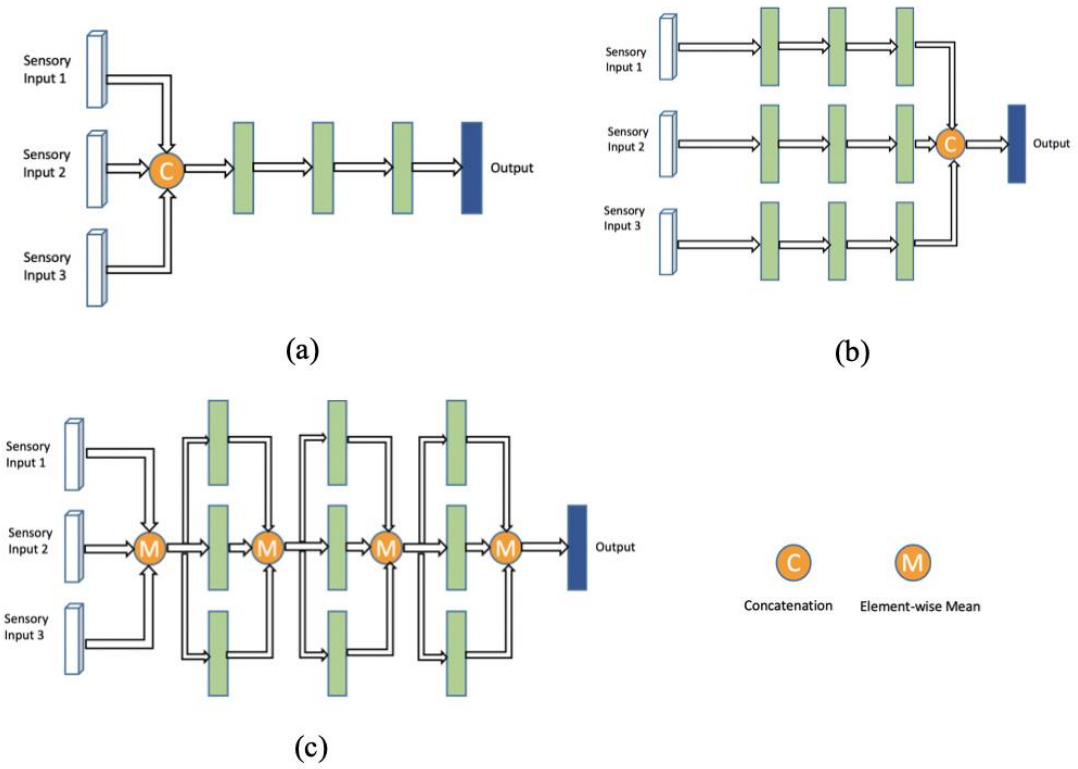

Figure 2. Three traditional fusion schemes in deep neural network. (a) Scheme of early fusion. (b) Scheme of late fusion. (c) Scheme of deep fusion.

Situations of sensor failures are not taken into consideration and no related analyzation is presented in this work. However, a resilient sensor fusion architecture

requires the sensor fusion architecture to be robust under both clean sensory inputs and sensor failures.

Previous work. A sensor fusion architecture of multiple deep experts is proposed in (Mees, Eitel, & Burgard, 2016), where multiple sensory inputs are first processed individually through convolutional neural networks to get the classification results. The outputs of classifiers are fused through weighted-sum by first extracting the fusion weights from multiple experts. A NetGated architecture is proposed in (Patel, Choromanska, Krishnamurthy, & Khorrami, 2017), where sensory inputs from camera and LIDAR equipped on a robot car are pre-processed by convolutional layers and fully-connected (FC) layers. The pre-processed features are then fed into a fusion weight extracting architecture to generate the fusion weights of two sensory inputs. The outputs of last early-stage FC layers are being weighted by fusion weights to compute the weighted sum as the fused feature. The fused feature is then processed by another FC layer to get the steering commands of the robot car's. More details about NetGated architecture will be discussed in following sections. A similar gating architecture is proposed in (Kim, et al., 2018), where fusion weights are extracted by first concatenating all sensory features and then processed by convolutional layers. Then fusion weights are multiplied with corresponding features and added up to produce fused feature maps.

Our contributions on sensor fusion algorithm. Gating architecture shows the improvements on robustness and prediction performances of deep neural networks especially under sensor failure cases comparing with traditional fusion methods without gating architecture. However, there are still some cases where the gating architectures

underperform the conventional non-gating architectures. For the black-box nature of deep neural network, it is difficult to give a clear analysis on the relationships between qualities of sensory inputs, fusion weights and the resulting performances.

Based on the previous works on gating architecture and their observed limitations, we proposed an optimized gating architecture which improves the robustness as well as resulting prediction performances with both clean and corrupted sensory inputs.

The contributions of this works are as follows:

- Propose a sensor fusion architecture called *ARGate*, which adopts the auxiliary path model to regularize the fusion weights of gating architecture in order to more robustly represent the qualities of corresponding sensory inputs (Zhao, Shim, Li, Zhang, & Li, 2019) .
- Based on ARGate, two regularization techniques are proposed: *weight sharing*, *regularization* on *fusion weight*. Equipped with proposed regularization techniques, ARGate shows significant improvements on performances and robustness under both clean sensory inputs and sensor failures.
- In-depth analysis is provided regarding to the functionalities of ARGate and NetGated, which explains the limitations of gating networks, as well as the effectiveness of fusion weights regularization to the improvements of prediction accuracy.

- Baseline models and proposed ARGate architectures are evaluated under two public datasets: HAR (Anguita, Ghio, Oneto, Parra, & Reyes Ortiz, 2013) and CAD-60 (Sung, Ponce, Selman, & Saxena, 2012).

To finish this work, I have close co-operations with my groupmates Mr. Yang Li and Mr. Myung Seok Shim. My divisions on the work include: proposing the ARGate and regularization techniques *weight sharing*, *fusion weight regularization*, analyzation on the mechanism of NetGated architecture and ARGate. I also experimentally shed light on the responsibilities of auxiliary-model regularization on the performance improvements and implement the all experiments on HAR dataset and part of the results on CAD-60 dataset.

1.3 Deep Neural Networks on Mobile Robot Car

Machine learning algorithms have been widely used on mobile robot car. In (Mahadevuni & Li, 2017), a reinforcement learning algorithm based on spiking neural network is proposed and simulated on mobile robot car. Different types of sensors have also been implemented on robots. In (Patel, Choromanska, Krishnamurthy, & Khorrami, 2017), camera and Lidar embedded on a mobile robot capture sensory data for the deep learning sensor fusion architecture, providing the robot with the knowledge of surrounding environments and hence helping make string commands.

In our work, I build up the sensor fusion hardware platform: a robot car equipped with multiple sensors as the basic testbed of our sensor fusion architecture in the future, this work is finished together with my former groupmate: Mr. Amarnath Mahadevuni. My divisions of work include: setting up FSM and PRU.

The rest part of this thesis is organized as follows:

Section 2 presents background of gating architecture, limitations and basic information about the multi-sensor mobile robot car. Section 3 introduces the ARGate architecture, corresponding regularization techniques and implementation details of mobile robot car. Section 4 introduces experimental settings and dataset. Section 5 illustrates the experimental results. Section 6 is the conclusion.

2. BACKGROUND AND PROBLEM

2.1 CNN Fusion Architecture

We implement the state-of-art CNN fusion architecture with traditional fusion method: element-wise mean. As shown in Fig. 3.

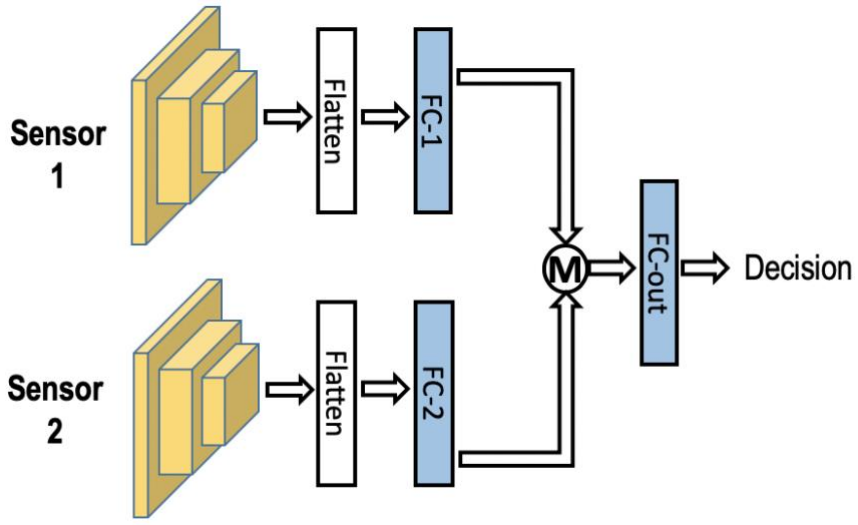

Figure 3. CNN baseline fusion architecture

Where "M" represents element-wise mean. Sensory inputs from Sensor 1 and Sensor 2 are first processed through convolutional layers and early fully connected layers "FC-1" and "FC-2". Fusion is done by calculating the element-wise mean value of outputs of "FC-1" and "FC-2". Then decision is made by going through another "FC-out" layer.

2.2 Gated Architecture

In (Patel, Choromanska, Krishnamurthy, & Khorrami, 2017), gated architecture is proposed in order to deal with sensor failures in a sensor fusion system whose aim is to enhance the performance of an unmanned ground vehicle. The vehicle is embedded with a camera and a LIDAR. The proposed gated architecture is called NetGated. Data from each sensor are first processed through multiple convolutional layers and fully connected layers individually in order to get the extracted features. Instead of applying fusion methods such as element-wise-mean or concatenation, fusion is done by feeding features into a gating architecture to calculate the scalar values that represent the qualities of corresponding sensory inputs. The network fuses the data from two sensors with the scalar values as weighting parameters. In this work, we name the scalar values as fusion weights. Fig. 4 demonstrates more details about the idea of NetGated.

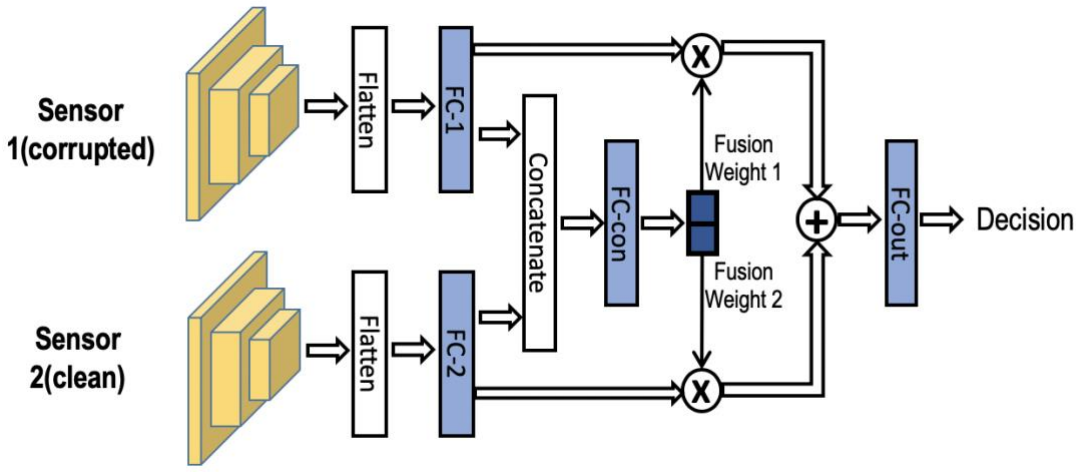

Figure 4. NetGated Architecture

As in Fig. 4, there are two input sensors in this system: Sensor 1 and Sensor 2. Sensory inputs of two sensors are first processed by convolutional layers(yellow) and fully connected(FC) layers(blue) separately. Features output from “FC-1” and “FC-2” are then concatenated to be further processed by the FC-con layer, whose result consists of two scalar values representing qualities of Sensor 1 and Sensor 2, named as “Fusion Weight1” and “Fusion Weight 2”. Fusion weights are then multiplied with output features of “FC-1” and “FC-2” to perform the weighted sum of two features as the fused data. The fused data are then fed into the last decision-making layer “FC-out” to produce the final decision.

2.3 Limitations of Gated Architecture

In NetGated Architecture, fusion weights should reflect the qualities of corresponding sensory inputs. If information provided by each sensory input contributes equally to the final prediction performance, the optimal effect of fusion weights should be: fusion weights of all sensors have similar values when all sensory inputs are clean. If one sensor x is corrupted while the rest sensors are clean, the fusion weight of sensor x should be much smaller than those of the rest sensors.

However, the gating architecture often shows the inconsistency between fusion weights and sensory qualities. Through our experimental results, we observe that fusion weights are sometimes fail to consistently represent the qualities of corresponding sensors. For example, in our implementation of NetGated Architecture under HAR dataset, the network has nine input channels, among which body_acc_x , body_gyro_x are fixed as corrupted inputs, whose inputs are pure Gaussian noise with no information.

When we print out the values of fusion weights of nine channels, fusion weight of body_acc_x is the largest among all fusion weights. This result shows that fusion weights sometimes tend to be unstable and behave inconsistently with qualities or importance of corresponding sensory inputs. We also cast doubt that the poor prediction accuracy might be related to the inconsistent fusion weights.

2.4 Analysis

Based on the inconsistency of fusion weights mentioned in last section, we propose a tentative analysis. In Fig. 4, the output features of "FC-1" and "FC-2" are first concatenated before "FC-con". If the concatenation layer is removed, and two fusion weights are extracted separately from two features generated by "FC-1" and "FC-2", it would be hard for the neural network to learn the "comparative importance" of Sensor 1 and Sensor 2. If Fusion Weight 1 is greater than Fusion Weight 2, we still cannot confirm that whether Sensor 1 is more important or better in quality than Sensor 2, because the values of Fusion Weight 1 only represent the quality or importance of Sensor 1 comparing with itself, same for Sensor 2.

However, concatenating the features may bring another problem. As in Fig. 5, Feature 1 and Feature 2 are first concatenated, and we expect corresponding Fusion Weight 1 to represent quality of Feature 1. But the outputs of concatenation layer are fully connected with "FC-con", which means that information from Feature 1 and Feature 2 are mixed here. As in Fig. 4, red and green lines represent data flowing from Feature 1 and Feature 2 to first output neuron of FC-con, while black and blue lines represent data flowing from Feature 1 and 2 to second output neuron of FC-con. Fusion

Weight 1 contains information from both Feature 1 and Feature 2 and same thing for Fusion Weight 2.

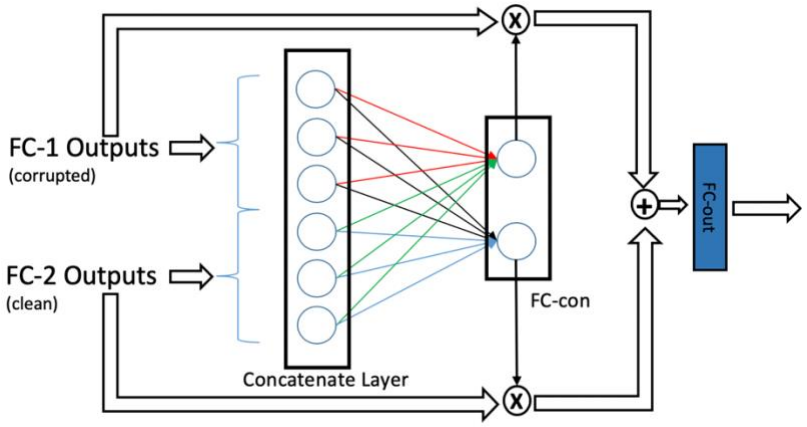

Figure 5. Mixture of data inside NetGated

This explains the reason why corrupted sensor may have larger fusion weights than clean sensor.

2.5 Control of Mobile Robot Car

In order to build up a sensor fusion hardware platform. With Amarnath Mahadevuni, we implemented a robot car. The robot car is designed to have multiple sensors, including USB camera, ultrasonic sensors and digital compass. The robot is able to move to the preset target, detect its surroundings. If there are obstacles detected, the robot is able to recognize the obstacles and avoid them. This hardware platform is able to

gather distance information by 5 ultrasonic sensors and direction information captured by digital compass. The “brain” of mobile robot is the single-board processor Beaglebone black. The robot is powered by eight 1.5Volts batteries. Motion is controlled by two servo motors.

BeagleBone Black. BeagleBone Black is a powerful single-board processor powered by Texas Instruments. The Beaglebone Black is equipped with an AM335x ARM processor and 2 32-bit PRU microcontrollers. One of the reasons that BeagleBone Black is very suitable for multi-sensor system is because of its large number of pin headers: 2*46 pin headers for analog/digital inputs/outputs, which enable us to get enough devices connected. Devices can get connected through I^2C, GPIO and so on. The BeagleBone Black also has one USB host, Ethernet connection and HDMI port. BeagleBone ships with Linux kernel 3.8. Fig. 6 shows a general view of BeagleBone Black on the implemented robot car:

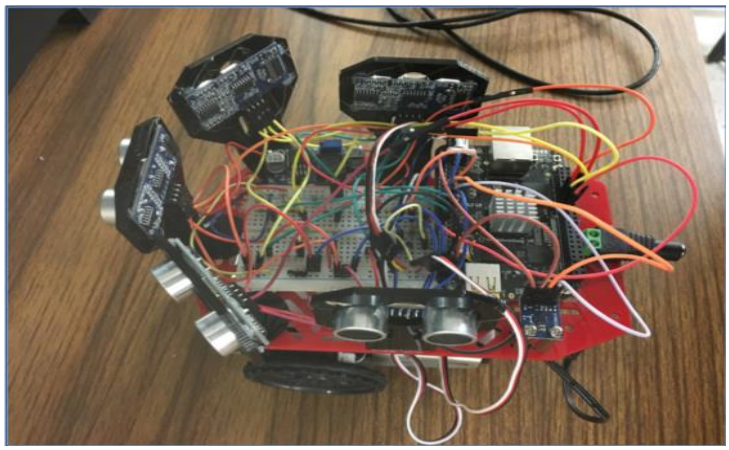

Figure 6. Robot car

Ultrasonic sensor. We adopt 5 HC-SR04 ultrasonic sensors to detect the distance information for the robot car. Five ultrasonic sensors are placed on the robot with intervals of 45°. As shown in Fig. 6.

As shown in Fig. 7, the unit vector on y axis indicates the direction that the robot is moving to, which is calculated by the digital compass.

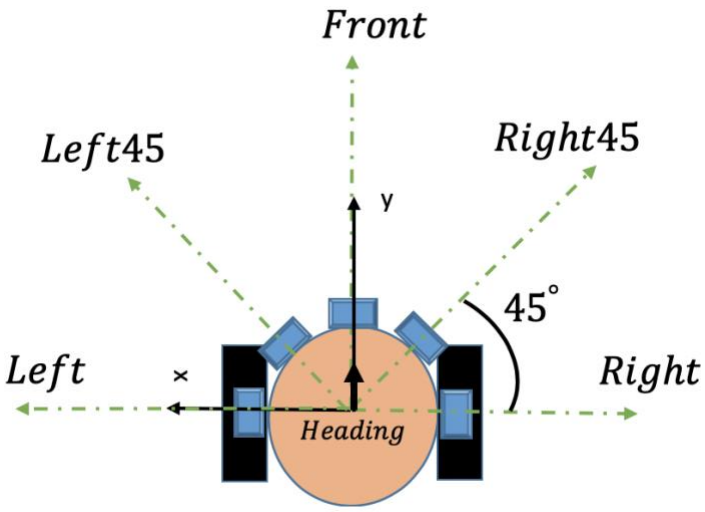

Figure 7. Distributions of 5 ultrasonic sensors on mobile robot

One ultrasonic sensor named *Front* is located here which detects the distance information right in front of the robot. There are also two ultrasonic sensors located on the two sides of the robot: *Left*, *Right*, aiming to detect the distance information on the two

sides of mobile robot. In case distance information may be missed between *Front* and *Left* and *Right*, two additional sensors are located as in Fig. 7 named *Left*45 and *Right*45, respectively.

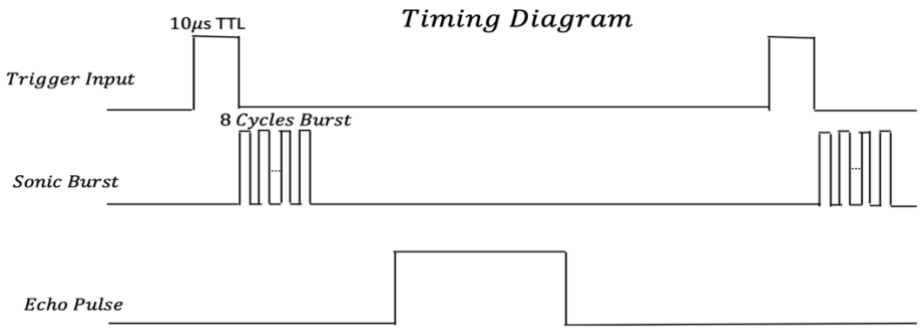

Figure 8. Timing diagram of HC-SR04.

For each single ultrasonic sensor, there are four signals: Trigger, Echo, VDD and GND.

Trigger is the output pin, which is used to send out a small TTL pulse with a high level of 10uS to trigger the ultrasonic sensor into working mode. The sensor sends out a serials of sonic burst to the outside environment. Echo is the input pin of ultrasonic senor to receive the echoes of sonic burst. After echo is received, the length of high level of echo pulse will be translated into distance information. The HC-SR04 is powered by 5Volts DC voltage.

The distance information returned by the ultrasonic sensor is 5 Volts, however the voltage of GPIO pins on BeagleBone Black can only receive 3.3 Volts signals,

therefore we build up a simple circuit to divide voltage from 5V to 3.3V with three 1Kohms resisters.

Digital Compass HMC5883L. The Honleywell HMC5883L is a multi-chip module with the capable of magnetic sensing embedded with the interface of digital signals for compassing. The HMC5883L can be implemented through I^2C bus. Digital Compass is implemented in order to help robot find the direction. An certain angle is set as the target direction for the robot. The 3-axis magnetic direction information returned by the compass is transformed into angle information to help robot calculate the difference between its current heading and the target heading.

Finite State Machine(FSM). Finite State Machine(FSM) is adopted to handle the basic motion control of the robot car. The state machine includes three states: Go-to-Goal, PRE-Wall-Follow and Wall-Follow.

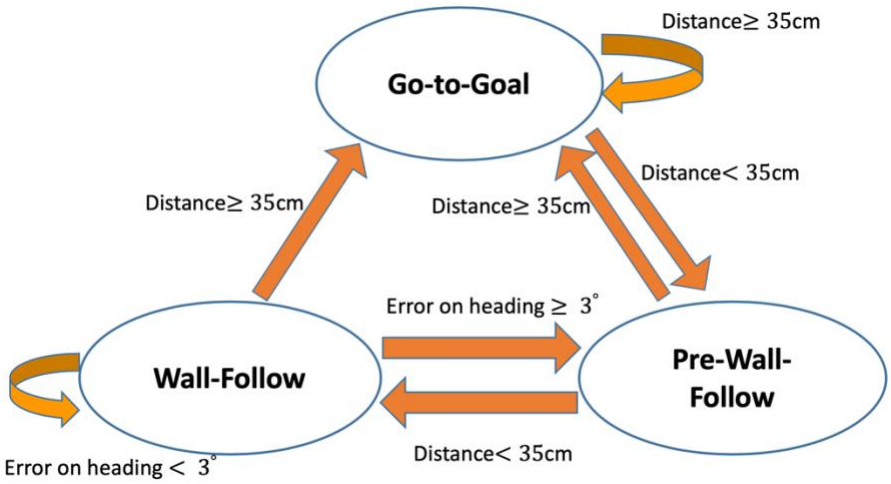

Figure 9. Finite State Machine

In Go-to-Goal state, distances information returned by of n out of 5 ultrasonic sensors are less than 35cm, where $n \in \{0, 1\}$ meaning that less than 2 sensors detect the obstacles, representing the situations when robot is on the clear route towards the target direction. PID controller is implemented here to control the robot to move towards the target.

Pre-Wall-Follow. In our implementation, the surfaces of obstacles are assumed to be flattened. Therefore we name this state as Pre-Wall-Follow. When there are over 1 ultrasonic sensors returning distance information less or equal to 35cm, the robot enters the Pre-Wall-Follow state. The robot will stop and begin to adjust the heading to be in parallel with the wall. Therefore the next job is to compute the direction of the wall.

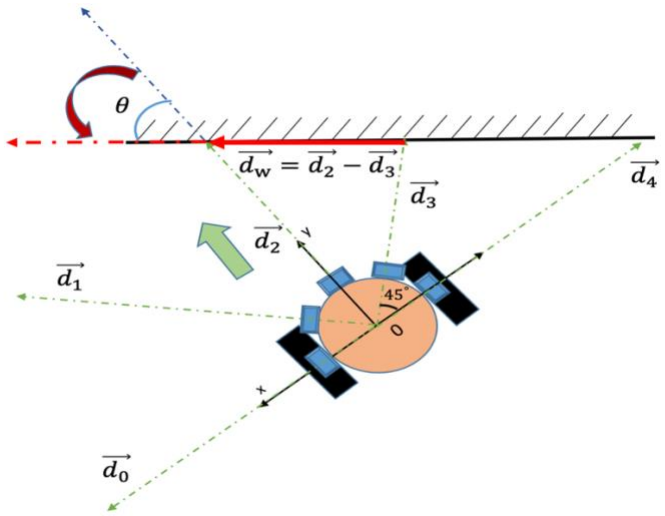

Figure 10. Calculate the direction of obstacle

As shown in Fig. 10, a coordinate system is built up on the robot, the vector from the center of robot to the *Left* sensor is defined as positive x axis, the direction from the center of the robot to the *Front* represents the positive y axis.

The angles between each sensor according to positive x axis are: $0^{\circ}, 45^{\circ}, 90^{\circ}, 135^{\circ}, 180^{\circ}$. The distance values returned by all five sensors are the norms of the five vectors: $\overrightarrow{d_0}, \overrightarrow{d_1}, \overrightarrow{d_2}, \overrightarrow{d_3}\ and\ \overrightarrow{d_4}$. As in Fig. 10, when distance values returned by $\overrightarrow{d_2}\ and\ \overrightarrow{d_3}$ are less than 35cm, the robot enters PRE-WALL-FOLLOW state. In our implementation, the robot turns left to avoid the obstacle, therefore the direction of the wall can be computed as $\overrightarrow{d_w} = \overrightarrow{d_2} - \overrightarrow{d_3}$. Since the heading of robot is represented as the unit vector of $\overrightarrow{d_2}$: $\overrightarrow{d_c} = \frac{\overrightarrow{d_2}}{|\overrightarrow{d_2}|}$, which is $(0, 1)$, as the green arrow in Fig. 10. The robot need to adjust its heading by the difference between $\overrightarrow{d_w}$ and $\frac{\overrightarrow{d_2}}{|\overrightarrow{d_2}|}$. The xy coordinate of vector $\overrightarrow{d_2}$ can be computed as:

$$\begin{cases} x_2 = \overrightarrow{d_2} \cdot \cos(2 * 45^{\circ}) \\ y_2 = \overrightarrow{d_2} \cdot \sin(2 * 45^{\circ}) \end{cases}$$

Similar calculation of coordinates of $\overrightarrow{d_3}$:

$$\begin{cases} x_3 = \overrightarrow{d_3} \cdot \cos(3 * 45^{\circ}) \\ y_3 = \overrightarrow{d_3} \cdot \sin(3 * 45^{\circ}) \end{cases}$$

Coordinates of $\overrightarrow{d_w}$ are:

$$\begin{cases} x_w = x_2 - x_3 \\ y_w = y_2 - y_3 \end{cases}$$

Angle that needs to be adjusted:

$$\theta = \arccos\left(\frac{\overrightarrow{d_w} \cdot \overrightarrow{d_c}}{\left|\overrightarrow{d_w}\right| \cdot \left|\overrightarrow{d_c}\right|}\right) = \arccos\left(\frac{x_w \cdot 0 + y_w \cdot 1}{\sqrt{({x_w}^2 + {y_w}^2)} \cdot 1}\right)$$

Then the robot adjust its heading by θ , as the brown arrow in Fig. 10. If there is no distance value from any ultrasonic sensor that is less than 35cm, the robot goes back to Go-to-Goal state. Otherwise the robot enters the Wall-Follow state.

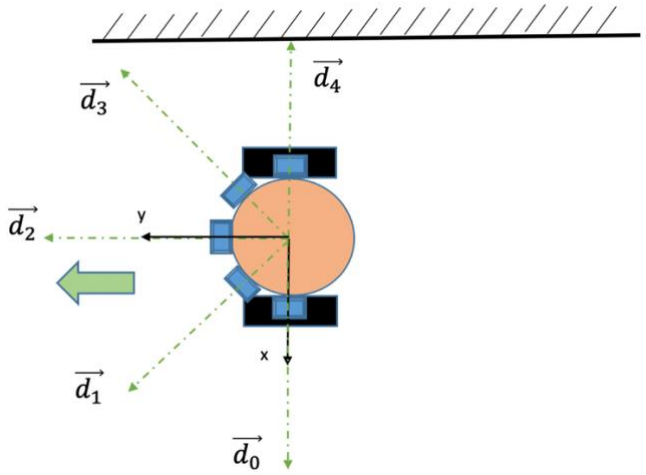

Figure 11. Adjust the heading in Pre-Wall-Follow state

Wall-Follow. In Wall-Follow state whenever there are more than two ultrasonic sensors returning the distance values less than 35cm as the robot moving alongside of the obstacle, the robot calculates the heading of obstacle.

If the difference between its current heading and the new direction is less than 3°, the robot continue its previous heading. Else, the robot goes back to Pre-Wall-Follow state to adjust its heading according to the new direction of wall. As shown in Fig. 12.

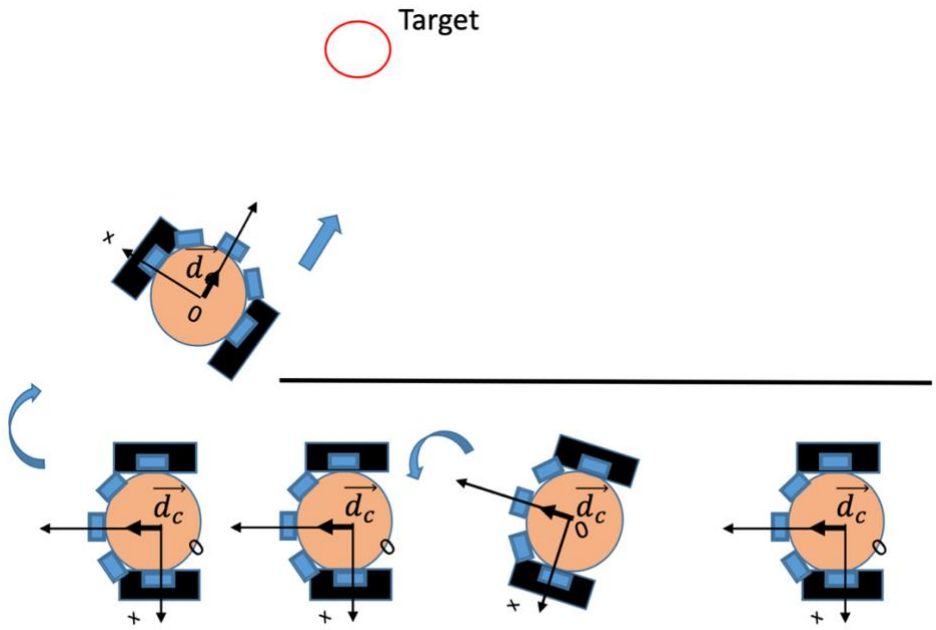

Figure 12. Wall-Follow state

After implementing the FSM, an experimental problem occurs: the robot often bumps into the obstacle where it should make turns and follow the wall. A few assumptions are considered: firstly, the ultrasonic sensors are not working properly. Secondly, the robot moves too fast for ultrasonic sensors to catch the obstacles. In order

to verify the first assumption, we put obstacles next to the robot when it's not moving and run the distance detection program, the ultrasonic sensors can accurately return the distance information of obstacles. Therefore, it is clear for us that the robot needs the distance information in real-time.

3. SOLUTIONS

3.1 Auxiliary Paths

In (Chen, Ma, Wan, Li, & Xia, 2017), an auxiliary path architecture is applied in the fusion layers to help further improve the prediction performance. As shown in Fig.13.

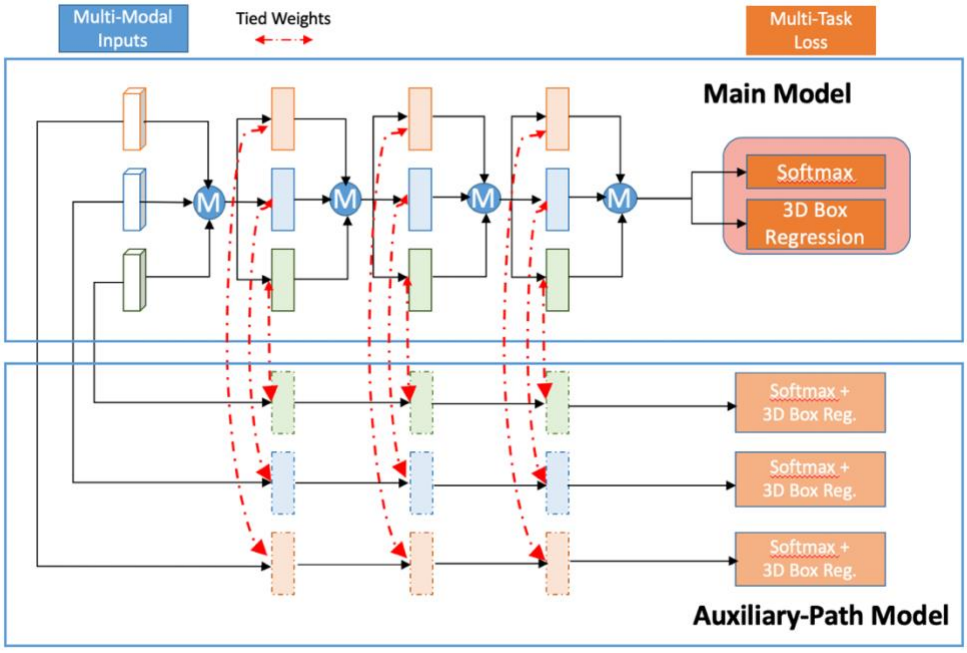

Figure 13. Auxiliary model

In Fig. 13, each Multi-Modal Input represents sensory input of one sensor. We name the top three channels as "Main Model" and the bottom three channels as "Auxiliary-Path-Model". M in blue represents the fusion method of "element-wise mean". Multi-Task Loss of main model consists of the $Softmax$ function to do object classification and a $3D\ Box\ Reg$ for the regression of 3-D bounding boxes. Multi-Modal Inputs are also processed through three additional channels called Auxiliary Paths, each layer on auxiliary paths share weights with corresponding layer in main model. The total loss function for this architecture is:

$$\text{Total Loss} = \text{Multi Task Loss} + \sum_{i=1}^{3} (Softmax + 3D\ Box\ Reg)_i$$

Where Multi-Task Loss represents the loss function of main model, and $(Softmax + 3D\ Box\ Reg)_i$ is the classification loss and regression loss of the $i - th$ auxiliary path. The optimizer should optimize both main model and auxiliary path model.

According to the experimental results in (Chen, Ma, Wan, Li, & Xia, 2017), prediction performance is improved by 1.8% on 3D object detection, and 6.3% on 2D object detection based on KITTI validation dataset after the implementation of auxiliary path model.

A tentative analysis of the function of auxiliary paths is as follows: through weight sharing and including loss functions of auxiliary paths into the total loss function, the optimizer has to consider both main model and auxiliary path model. There's no fusion step in auxiliary paths, therefore layers on auxiliary paths are to some extent

regularizing layers in main model to prevent them from getting overfitted in the deep fusion architecture.

Based on the idea of auxiliary paths, we propose our own fusion architecture.

3.2 Auxiliary Paths on Selected Channels

We introduce the idea of auxiliary paths into gating architecture. For the problem of fusion weight inconsistency mentioned above, one reason might be because the gating architecture is not fully trained, so that fusion weights need to receive some extra guidance to get further trained, in order to correctly represent the importance of corresponding sensory inputs. Therefore based on the auxiliary path idea mentioned in last section, we set the auxiliary path as the competitor of the main model, whose job is to "push" gating architecture to "force" dataflow within NetGated architecture to get "less mixed" so that fusion weights can be more consistently representing qualities of sensory inputs.

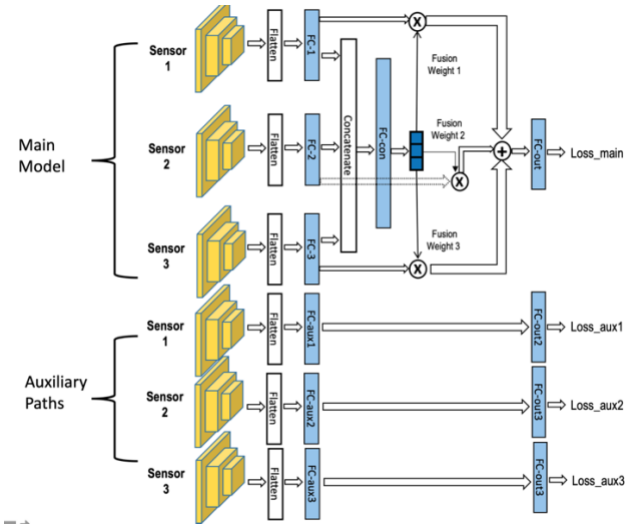

Figure 14. Auxiliary-model on selected channels

Following this idea we implement the NetGated architecture with auxiliary-model as shown in Fig. 14. Beyond the NetGated network, three additional channels with inputs of Sensor 1, 2 and 3 are added as auxiliary paths. The sensory inputs are processed through convolutional/pooling layers and fully connected layers with the same sizes as in main model, no weight sharing is implemented here. In order to boost the accuracy of main model by auxiliary paths, we design the following loss function:

$$\text{Loss}_{total} = \begin{cases} \alpha \cdot Loss_{main} + \sum_{i=1}^{n} Loss_{aux_i}, & if\ Loss_{main} \geq \min\left(Loss_{aux_i}\right) \\ Loss_{main} + \sum_{i=1}^{n} Loss_{aux_i}, & otherwise \end{cases} \quad (1)$$

Where i is the $i-th$ auxiliary path, α is a user-specified weighting factor, $\min\left(Loss_{aux_i}\right)$ is the minimum value of all the loss functions of auxiliary paths.

As shown in the equation of Loss_{total} above, when $Loss_{main} \geq \min(Loss_{aux_i})$, corresponding to cases when main model performs worse than the auxiliary path with minimum loss function, then $Loss_{main}$ is multiplied with a factor α , which is set to be 5 in our experiments, in order to raise the weighting of $Loss_{main}$ in total loss comparing with $Loss_{aux}$. And hence main model will be more emphasized during training. This loss function treats auxiliary paths as competitors to the main modal.

However, based on our experimental experiences, the prediction accuracy of a NetGated architecture of multiple sensors is always better than architectures using single sensors as the current settings of auxiliary path model. For this reason, the auxiliary paths architecture shown in Fig. 14 is not competitive enough for the main modal.

Which also means in Loss_{total}, the condition of $Loss_{main} \geq \min(Loss_{aux_i})$ may never occurs. Therefore we need to find a more competitive design of auxiliary paths.

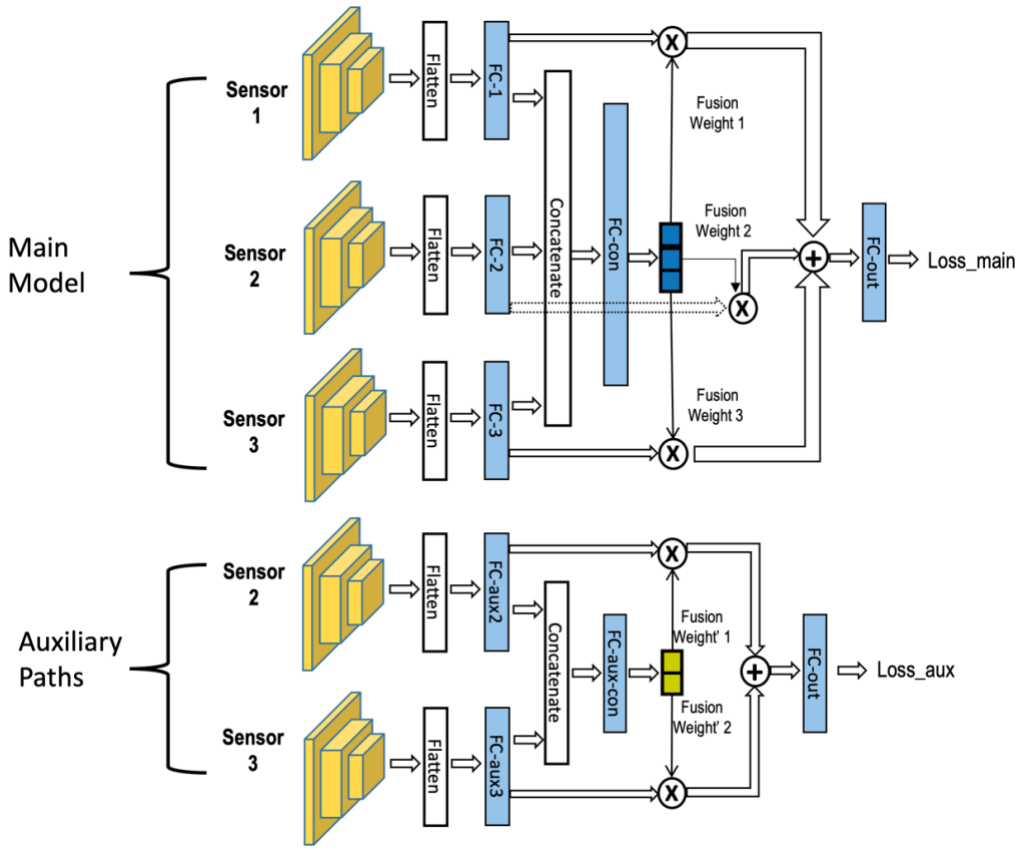

Figure 15. Auxiliary-model based on NetGated architecture with selected channels

Based on the analyzations above, the optimal choice of auxiliary path is the NetGate architecture with part of sensory inputs among all the sensors. As shown in Fig. 15.

In Fig. 15, we consider a sensor failure situation when Sensor 1 is corrupted, the noises brought by Sensor 1 are mixed up with clean sensory inputs of Sensor 2 and Sensor 3, and therefore bringing the prediction accuracy down. If the negative influence brought by noises on Sensor 1 is large enough that the performance of main model may be worse than the performance of NetGate architecture with Sensor 2 and Sensor 3 as inputs, then setting auxiliary path as the NetGated architecture of Sensor 2 and Sensor 3 will be a competitive choice.

Then Loss_{total} is designed as:

$$\text{Loss}_{total} = \begin{cases} \alpha \cdot Loss_{main} + Loss_{aux}, & if\ Loss_{main} \geq Loss_{aux} \\ Loss_{main} + Loss_{aux}, & otherwise \end{cases} \quad (2)$$

Where α (set as 5) is the user-specified weighting factor to adjust the weighting between $Loss_{main}$ and $Loss_{aux}$. If main modal performs worse than the auxiliary path model, more weighting will be put on $Loss_{main}$. In this design, the auxiliary path need to be selected based on previous knowledge of the noisy channel. Which sometimes may not be available. Through experimental results, fusion weights show more consistency with qualities of sensory data.

3.3 Weight Sharing

Auxiliary path on selected channels architecture mentioned in last section requires the previous knowledge of the noisy channel, otherwise we can't design the proper auxiliary path model. It also has limitations when multiple channels are corrupted, even the performance of NetGated network with inputs of all the clean channels is not competitive enough to the main model.

Therefore, we propose the Auxiliary Regulated Gate (ARGate) architecture with weight sharing. As shown in Fig. 16. Assuming there are two auxiliary paths, parameters of convolutional layers and early fully connected layers in main modal are shared with those of corresponding layers of auxiliary paths. In order to avoid the pre-knowledge of noisy channels, the loss function is designed as follows:

$$\text{Loss}_{total} = \alpha \cdot Loss_{main} + \sum_{k=1}^{K} Loss_{aux}^{k} \quad (3),$$

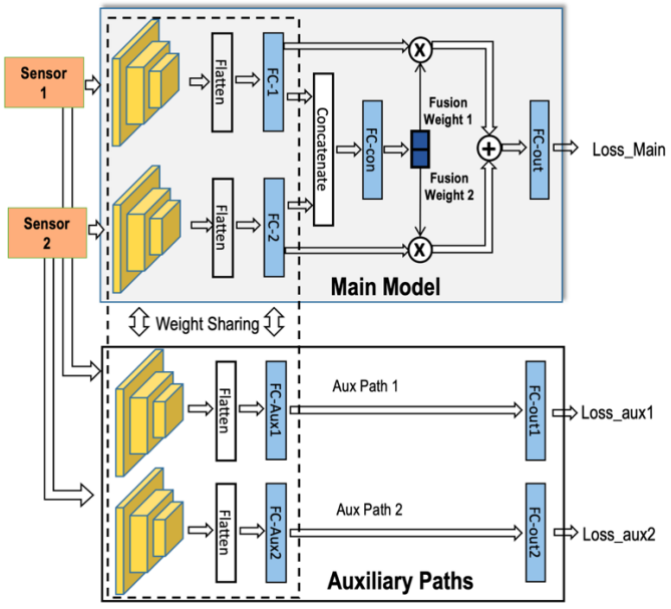

Figure 16. Auxiliary Regulated Gate with Weight Sharing

where $Loss_{main}$ and $Loss_{aux}^{k}$ are the loss functions of the main model and the $k-th$ auxiliary path, respectively. α represents an user-defined weighting parameter.

Since we need to consider the losses of all auxiliary paths in the total loss function, the convolutional layers and early fully connected layers of auxiliary paths can use the corresponding sensory inputs to get good classification performances, it might be a more effective regulator to share weights with corresponding layers in the main model to "internally" force the "FC-con" layer to generate consistent fusion weights, to avoid the information mixed-up inside the "FC-con" layer , instead of externally forcing the main model to get further trained by competitors mentioned in last section.

The weight sharing model improves the qualities of fusion weights as well as the prediction performances. More detailed results will be shown in section Evaluation.

3.4 Fusion Weights Regularization

Even though the ARGate-WS architecture shows improvements in performances under some sensor failure cases, we observed that there are still cases when one or multiple sensors are corrupted, weight sharing fails to provide the improvements. The reason lies in the fact that auxiliary path model with corrupted inputs are not capable of providing the main model with positive information to boost the performance. Therefore, we further explore the additional regularization of auxiliary paths over main model, details can be seen in Fig. 17.

The idea of this architecture is a technique called Fusion Weight Regularization(FWR). The basic idea is that the loss functions of auxiliary paths show the same trend to qualities of sensory inputs as the fusion weights do. As in Fig. 17, if Sensor 1 fails with inputs of Gaussian noise while Sensor 2 is clean, the value of Loss_aux1 will be much larger than that of Loss_aux2. If we introduce an inverse

function of Loss_aux, the output value will have the same relevance of sensor quality as fusion weights do. Therefore we design the loss function below to use losses of auxiliary paths to regularize fusion weights of main model:

$$\text{Loss}_{total} = \alpha \cdot Loss_{main} + \beta \cdot \sum_{k=1}^{K} Loss_{aux}^{k} + \sum_{k=1}^{K} \left(w_{fusion}^{k} - e^{-\widehat{Loss_{aux}^{k}}^{2}} \right)^{2} \quad (4),$$

where β is another user-specified weighting parameter, w_{fusion}^{k} represents the value of fusion weight of the k-th input sensor, which is also the k-th output of "FC-con" layer. The inverse function of auxiliary path loss is designed as $e^{-Loss_{aux}^{2}}$.

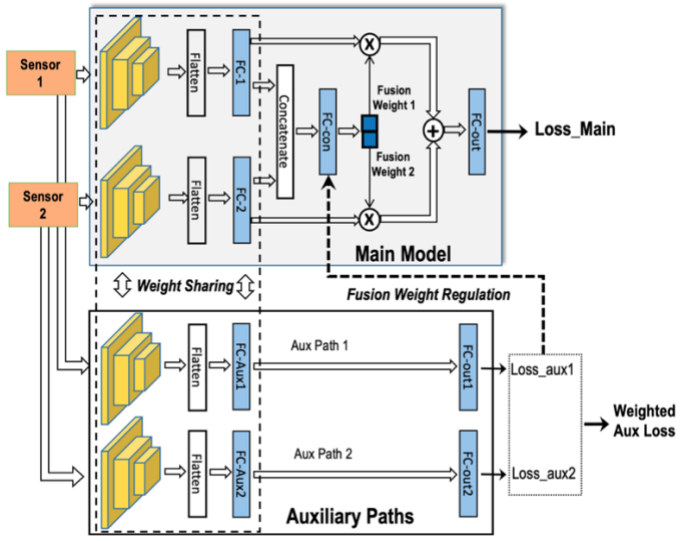

Figure 17. Auxiliary Regulated Gate with Weight Sharing and Fusion Weight Regularization.

Since the fusion weights is normalized using $L2$ norm and then softmax normalization, we implement the same normalization method on $e^{-Loss_{aux}^2}$.

The $\widehat{e^{-Loss_{aux}^k}}^2$ reprensents the normalized $e^{-Loss_{aux}^2}$. L2 normalize $e^{-Loss_{aux}^2}$ between [-1,1], *softmax* further normalize the value between [0,1]. We name the above architecture with both weight sharing and weight regularization as ARGate-WS-FWR.

ARGate-WS-FWR utilize auxiliary paths to regularize main model in two steps, first step is to use weight sharing(WS) in convolutional layers and early fully connected layers. Second step is to use loss functions of auxiliary paths to regularize the fusion weights of main model. Detailed experimental results will be shown.

3.5 Auxiliary Loss Weighting

Loss functions of auxiliary paths are in the total loss function of ARGate-WS and ARGate-WS-FWR. However, if one or multiple sensors corrupt, then their large loss functions may dominate other terms in total loss functions, the performances of network may be degraded because of this. In (Zhao, Shim, Li, Zhang, & Li, 2019) fusion weights are extracted from main model as shown in purple dashed arrow in Fig. 18 to perform the Auxiliary Loss Weighting(ALW) to auxiliary paths to constraint the loss functions of auxiliary paths. The new loss function is designed as:

$$\text{Loss}_{total} = \alpha \cdot Loss_{main} + \beta \cdot \sum_{k=1}^{K} w_{fusion}^k \cdot Loss_{aux}^k + \sum_{k=1}^{K} (w_{fusion}^k - \widehat{e^{-Loss_{aux}^k}}^2)^2 \quad (5)$$

where fusion weights of main model w_{fusion}^k are multiplied with auxiliary path loss $Loss_{aux}^k$. The idea is that if the $k-th$ sensor fails while the rest sensors are all

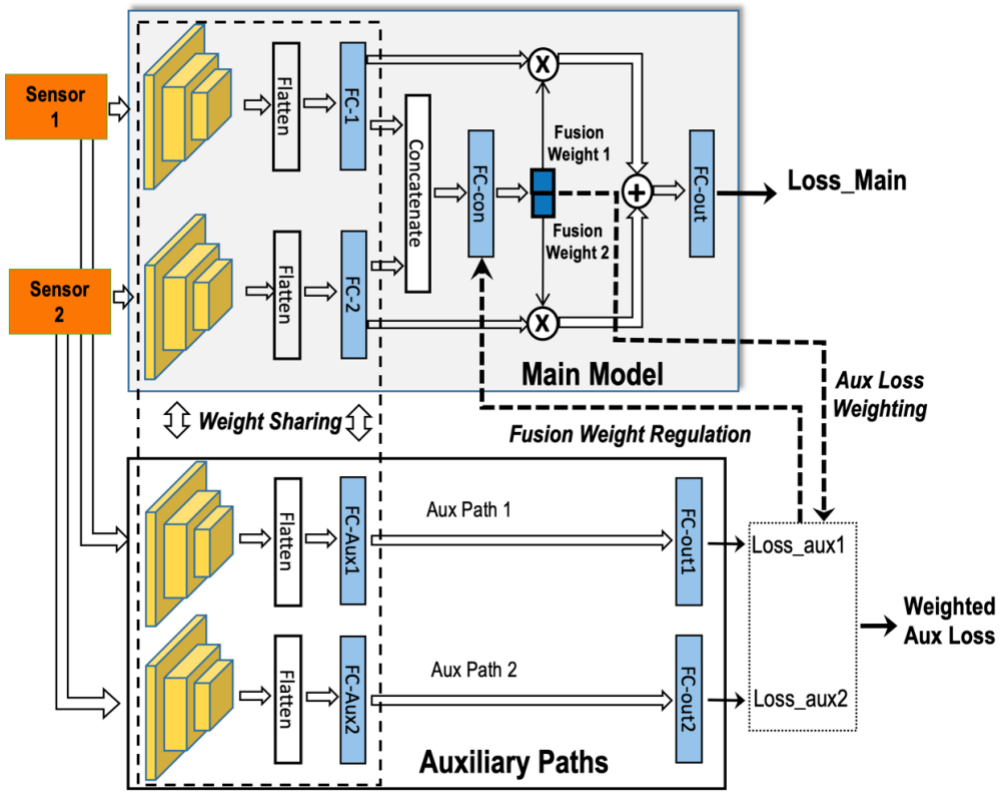

Figure 18. Full ARGate arthictecure (ARGate-F)

functioning, the pure noisy inputs would generate a large $Loss_{aux}^{k}$, which will dominate the total loss function Loss_{total}. The qualities of sensory inputs can be represented by fusion weights of main model, therefore applying w_{fusion}^{k} as a weighting term to multiply with $Loss_{aux}^{k}$ would be a solution to this issue.

3.6 Programmable Real-Time Unit

Two Programmable Real-Time Units (PRU) are embedded in BeagleBone Black, which is designed specifically for handling real-time computing.

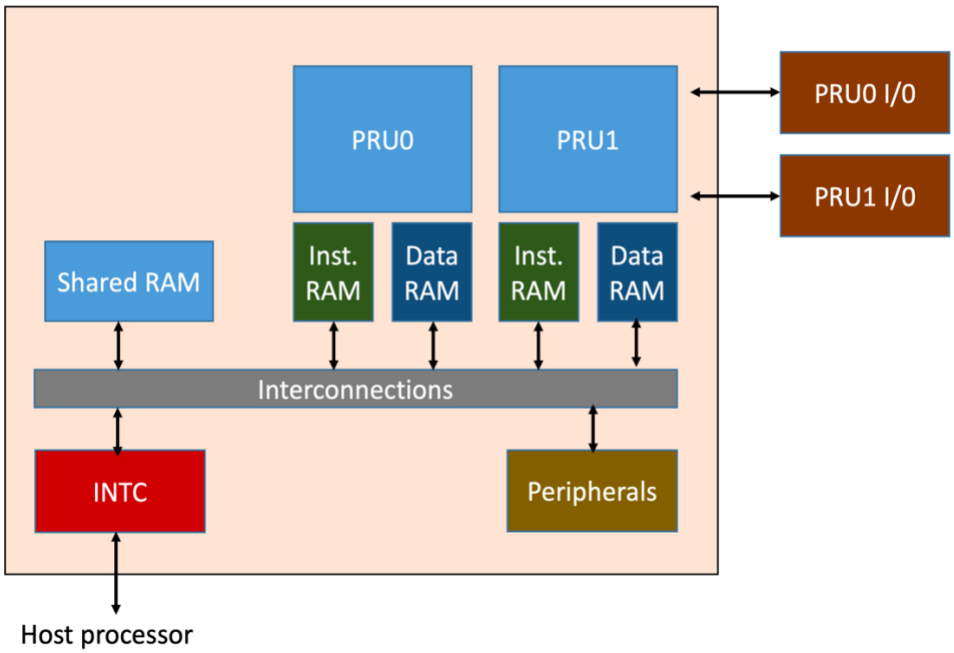

Figure 19. Programmable Real-Time Unit

Programmable real-time unit (PRU) is a fast (200-MHz, 32-bit) processor with the input/output accession of single-cycle instructions to a number of the pins, it also has the accession to the memory of the main processor on BeagleBone Black (AM3358).

Advantages of PRU can be concluded as follows:

- Based on RISC, no pipeline, no branch latency, most instructions can be finished within 1 clock cycle, making execution predictable, suitable for real-time processing.
- Runs in parallel with host AM335x processor.

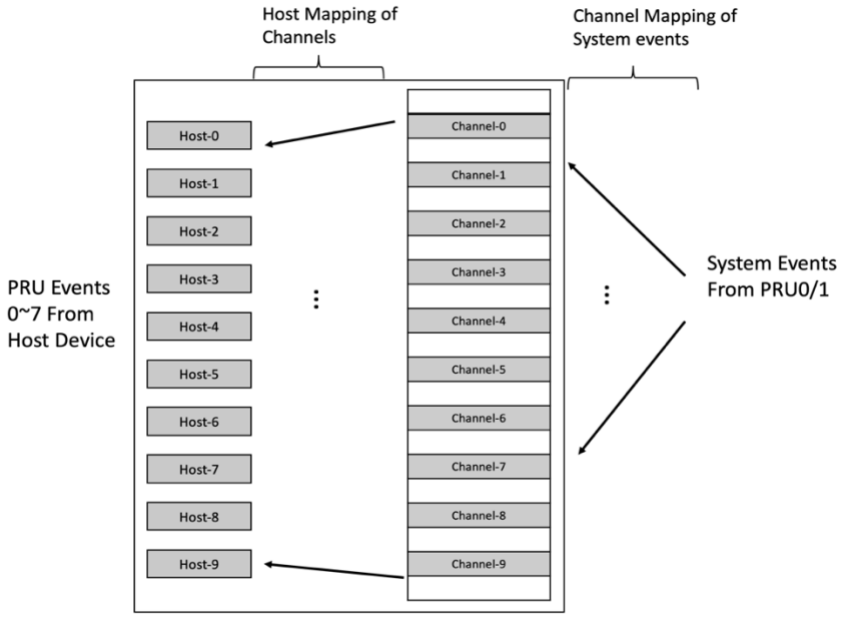

Figure 20. PRU interrupt controller (INTC)

Fig. 20 shows the architecture of interrupt controller of PRU. System Events 32 to 63 are generated by PRU0/1. We first connect 5 ultrasonic sensors to PRU0/1, then map the events of Echo pins of 5 ultrasonic sensors receiving the distance information to 5 system events between 32 to 63. Then the 5 system events are further mapped to 5 channels in PRU INTC. On the host processor, 5 interrupts are mapped to 5 PRU events on host board through Host Mapping of Channels. Through this two-step mapping, we can map events generated on two PRUs to the PRU events recognized by the host processor. Since distance detection program on PRU and control program on host processor are running in parallel. Our control program can read the distance information in real-time.

We implement the assembly code according to PRU assembly instruction set to directly get control of the reading and writing of registers and to generate interrupts. For each ultrasonic sensor, we first activate Trigger signal, and send out the sonic burst. Whenever the Echo register receives the data, save the data into memory of PRUs, and an interrupt is generated. Same steps for 5 ultrasonic sensors are done in sequence.

Device Tree file defines the modes of pins which we are going to use on BeagleBone Black. For example:

0x030 0x07 /* P8_12 gpio1[12] GPIO44 out pulldown Mode: 7 */

Bit 0-2 - Mode. Bit 3 with a value of 1 represents disable Pulling, of 0 stands for enable Pulling. Bit 4 with a value of 1 means Pull up, of 0 stands for Pull down. Bit 5 with a value of 1 means Input, 0 means Output

Control file running on host processor calls functions to initialize PRU, open PRU, PRU events mapping, memory mapping, and wait for interrupts from PRU. Pypruss library is adopted to call C functions in python codes.

After applying PRU to handle real-time distance information computing, the latency problem is solved and the robot can successfully stop and adjust its heading to follow the obstacles.

4. EXPERIMENTAL SETTINGS

4.1 Basic Settings

We compare the performances of fusion CNN baseline (Chen, Ma, Wan, Li, & Xia, 2017), the baseline gating architecture: NetGated (Patel, Choromanska, Krishnamurthy, & Khorrami, 2017) and the proposed auxiliary-model ARGate architectures. All architectures are evaluated based on two public datasets for human activity recognition: HAR (Anguita, Ghio, Oneto, Parra, & Reyes Ortiz, 2013) and CAD-60 (Sung, Ponce, Selman, & Saxena, 2012). We implement ADAM optimizer, the value of learning rate is 0.001. The loss functions of baseline, NetGated, ARGate is cross-entropy. Simulation are performed on Ubuntu 16.04, programming language is selected as Python 2.7 based on Pytorch 0.4.0 (Paszke, et al., 2017) , programs are simulated on NVIDIA TITAN Xp GPUs.

4.2 Datasets

Human Activity Recognition(HAR) Dataset. The Human Activity Recognition(HAR) dataset (Anguita, Ghio, Oneto, Parra, & Reyes Ortiz, 2013) includes data captured by an accelerometer and a gyroscope sensor embedded on smartphones, which are carried out on 30 volunteers. Data are gathered by the sensors at a rate of 50Hz, 50% overlapped sliding windows with 128 readings which split the raw data into different examples. Labels include six human activities: Walking, Walking Upstairs, Walking Downstairs, Sitting, Standing, Laying. Data from each type of sensor are captured in forms of 3-axial. To be more specifically, the dataset includes 3-axial total

acceleration (total_acc_x, total_acc_y, total_acc_z) data, 3 -axial body acceleration (body_acc_x, body_acc_y, body_acc_z) data and 3 -axial body gyroscope data (body_gyro_x, body_gyro_y, body_gyro_z). Where body acceleration signals are obtained by subtracting gravity acceleration from the total acceleration.

In the implementation of this work, the nine data are defined as nine sensory inputs. Sensory inputs for all 9 channels are distributed between [-1, 1]. There are 7352 training examples and 2947 testing examples.

The CAD-60 Dataset. In (Sung, Ponce, Selman, & Saxena, 2012) the dataset of CAD-60 is introduced, which contains 60 videos of human activities captured by Kinect device from Microsoft. The Kinect is equipped with a RGB camera and a depth sensor. Videos record 14 human activities of 4 people. Each video is preprocessed into RGB images, skeletal data and Depth images.

A preprocessing code is provided by the dataset to extract 5 features from raw RGB images, skeletal data and Depth images, including skeletal HOG of the depth image, skeletal, skeletal RGB HOG, and depth HOG. 5 extracted features are treated as 5 sensory inputs. The “new person” scheme in (Sung, Ponce, Selman, & Saxena, 2012) is adopted in our implementation which uses the three people’s data for training and the rest people’s data for testing.

4.3 Network Configurations

Training for all models in HAR dataset takes 200 epochs, the batch size is 16. Under CAD-60 dataset, training takes 100 epochs, batch size is 128.

CNN Baseline in HAR Dataset. Before fusion, CNN baseline has following layers to extract features from each sensory input before fusion: $\mathrm{C}(16,3,1) - \mathrm{P} - \mathrm{FC}(256) - \mathrm{FC}(256)$, here $\mathrm{C}(\mathrm{n},\mathrm{f},\mathrm{s})$ is a 1-D convolutional layer, the layer has n filters with sizes of f, the convolution stride is s. P stands for a max-pooling layer, the sliding window for pooling is non-overlapped with the size of 2. $\mathrm{FC}(\mathrm{n})$ stands for a FC layer. ReLU is selected as the activation function. After sensory inputs of all 9 channels are processed, the 9 outputs are fused through element-wise mean. Fused data are then fed into three additional FC layers: $\mathrm{FC}(128) - \mathrm{FC}(64) - \mathrm{FC}(6)$. Where the classification decision is made by the last $\mathrm{FC}(6)$ layer with a $softmax$ classifier.

NetGated in HAR Dataset. In order to compare with CNN baseline in the number of tunable parameters fairly, each sensory inputs is processed through a simpler structure in NetGated model: $\mathrm{C}(16,3,1) - \mathrm{P} - \mathrm{FC}(256)$. As shown in Fig. 13, the extracted features from 9 FC layers are first concatenated and then processed through a structure of $\mathrm{FC}(256) - \mathrm{FC}(9)$. $\mathrm{FC}(9)$ generates fusion weights of features of all channels. Before fed into next layer, fusion weights are normalized in order to match the corresponding physical meaning. The normalized fusion weights are used to weight the outputs of the last early FC layers: $\mathrm{FC}(256)$ to perform a weighted sum to get the fused data. The fused data are processed with two additinal FC layers: $\mathrm{FC}(256) - \mathrm{FC}(6)$.

ARGate in HAR Dataset. As in Fig. 17, convolutional and FC layers for auxiliary paths are added. Weight sharing is performed between each convolutional and early FC layer in auxiliary path and corresponding convolutional and FC layers in the main model. The output features of these FC layers are fed into another $\mathrm{FC}(6)$ layer to generate the classification decision of auxiliary paths. The outputs of the last early $\mathrm{FC}(256)$ layer in the main model are concatenated and processed through FC layers $\mathrm{FC}(256) - \mathrm{FC}(9)$. After the weighted sum, different from NetGated, the fused feature is processed through only one $\mathrm{FC}(6)$ layer to get the classification decision for fair comparison in terms of number of parameters.

CNN Baseline in CAD-60 Dataset. In CAD-60 dataset, before fusion, sensory inputs of skeletal feature channels are processed through three convolutional layers and one FC layer. While for skeletal HOG on RGB, input features are processed through three convolutional layers with different sized and one FC layer (FC-600) . Same number of convolutional layers with different sizes and a FC(600) for skeletal HOG on depth, RGB HOG, and depth HOG. Since sensory inputs for RBG HOG and depth HOG are small in sizes, therefore no max-pooling layers are employed on those two channels. The output features from last five $\mathrm{FC}(600)$ layers are fused through element-wise mean, the fused features are further processed by four fully connected layers to get the classification decision.

NetGated in CAD-60 Dataset. Same early feature extraction schemes are implemented in NetGated as in CNN Baseline. The outputs of last early $\mathrm{FC}(600)$ are first concatenated and then fed into $\mathrm{FC}(3000)$ and a $\mathrm{FC}(5)$ to extract the normalized

fusion weights of five features. The fused data are calculated through weighted sum of extracted features with their corresponding fusion weights. The fused data are further processed through $\mathrm{FC}(1200)$ and a $\mathrm{FC}(14)$ for the classification.

ARGate in CAD-60 Dataset. Same setups are implemented in early feature extraction before fusion in main model of ARGate as in baseline CNN and NetGated. As in NetGated, five extracted features are concatenated and fed into $\mathrm{FC}(3000)$ and $\mathrm{FC}(5)$ to extract fusion weights. After normalization of fusion weights, features are again fused through weight sum of output features from five last early FC layers, then fused feature is processed through $\mathrm{FC}(200) - \mathrm{FC}(14)$. $\mathrm{FC}(14)$ generates the classification decision. For early feature extraction layers, weight sharing is performed between auxiliary paths and corresponding layers in main model. The output features of last early FC layers are fed directly into $\mathrm{FC}(200) - \mathrm{FC}(14)$ for final outputs of auxiliary paths, which are introduced to regularize fusion weights.

The total number of tunable parameters in baseline CNN, NetGated and ARGate are 2619136, 2594236, 2594236, respectively. In ARGate, α is set as 5.0 in (3), and α, β in (4) are set to be $\frac{1}{9}, \frac{5}{9}$, respectively.

4.4 Fusion Weight Normalization

In (Patel, Choromanska, Krishnamurthy, & Khorrami, 2017), after fusion weights are generated from "FC-con" layer as shown in Fig.12, there's no normalization step for fusion weights. In our implementation, we normalize fusion weights for both NetGated and proposed ARGate models in order to enable fusion weights to have the physical meaning of weighting parameter. Normalization steps consist of 3 steps. Firstly, fusion

weights w_{fusion} are normalized by L2 to be distributed between [-1, 1]. Secondly, in order to enlarge the differences between small fusion weights which represent low qualities of corresponding sensory inputs and large fusion weights, we perform:

$$w'_{fusion} = \left(l^2_{norm}\left(w_{fusion}\right) + 1\right) * 2 \quad (6),$$

where fusion weights normalized by L2 norm are first transformed to be distributed between [0, 2], and then stretch to be between [0, 4]. Thirdly, in order normalize fusion weights to be distributed between [0,1] and have a sum of 1, we perform *softmax* normalization:

$$w_{fusion,n} = softmax(w'_{fusion}) \quad (7),$$

where w'_{fusion} represent the normalized results from step 2. Through the 3-step normalization, each scalar fusion weight is capable of interpreting the quality or importance of corresponding sensory features as a weighting parameter.

4.5 Sensor Failures

In this section we design sensor failure schemes on training and testing for HAR and CAD-60 datasets, trying to compare the robustness of CNN baseline, NetGated and proposed ARGate architecture in comprehensive failure schemes.

Values of sensory input data in two datasets are distributed between [-1, 1]. In order to simulate the inputs when sensors corrupt and produce no information but pure noise, we consider three types of noise distributions: *zero*, *uniform* and *Gaussian.* More specifically, inputs of failing sensors are modeled by generate sensory inputs with values of zero, noise following a uniform distribution $\mathrm{U}(-1,1)$ and Gaussian distribution $\mathrm{N}(0,1)$.

In training and testing sets of both datasets, $\frac{1}{3}$ data of each set are randomly selected as clean, while the rest $\frac{2}{3}$ are set as corrupted by one or multiple sensors based on one of the failing schemes below:

Fixed failing assignment. Fixed failing assignment simulates the situations when certain one or multiple sensors fail permanently. We define n_{fclean} as the number of clean channels among total number of n sensors. The rest sensors are assumed to have permanent failure in both training and testing sets.

Random failing assignment. Instead of fixing the corrupted sensors in fixed failing assignment, random failing assignment randomly select failing sensors to better simulate the random sensor failure cases in reality. For each example, we randomly select n_{rclean} channels as clean channels, while the reset $\mathrm{n} - n_{rclean}$ channels are defined as corrupted. Same implementations are performed in both training and testing sets, where sensor failure situations may vary example by example.

Test generalized failing assignment. Since sensor failures may have various cases which cannot be all included in training dataset, thus in reality we need to consider cases that the neural networks haven't met before. Therefore we design the sensor failure assignment when test sets contain corrupted examples with a different number of corrupted sensors comparing with the number of failing examples in training sets. Which also means that examples in test set has higher variations on sensor failures than those in training set.

5. EVALUATIONS

5.1 Distribution of Fusion Weight

In order to further study the functioning of weight sharing(WS), fusion weight regularization(FWR), we examine the fusion weight distribution in NetGated, ARGate-WS and ARGate-WS-FWR. We picked distribution of fusion weight of total_acc_y in HAR dataset under random failing assignment when n_{rclean}=1, which means eight out of nine sensory inputs are randomly selected to be corrupted in each example. To better demonstrate the effects of proposed architecture, examples are split into two sets: one with corrupted total_acc_y, the other set includes examples when inputs on total_acc_y are clean. As in Fig. 21, (a), (b) and (c) display the fusion weight distributions of the first subset for NetGated, ARGate-WS and ARGate-WS-FWR, respectively. While Fig. 21(d),(e) and (f) display the distribution of fusion weights for examples in the second situation mentioned above.

From the aspects of physical meaning of fusion weights, if the model is properly trained, the fusion weight values of a clean sensory inputs should be much greater than those of failing sensory inputs. In Fig. 21(a), (b) and (c), when total_acc_y is corrupted, a peak is shown around a value of 0.38 in the fusion weight distribution of NetGated, which is not shown in ARGate-WS and ARGate-WS-FWR. ARGate-WS significantly reduce the most large fusion weight values comparing with NetGated. Moreover, comparing with Fig. 21(b) where there are still some fusion weights with values larger than 0.38, most fusion weights are constrained within a small range around 0.06 to 0.08

in Fig. 21(c), which also demonstrates the effectiveness of fusion weight regularization(FWR). For clean examples, one can expect that values of fusion weights of total_acc_y should be larger for the clean examples than the corrupted examples as in first subset.

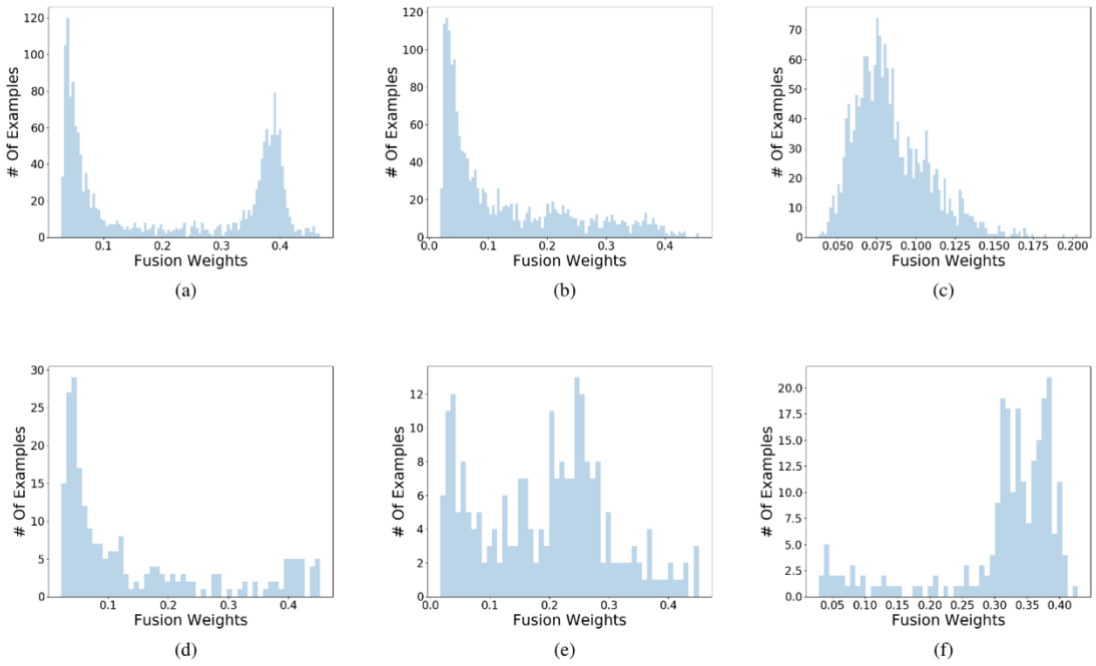

Figure 21. Distributions of fusion weights of input examples on channel total_acc_y based on NetGated, ARGate-WS and ARGate-WS-FWR, failing scheme is random failing sensor assignment, n_{rclean} is set as 1. (a), (b) and (c) demonstrate the distributions of fusion weights for corrupted sensory inputs on channel total_acc_y based on NetGated, ARGate-WS and ARGate-WS-FWR. (d), (e) and (f) are the fusion weight distributions of clean sensory inputs on total_acc_y.

As shown in Fig. 21(d), the distribution of fusion weights in NetGated has a peak around a small value of 0.05, which takes a large proportion of total fusion weights. While in Fig. 21(e), the percentage of fusion weights with very low values reduced by ARGate-WS, another peak between 0.2 and 0.3 appears. In Fig. 21(f), ARGate-WS-FWR further reduced the second peak in Fig. 21(e).

With this random failing sensor assignment when n_{rclean}=1, the NetGated, ARGate-WS and ARGate-WS-FWR have the prediction accuracies of 62.90%, 65.69% and 66.09%, respectively. Combining with distributions of fusion weight in Fig. 20, weight sharing(WS) learns a more robust fusion weights than NetGated, and fusion weight regularization(FWR) further improves the qualities of fusion weights, hence showing the best performance.

5.2 Results on the HAR Dataset

Fixed Failing Assignment. Fixed falling assignment proposed in last section is adopted here to evaluate the performances of CNN Baseline, NetGated and proposed ARGate-WS-FWR. We perform two cases: $n_{fclean} = 5$, when $body\_total\_acc\_x$, $body\_acc\_x$ and $body\_gyro\_x$ are set to be corrupted sensors, and $n_{fclean} = 6$, when $body\_acc\_z$ and $body\_gyro\_x$ fail. All failing sensors have inputs following uniform distribution between -1 and 1.

Table. 1 shows the performances of three models under two fixed failing assignments in HAR dataset. NetGated shows a improvements over Baseline CNN up to 1.6%, and ARGate-WS-FWR always shows the best performance and can further improve the performances of NetGated up to 2.37%.

Table 1. Prediction accuracies under HAR dataset with fixed failing assignment.

Number of Clean Channels	Baseline	NetGated	ARGate-WS-FWR
$\boldsymbol{n_{fclean} = 5}$	87.68%	89.28%	90.97%
$\boldsymbol{n_{fclean} = 6}$	80.59%	81.94%	84.31%

Table 2. Prediction accuracies under HAR dataset with clean and random failing sensor assignment.

Number of Clean Channels	Failure Model	Baseline	NetGated	ARGate-WS	ARGate-WS-FWR
All Clean	-	94.06%	94.50%	94.96%	95.09%
$n_{rclean} = 8$	Zero	93.02%	93.17%	94.66%	94.04%
	Uniform	92.35%	92.20%	92.45%	92.46%
	Gaussian	92.94%	93.28%	94.97%	94.35%
$n_{rclean} = 5$	Zero	88.36%	87.95%	88.63%	88.83%
	Uniform	86.73%	86.80%	88.53%	89.17%
	Gaussian	88.41%	89.04%	89.52%	90.07%
$n_{rclean} = 1$	Zero	**71.56%**	**71.12%**	**74.38%**	**74.44%**
	Uniform	**62.06%**	**62.90%**	**65.69%**	**66.09%**
	Gaussian	**69.67%**	**70.54%**	**71.83%**	**72.58%**

Random Failing Assignment. In order to evaluate models under random sensor assignments, we create comprehensive sensor failure situations with combinations of the

number of clean channels and failure models. We randomly choose clean sensors with the number of $n_{rclean} \in \{1, 5, 8\}$.

Three failure models are being considered: corrupted sensory inputs with zero values, corrupted inputs following uniform distribution between -1 and 1, and Gaussian distribution $\mathrm{N}(0,1)$. 4 models are implemented and compared: baseline CNN, NetGated, proposed ARGate-WS and ARGate-WS-FWR. When all nine sensors are clean, NetGate shows an improvement on prediction accuracy of 0.44% over the baseline CNN, while improvements over baseline generated by ARGate-WS and ARGate-WS-FWR are 0.9%, 1.03%, respectively. ARGate-WS-FWR always shows improvements over baseline and NetGated, and is generally outperforming ARGate-WS, which demonstrates the effectiveness of weight sharing(WS) and fusion weight regularization(FWR). Particularly when $n_{rclean} = 1$, with Uniform failure model, ARGate-WS-FWR outperforms the baseline, NetGated, ARGate-WS by 4.03%, 3.19%, and 0.4%, respectively.

Testing Generalized Failing Sensor Assignment. We implement failing sensor assignment for model generalization as in Table. 3. In first column, $(\mathrm{a}, \mathrm{b})(\mathrm{c}, \mathrm{d})$ represents that the number of failing channels in training dataset in each example is randomly selected from [a, b], while the range of the number of corrupted channels in testing set is between [c, d]. With this set-up, we aim to simulate situations when sensor failure scenario in test set is more complicated than those in training set, and even situations when cases in testing set are completely different from those in training set.

Table 3. Prediction accuracies under HAR dataset with testing generalized failing sensor assignment.

Number of Failing Channels	Baseline	NetGated	ARGate-WS-FWR
(1,2)(3,8)	72.91%	72.75%	76.87%
(1,3)(4,8)	70.98%	70.78%	75.09%
(1,4)(5,8)	69.38%	69.53%	72.41%

In Table. 3, when the number of failing channels is (1, 2)(3, 8) and (1, 3)(4, 8), NetGated even performs worse than baseline, but ARGate-WS-FWR always shows the best performance, generating improvements up to 4.11% and 4.31% over Baseline and NetGated, respectively.

5.3 Results on the CAD-60 Dataset

Fixed Failing Sensor Assignment. In CAD-60 dataset, we consider cases when n_{fclean} equals 1 and 4, respectively. When $n_{fclean} = 1$, RGB HOG, skeletal features, Depth HOG and skeletal HOG features on Depth Image are simulated to be corrupted sensors. When $n_{fclean} = 4$, the channel skeletal channel is fixed as corrupted channel. As shown in Table. 4, NetGated performs the worst, but ARGate-WS-FWR always shows the best performance.

Table 4. Prediction accuracies under CAD-60 dataset with fixed failing sensor assignment.

Number of Clean Channels	Baseline	NetGated	ARGate-WS-FWR
$\boldsymbol{n_{fclean}} = 1$	60.60%	59.98%	65.09%
$\boldsymbol{n_{fclean}} = 4$	64.15%	61.72%	72.34%

Random Failing Sensor Assignment. Here the number of randomly failing sensors $n_{rclean} \epsilon \{1, 2, 3, 4\}$. We also compare the baseline, NetGated, the proposed ARGate-WS, ARGate-WS-FWR. As can be seen in Table. 5, when all input sensors are clean, ARGate-WS-FWR improves the performance of baseline, NetGated, ARGate-WS by 0.36%, 0.8% and 0.22%, respectively. As for other failure scenarios, ARGate-WS-FWR generally shows the best performance. NetGated performs worse than baseline, ARGate-WS and ARGate-WS-FWR improve the performance over baseline and NetGated in most cases. The largest improvement appears when $n_{rclean} = 4$. ARGate-WS-FWR outperforms the baseline and NetGated by 9.05% and 13.68%, respectively.

Table 5. Prediction accuracies under CAD-60 dataset with clean and random failing sensor assignment.

Number of Clean Channels	Failure Model	Baseline	NetGated	ARGate-WS	ARGate-WS-FWR
All Clean	-	87.01%	86.57%	87.15%	87.37%
$n_{rclean} = 4$	Zero	**71.91%**	**68.87%**	**75.01%**	**78.67%**
	Uniform	**69.76%**	**65.13%**	**75.07%**	**78.81%**
	Gaussian	**71.55%**	**73.81%**	**73.91%**	**75.74%**
$n_{rclean} = 3$	Zero	72.91%	65.48%	71.93%	71.47%
	Uniform	69.38%	67.61%	72.58%	71.96%
	Gaussian	88.41%	89.04%	89.52%	90.07%
$n_{rclean} = 2$	Zero	67.94%	67.98%	65.18%	64.41%
	Uniform	64.98%	62.98%	66.07%	66.59%
	Gaussian	67.41%	66.55%	64.62%	66.96%
$n_{rclean} = 1$	Zero	59.07%	28.18%	57.43%	59.89%
	Uniform	61.42%	57.10%	60.10%	61.35%
	Gaussian	57.44%	57.75%	57.39%	57.55%

Testing generalized failing sensor assignment. We also generate the simulations for failing sensor assignment for model generalization, as shown in Table. 6.

As shown in Table. 6, NetGated model in most cases underperforms the baseline model, while ARGate-WS-FWR always shows the best performance.

Table 6. Prediction accuracies under CAD-60 dataset with testing generalized failing sensor assignment.

Number of Failing Channels	Baseline	NetGated	ARGate-WS-FWR
(1,4)(2,4)	64.08%	63.71%	67.86%
(2,3)(2,8)	55.36%	55.16%	58.01%
(2,4)(1,4)	60.77%	61.05%	62.04%

The results in Table. 6 shows that proposed ARGate model shows the best generalization ability under CAD-60 dataset.

6. CONCLUSION

In this work, we first implemented the hardware platform for sensor fusion, the robot car, and solved the real-time distance detection problem through PRU. On the algorithm design side, we clarified the limitations of traditional fusion methods as well as gating architectures. We proposed the more robust ARGate architecture together with two regularization techniques. The experimental results on two public dataset show the significant improvements on prediction accuracy, especially under sensor failure schemes. Our future work will be on the application of ARGate architecture to sensor fusion of more complicated sensory inputs. A camera will be implemented on the robot car in order to facilitate a more comprehensive sensor fusion network on it.

REFERENCES

Anguita, D., Ghio, A., Oneto, L., Parra, X., & Reyes Ortiz, J. L. (2013). A public domain dataset for human activity recognition using smartphones. *ESANN*.

Bohez, S., Verbelen, T., De Coninck, E., Vankeirsbilck, B., Simoens, P., & Dhoedt, B. (2017). Sensor fusion for robot con- trol through deep reinforcement learning. *In Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, (pp. 2365–2370).

Chen, X., Ma, H., Wan, J., Li, B., & Xia, T. (2017). Multi-view 3d object detection network for autonomous driving. *CVPR* (pp. volume 1, pp. 3). IEEE.

Dehzangi, O., Taherisadr, M., & ChangalVala, R. (2017). Imu-based gait recognition using convolutional neural networks and multi-sensor fusion. *Sensors*, 17(12):2735.

Garcia, F., Martin, D., De La Escalera, A., & Armingol, J. M. (2017). Sensor fusion methodology for vehicle detection. *IEEE Intelligent Transportation Systems Magazine*, 9(1): 123–133.

Geiger, Andreas, Lenz, P., & Urtasun., R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE.

Girshick, R. (2015). Fast r-cnn. *Proceedings of the IEEE international conference on computer vision.*

Girshick, R. a. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition.*

Gravina, R., Alinia, P., Ghasemzadeh, H., & Fortino, G. (2017). Multi-sensor fusion in body sensor networks: State-of- the-art and research challenges. *Information Fusion*, 35: 68–80.

Jain, A., Koppula, H. S., Soh, S., Raghavan, B., Singh, A., & Saxena, A. (2016). Brain4cars: Car that knows before you do via sensory-fusion deep learning architecture. *arXiv preprint arXiv*, 1601.00740.

K, H., X, Z., & S, R. (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. *European conference on computer vision.* Cham: Springer.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. *In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, (pp. 1725–1732).

Kim, J., Koh, J., Kim, Y., Choi, J., Hwang, Y., & Choi, J. W. (2018). Robust deep multi-modal learning based on gated information fusion network. *arXiv preprint arXiv*, 1807.06233.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems*, 1097-1105.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. . *The handbook of brain theory and neural networks*, 3361(10).

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (Springer). Ssd: Single shot multibox detector. *In European conference on computer vision*, 21-37.

Mees, O., Eitel, A., & Burgard, W. (2016). Choosing smartly: Adaptive multimodal fusion for object detection in changing environments. *In Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on* (pp. 151–156). IEEE.

Ordonez, F. J., & Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., . . . Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS-W.*

Patel, N., Choromanska, A., Krishnamurthy, P., & Khorrami, F. (2017). Sensor modality fusion with cnns for ugv au- tonomous driving in indoor environments. *In International Conference on Intelligent Robots and Systems (IROS).* IEEE.

Ramachandram, D., & Taylor, G. W. (2017). Deep multimodal learning: A survey on recent advances and trends. *IEEE Signal Processing Magazine*, 34(6):96–108.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition.*, (pp. 779-788).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *In Advances in neural information processing systems* , 91-99.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv*, 1409.1556.

Sung, J., Ponce, C., Selman, B., & Saxena, A. (2012). Unstructured human activity detection from rgbd images. *In Robotics and Automation (ICRA)* (pp. 842-849). 2012 IEEE International Conference on.

Wei, P., Cagle, L., Reza, T., Ball, J., & Gafford, J. (2018). Lidar and camera detection fusion in a real-time industrial multi-sensor collision avoidance system. . *Electronics*, 7(6):84.

Yurtman, A., & Barshan, B. (2017). Activity recognition invariant to sensor orientation with wearable motion sensors. *Sensors*, 17(8):1838.

Zhao, C., Shim, M. S., Li, Y., Zhang, X., & Li, P. (2019). Deep Neural Networks with Auxiliary-Model Regulated Gating for Resilient Multi-Modal Sensor Fusion. arXiv preprint arXiv:1901.10610 (2019).

Zhao, Y., & Zhou, S. (2017). Wearable device-based gait recog- nition using angle embedded gait dynamic images and a convolutional neural network. *Sensors*, 17(3):478.