

Application of Convolution Neural Network in Design and Fabrication of Robots for Transporting Goods in Factories

Nguyen Hoang Dung

Hanoi University of Science and Technology – No.1, Dai Co Viet str., Hai Ba Trung dist., Hanoi, Vietnam

Received: August 10, 2019; Accepted: November 12, 2020

Abstract

Nowadays, the use of freight robots in factories will help people reduce their labor force and move into difficult and dangerous places more easily. Only smart robots will help people move equipment, goods to destinations that have been designed to own lanes in factories or remotely control these cargo robots to move following the demands of the controller. Goods will be delivered to the right place, helping to reduce labor costs for factories, increase productivity, thereby increasing the profits of businesses. Understanding the necessity of the design of transport robot and the development of artificial intelligence field along with the development of some types of embedded computers, the research team proposed a method to use convolutional neural networks deployed on the embedded computer platform to design a smart robot model to transport goods in the factory.

Keywords: AI, deep learning, CNN, robot.

1. Introduction

In order to meet the increasingly popular demand for industrial automation, robots, especially robots for transporting goods in factories help businesses optimize production lines, ensure productivity, and secure the fastest transport time.

Cargo transport robots in factories are self-propelled robot products capable of navigating under human programming or moving under human control. In order to transport goods to the right place, these robots will be programmed to follow the markers available with tape, wire on the floor, or use laser navigation to move. Currently, cargo transport robot models have been applied in many factories, airports, warehouses, wharves, etc. Here are the advantages of using the transport robot: (1) Ensuring productivity even without workers; (2) Providing safe, effective and cost-effective transportation solutions for businesses and (3) Increasing productivity for the supply chain.

In the process of developing a transport robot in a factory, software plays an equally important role as hardware. Software is like the brain, ensuring the robot's operation. In the past, freight vehicles were characterized by engines, gearboxes, actuators, steering wheel, gasoline, etc, but today, a robot is like a computer that can replace many factors such as people, mechanics, fuel, etc.

Combining the development of artificial intelligence, machine learning, and deep learning with the development of camera technology, technology research on smart transport robots focuses on two main areas: lane detection and object detection.

- *Lane detection:* This issue has been studied for many decades. Most lane detection systems have been developed and used in a wide variety of vehicles and robots. In the cargo transport robot control application, lane detection helps the robot determine the way to the point where the cargo needs to be transported.

- *Object detection:* This is an important component of the transport robot system. Recently, technology has made great progress, in addition to detecting static objects, it is now possible to detect dynamic objects as well. In the transport self-propelled robot control application, object detection is used in many cases, for example detecting obstacles for the robot to automatically avoid or detecting the hand and moving the robot according to the operator's hand controller so that it can move when people cannot move.

Based on the advantages of applying convolutional neural network (CNN) models and researching available documents, the research team has proposed to use convolutional neural network (CNN) models for lane detection and object detection of transport robots in factories. The design, fabrication

*Corresponding author: Tel: (+84) 913004120
Email: dung.nguyenhoang@hust.edu.vn

methodology and results achieved will be presented in the next sessions of this paper.

2. System design

2.1. System architecture

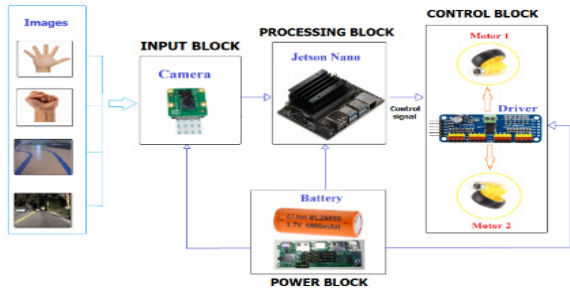


Fig. 1. System architecture

Fig.1 shows the design model of a cargo robot system in a factory. The system consists of 4 main components: (1) *the input block* is a camera integrated on the robot with the role of receiving image data to send to the processing block. In addition, this block also functions as the senses of the robot thereby helping the robot perceive its surroundings; (2) *the processing block* is responsible for collecting and processing images from the input block and giving control commands to the control unit. The processing block will be the main processing unit of the entire system. The trained deep learning models will also work in this block; (3) *The control block* consists of motors and motor control circuits. This block will receive control signals from the processing block to help the robot move according to the desired requirement; (4) *The power block* uses a rechargeable battery system as the main power supply for the whole system, through a voltage stabilizer to provide stable voltage and current. The specific composition and functions of the components contained in each block will be detailed as shown in Table 1.







2.2. Deep learning model for lane walking robot problem

2.2.1 Rotation angle

Assuming the car is moving in a straight direction with constant speed, consider the position of the car at time t_0 , if the car wants to change direction and move in time Δt , the new position will have a new straight direction created with initial direction of an angle α .

From there, the team selects the control signal for the vehicle which is the angle between the vehicle's direction of travel and the lane to determine the direction of movement for the vehicle. For the lane input image, this angle of rotation will be generated by the desired movement direction with the Ox axis.

Table 1. Functions of each component

Hardware component	Functions, duties
 <p>NVIDIA Jetson Nano Developer Kit</p>	The processing unit using the NVIDIA Jetson Nano Developer Kit is an embedded computer with only one compact circuit board. NVIDIA Jetson Nano Developer Kit uses Quad-core ARM® Cortex® - A57 MPCore processor, NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores 0.5 TFLOPs (FP16), 4GB 64-bit LPDDR4 1600MHz - 25.6 GB / s, HDMI 2.0, 802.11 b / g / n wireless LAN, Bluetooth 4.1, 40-pin GPIO pin. The corresponding voltage and current source is 5V / 3A. NVIDIA Jetson Nano Developer uses 128 CUDA GPU capable of handling artificial neural networks directly on the kit, giving out signals fast enough to control the vehicle.
 <p>Raspberry Pi Camera Module V2.0</p>	Input block uses Raspberry Pi Camera module v2.0 with 8 Megapixels resolution, 1080p30, 720p60, 640 x 480p60 / 90 recording mode, Sony IMX219 sensor, 3280 x 2464 pixels sensor resolution, pixel size is 1.12 μm x 1.12 μm, 3.04mm focal length, 62.2o horizontal field of view, 48.8o vertical field of view, CSI connection.
 <p>PCA9685 control circuit</p>	The control unit uses PCA9685 module with 16 channels, frequency 40 ~ 100Hz, 12bit PWM resolution, I2C communication, using 2.3 ~ 5.5VDC voltage.
 <p>Decelerated DC motor</p>	The control unit uses a decelerated DC motor with no-load speed of 125 rpm at 3V operating voltage (speed 26m / min), 208 rpm at operating voltage 5V (velocity 44m / min), no-load current 70mA, maximum current 250mA.
 <p>Li-ion battery BL26650</p>	Power unit uses rechargeable Li-ion BL26650 battery with voltage of 3.7v, capacity of 6800mAh, weight 90g, explosion-proof and anti-watering or blistering.
 <p>QC 3.0 charger and sTabilizer circuit</p>	Power unit uses 24W QC3.0 charging and voltage regulator circuit. Input: 6 - 32 V. Output: 5V / 3.4A - 9V / 2.5A - 12V / 2A

2.2.2 CNN architecture

The CNN [1] architecture that the team uses is covered in an article by NVIDIA [2] that revolves around autonomous vehicles. This is a CNN network that serves the regression problem with a single output which is the control signal for the vehicle with the architecture described in Fig.2. The network consists of 9 layers in which one normalization layer, five convolutional layers and three fully connected layer

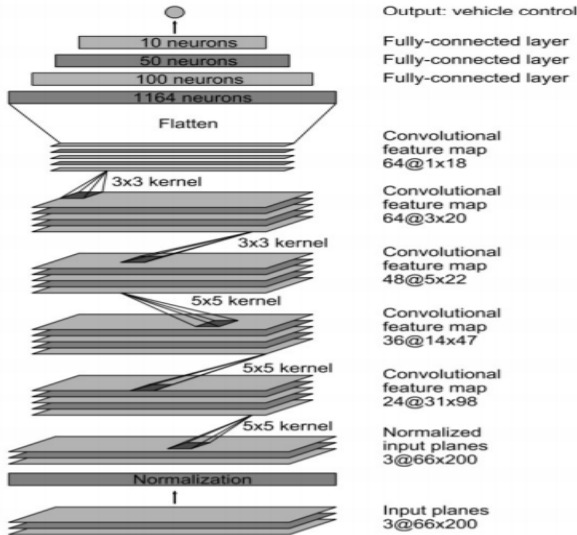


Fig. 2. Architecture of the model

The input image will be divided into YUV planes which are then transmitted to the network. The first layer of the network normalizes the image. Convolution classes are designed to extract the features and characteristics of the lane. The first three convolution layers use a 5x5 kernel matrix and the 2x2 sliding matrix and the other 2 convolution layers use only the 3x3 kernel matrix. As can be seen, this is a relatively small network with the total number of parameters after calculating is 252,219, but it still gives very good results as published by the paper.

2.2.3 Database

The collected data set included more than 5500 photos exported from 2 hours of driving in bright, sunny, and shady weather. In addition, approximately 300 lane images were recorded in the laboratory for the purpose of diversifying input data. Fig.3 illustrates some pictures in a custom dataset with lanes under lab conditions.

Each data set will be divided at the ratio of 7: 3 for training and validation. In addition, the test set will include 500 actual lane images and 30 laboratory lane images (corresponding to 10% of the total number of images in the dataset). The test set will be used to evaluate how accurate it is in practice with the test data.

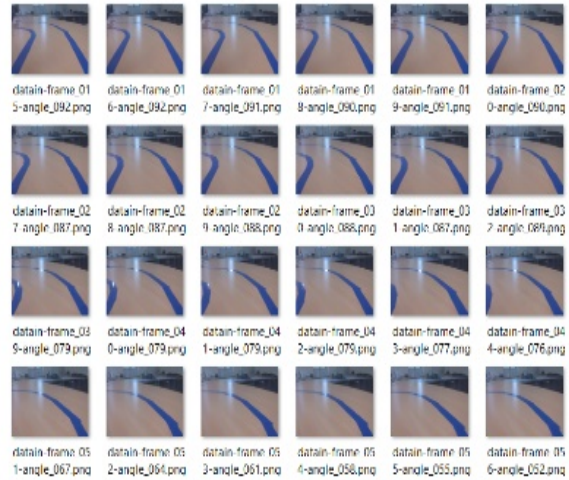


Fig. 3. Laboratory dataset

2.2.4 Training and model evaluation

The team uses the TensorFlow library together with the Keras application programming interface for training. The loss function of the model is determined by the Mean total Square Error - MSE with the formula (1) and the results are shown in Fig.4 with n : amount of data for testing; Y_i : real value of data i ; Y_i^p : predictive value for data i :

$$\rightarrow MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - Y_i^p)^2 \quad (1)$$

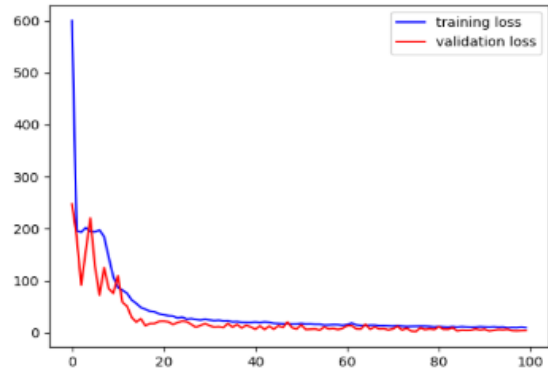


Fig. 4. Results of loss function after training

Input	Operator	T	c	n	s
224 ² x 3	conv2d	-	32	1	2
112 ² x 32	bottleneck	1	16	1	1
112 ² x 16	bottleneck	6	24	2	2
56 ² x 24	bottleneck	6	32	3	2
28 ² x 32	bottleneck	6	64	4	2
14 ² x 64	bottleneck	6	96	3	1
14 ² x 96	bottleneck	6	160	3	2
7 ² x 160	bottleneck	6	320	1	1
7 ² x 320	conv2d 1 x 1	-	1280	1	1
7 ² x 1280	avgpool 7 x 7	-	-	1	-
1 x 1 x 1280	conv2d 1 x 1	-	k	-	-

Fig. 5. Overall architecture of Mobilenet V2 [3]



Fig. 6. Attributes extracted from Mobilenet V2 networks

2.4 Deep learning model for the problem of controlling the robot moves according to the hand

This system selects MobileNet V2 as the feature extraction network and SSD as the object detection unit. The combination of these two networks will form a hand detection model with fast processing speed, low latency, good accuracy, suitable for deployment on NVIDIA Jetson Nano embedded computer for good performance. From there, it meets the requirements of the application transport robot control using the hand tracking system in the factory. The hand detection system consists of four main phases as (1) Preprocessing; (2) Feature Extractor; (3) Detection and (4) Postprocessing.

2.4.1 Feature Extractor with Mobilenet V2

For embedded systems, a lightweight and low-latency CNN network is required. The CNN network that fits that requirement is Mobilenet V2. Mobilenet V2 has a parameter count of only about 3.5M, resulting in faster processing speed and accuracy not inferior to conventional CNN networks.

The difference between Mobilenet V2 compared to conventional CNN networks is that it uses a separate depth convolution layer instead of the standard convolution layer. This is the reason why MobileNet V2 has much fewer parameters than the rest of CNN networks. In addition, the special feature of MobileNet V2 is that this network uses two structures: Linear Bottlenecks and Inverted Residual Structure to form the blocks in the model.

MobileNet V2's architecture consists of a full convolution layer with 32 filters, followed by 19 bottleneck layers depicted in Fig.5. Properties extracted from the Mobilenet V2 network are depicted in Fig. 6.

2.4.2 Object detection with Single Shot MultiBox Detector (SSD)

Fig.7 and Fig.8 respectively show the architecture and schematic diagram of how an SSD works when combined with Mobilenet V2. Single Shot MultiBox Detector (SSD) [4], all of the object detection and classifications, are done in the same network. The name of the model - Single Shot MultiBox Detector also tells us that the model uses many frames with different ratios to identify the object area and classify the object, reduce the step of creating area proposal network compared to Fast R-CNN, so the processing speed is increased many times while accuracy remains guaranteed.

In short, the SSD model will be a combination of 2 steps: (1) Extract feature maps from CNN network; (2) Apply a convolution filter to detect objects on feature maps with different resolutions (small objects will be detected in higher resolution feature maps, and vice versa for large objects). SSD is based on a forward propagation process of standard architecture, MobilenetV2 produces an output block of three-dimensional feature maps at an early stage. The MobilenetV2 network architecture here is a base network. We will then add the structures behind the base network to perform object recognition as part of the Extra Feature Layers in the diagram. These layers are intended to reduce the size of the feature map, thereby reducing the number of frames to be forecasted and allowing for predicting and detecting objects of various shapes of sizes. Large feature maps detect small objects well, and small feature maps help detect small objects better. These classes are simply explained in Fig.8.

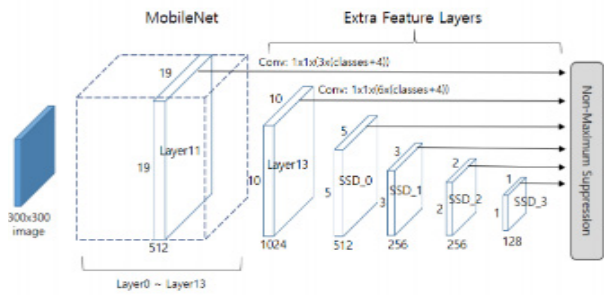


Fig. 7. Architecture of SSD MobileNet V2

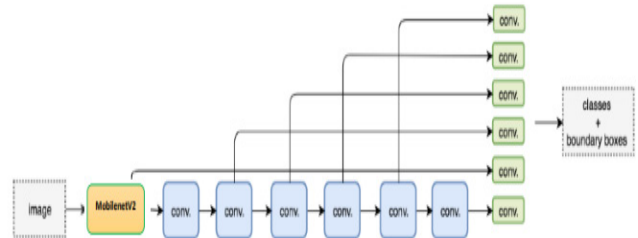


Fig. 8. Diagram simulating how MobileNet V2 SSD works

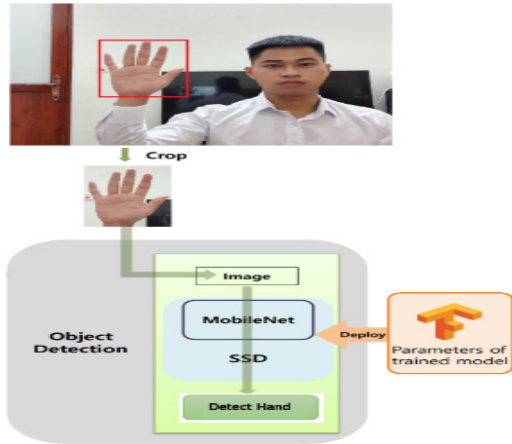


Fig. 9. Training procedure for hand detection model

2.4.3 Train the hand detection model

The training procedure of the hand detection model is shown in Fig.9. The first step of the process is to collect data (8000 images in the Hand Gesture Database GTI and 4000 self-collected images), then label the hand object in each image and finally edit the configuration file and run the training. The training hand detection model ran on NVIDIA Jetson Nano embedded computer.

2.4.4 Robot control convention

The robot control convention is implemented by the research team as follows: (1) Hand in fist state: Robot stops and (2) Hand spread: Robot moves according to hand. Moving left or right will depend on whether the hand is on the right or left side of the camera. Moving forward or backward will depend on the hand distance to the camera. If the hand is far from the camera, the robot will advance, the hand near the camera, the robot will back.

3. Results and discussion

3.1 Evaluation and results of robot model moving by lane

3.1.1 Evaluation on test set

The model is evaluated with the test set and based on mean squared error (MSE) and the coefficient of determination R^2 determined by the formula (2) with n is number of data; y_i is real data value i ; \hat{y}_i is predicted value for data i and \bar{y} is the average value of the dataset:

$$R^2 = 1 - \frac{\sum_i^n (y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \bar{y})^2} \quad (2)$$

With $MSE = 4.7$ and $R^2 = 97.65\%$, it is possible to see that this is a pretty good result and the lane

characteristics are clearly extracted when put into convolutional layers, not the environmental factors. Fig.10 illustrates the input image after going through the 3rd convolutional layer.

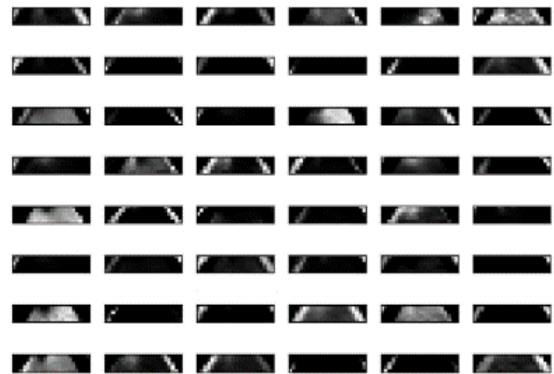


Fig. 10. Feature map after the third layer of convolutional

3.1.2 Actual evaluation

Testing the model on hardware with the self-generated lane is good when the car is in lane. The average speed to handle each frame is 45ms equivalent to the output video with an average of 22 frames per second (fps).

3.2 Evaluation and results of the hand-followed robot controller model

In this section, the results and evaluation of the hand-followed robot control model will be detailed. First, we will evaluate the hand detection model. Next is the process of optimizing the model so that the model can work best on the available hardware and thereby helping the freight robot follow the hand of the operator.

3.2.1 Evaluate the hand detection model with SSD Mobilenet V2



Fig. 11. Test robot moves in lane

The evaluation of the hand detection model is based on parameters such as the Mean Average Precision (mAP, Fig. 12) value and the value of the loss functions, namely the loss function on the validation set (Validation loss, Fig.13) and training set

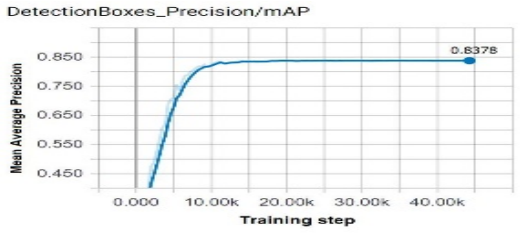


Fig. 12. mAP

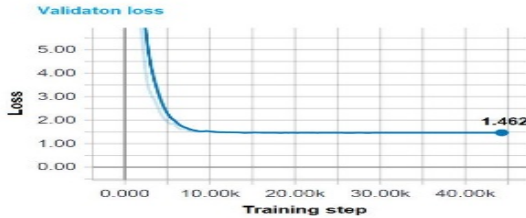


Fig. 13. Validation loss



Fig. 14. Training loss

(Training loss, Fig.14). "mAP" is the main data to evaluate the model, mAP is calculated as the average accuracy value over 10 Intersection Over Union (IOU) thresholds from 0.5 to 0.95 respectively, each threshold increases with the value of 0,05. IOU is a parameter used to evaluate the overlap between a given bounding box and a prediction box. By calculating IOU and considering a fixed threshold, we can tell whether the prediction box is a correct prediction or a false prediction. The IOU index is calculated by dividing the overlapping area between a given bounding box and the predictive box by the integrated area by the two boxes. After training 44,000 steps (corresponding to about 100 epochs) and using the Tensorboard as a tool to evaluate the model, we obtained a graph showing the average accuracy of the model. The results obtained the mAP average accuracy value of the hand gesture detection model was 83.78%. This is considered a good parameter threshold for an object detection problem.

In addition to the average precision value, the value of the loss functions is also an important parameter to evaluate whether the object detection model is working well or not. The smaller the value of the loss function, the better the model works. In addition, comparing the loss function on the validation set and the loss function on the training set also helps to check if the model is overfitting or underfitting, thereby giving timely solutions.

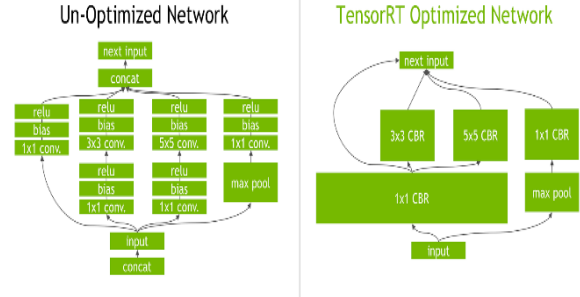


Fig. 15. Example of a TensorRT operation

The loss function of the SSD model used in this paper is constructed by Localization loss (to evaluate detection) and Confidence loss (to evaluate object classification). Specifically, the formula (3) with N is the number of boxes that match a given default box; α is an equilibrium coefficient for the effect of Localization loss; Lconf is Confidence loss and Lloc is

$$L = \frac{1}{N} (L_{conf} + \alpha L_{loc}) \quad (3)$$

After training model for about 44000 steps, we have the results as shown in Fig.14 and Fig.15, showing the loss function on the validation set and the loss function on the training set. The loss value on the validation set gradually decreases and gradually stabilizes at the threshold of 1,462, It is similar to the validation set, the loss value on the training set also gradually decreases and gradually stabilizes at the threshold of 1378. For an object detection model using a MobilenetV2 SSD, the loss value usually falls between 1 and 2, so with the 2 values obtained above, we can evaluate that the model works quite well. In addition, comparing the two graphs, we can see that the two charts both tend to decrease gradually and are equally stable at each training step. Hence, it can be concluded that the pattern shows no signs of overfitting or underfitting.

3.2.2 Optimization of the hand detection model

When testing the hand detection model on the robot's processor, the results obtained from the hand detection are very good but the processing speed is only 5fps, with this processing speed, the robot is very difficult to move according to the hand of the operator. That requires the team to optimize the model on the robot's hardware. The team used the TensorRT [5] to optimize the model running on the jetson nano processor.

The operating principle of TensorRT is illustrated in Fig.15. During optimization, TensorRT performs, transformations and optimizations that are important to the model. First, the classes with unused outputs are discarded to avoid unnecessary

computation. Next, if possible, certain layers such as convolution, Bias, and ReLU are merged to form a single layer to improve latency, throughput, power efficiency, and memory consumption. The process of optimizing the model is shown in Fig.16. The trained model will be in the form of "frozen inference graph", before optimization, it is necessary to set parameters in the configuration file to match the parameters of the original Tensorflow model, some parameters need to be set correctly to avoid errors during optimization such as the number of classes and the names of classes. When running, TensorRT will convert the freeze inference graph to the UFF file and convert it from the UFF file to TensorRT engines. Finally, on the Jetson Nano embedded computer, we use a combination of

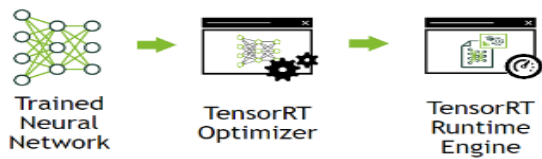


Fig. 16. Model optimization process with TensorRT

TensorRT engines with code to perform object detection. After optimizing the hand detection model and running again on the robot's processor, the results were much better than before. Average accuracy remained unchanged and reached the threshold of 83.78%, processing speed increased by more than 4 times compared to before optimization and reached 21 fps.

3.2.3 The comparison of test results between some other deep learning network models

In order to be able to explain why the choice of an SSD network model in combination with MobilenetV2 deployed on a Jetson Nano embedded computer as a hand detection model, the team ran a test system with several network models. Learn more deeply (using the same training data set and deploy on the same hardware as the article presented) and compare the results from actual testing of these models with the MobilenetV2 SSD model already come to the final conclusion. The comparison results are presented in Table 2. We can see that the ResNet-50 SSD gives the highest average accuracy, but the processing speed is only 9 fps, which is quite slow and cannot respond to a Cargo robot system in the factory. In contrast, the MobilenetV2 SSD model offers a much higher processing speed (21 fps), but the average accuracy is not too low compared to the ResNet-50 SSD (83.78% and 87.90%). Based on the above comparison results, we can see that the MobilenetV2 SSD is more suitable than the other deep learning network models, it meets two conditions simultaneously: accuracy and

processing speed when running on a Jetson Nano embedded computer. In order to be applied to the transportation of goods in factories, the robots must meet many strict conditions in terms of accuracy or speed of control signal processing. The deployment of the transport robot model moving by following the hand of the operator model using the MobileNet V2 network combined with the SSD network as the hand detection device to provide control signals has partly solved that problem. With 83.78% of accuracy and 21 fps processing speed of the model, the transport robot can follow the hand of the operator in factories.

Table 2. Results of actual implementation of the system with different deep learning models

Deep learning model	FPS	mAP
SSD Mobilenet V2	21 fps	83,78 %
Tini YOLO V3	14 fps	77,60 %
SSD ResNet-50	9 fps	87,90 %



Fig. 17. Running robot

3.3 General Results

Robots designed and tested by the research team can operate stably with good and stable movement and transportation on flat surfaces of factories and workshops. The deep learning models are used to meet the processing speed and precision required by a robot cargo system in the factory, specifically the robot can easily move in a lane or the robot can move as soon as the hand of the operator signals it. During the trial run continuously for a long time, the electronic components operate normally, meeting the requirements of the research. Fig.18 shows a picture of the robot model after its assembly is complete. The selection of hardware devices and then assembling them together rather than design from scratch is part of the team's plan to be able to shorten the time to build the hardware and focus on software implementation. After a long period of design, fabrication, and commissioning, it can be seen that the freight robot system presented in this article has met the requirements for stability, compactness, and flexibility in movement, and can be applied in factories in the future.

4. Concluding remarks

This article presents the application of using convolution neural networks in designing and manufacturing cargo transport robot models in factories. After a long time of research, design, and testing, the team successfully designed the model. The primary goals have been met. The cargo transport robot model will include two modes of movement, the first mode is automatic lane movement and the second mode is the hand movement of the operator. Although the robot model is relatively small in size, it can thus be the core to produce larger robots, transporting more goods. About two deep learning models for controlling lane-driven robots and a deep learning model for controlling robots followed by hand. These two models have demonstrated the superiority of accuracy and stability in processing speed so that the robot can automatically move easily in a lane or followed the hand of the operator in settings such as factories.

The research results still have some limitations, so the group will continue to research and improve their work such as (1) Research to minimize the impact of the outside environment on the robot's mobility; (2)

Research plans to help robots move in difficult terrain or rough surfaces in some factories; (3) Apply the design of larger transport robots so that more cargo can be transported; (4) Incorporate models of obstacle recognition and (5) Design and fabricate some hardware circuits from scratch without using the available hardware modules.

References

- [1] <https://vinodsblog.com/2018/10/15/-every-thing-you-need-to-know-about-convolutional-neural-networks/>, Last accessed on 28/06/2020.
- [2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, End to End Learning for Self-Driving Cars, Holmdel, USA, 2016.
- [3] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang Chieh Chen MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2018.
- [4] Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C., Berg A, SSD: Single Shot MultiBox Detector In ECCV, 2016.
- [5] <https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html>, Last accessed on 28/06/2020.