Mobile Device Based Outdoor Navigation With On-line Learning Neural Network: a Comparison with Convolutional Neural Network

Zejia Zheng
Michigan State University
428 South Shaw Lane, East Lansing
zhengzej@msu.edu

Juyang Weng
Michigan State University
428 South Shaw Lane, East Lansing
weng@cse.msu.edu

Abstract

Outdoor navigation is challenging with its dynamic environments and huge appearance variances. Traditional autonomous navigation systems construct 3D driving scenes to recognize open and occupied voxels by using laser range scanners, which are not available on mobile devices. Existing image-based navigation methods, on the other hand, are costly in computation and thus cannot be deployed onto a mobile device. To overcome these difficulties, we present an on-line learning neural network for real-time outdoor navigation using only the computational resources available on a standard android mobile device (i.e. camera, GPS, and no cloud back-end). The network is trained to recognize the most relevant object in current navigation setting and make corresponding decisions (i.e. adjust direction, avoid obstacles, and follow GPS). The network is compared with state of the art image classifier, the Convolutional Neural Network, in various aspects (i.e. network size, number of updates, convergence speed and final performance). Comparisons show that our network requires a minimal number of updates and converges significantly faster to better performance. The network successfully navigated in regular long-duration testing in novel settings and blindfolded testing under sunny and cloudy weather conditions.

1. Introduction

In this paper, we describe the design and learning system for our outdoor navigation application based on the Developmental Network (DN). The goal for this application is to help visually impaired people walk around using the mobile phone built-in camera and GPS. Although there are many existing algorithms for outdoor robotic navigation, these algorithms are often dependent on proximity sensors such as radars which are not available in current mobile devices [12, 7]. Visual based approaches proposed in [2, 9], on the other hand, are costly in computation, which makes real-time training and testing with no cloud back-end support impossible. Developing a mobile device based navigation application requires careful engineering of a complex system which leverages action accuracy to balance the trade-offs imposed by the limited computational resources of a mobile device.

In contrast with other outdoor navigation systems which often have their own motors to carry out precise movements [7, 12], our mobile phone based application provides only rough instructions to the human user (i.e. left, slightly left, right, slightly right, forward, and stop). Instructions in our application are more similar to labels in image classification problem than detailed movement instructions (e.g. turn right 32 degrees versus turn right). Thus it is natural for one to resort to current state of the art image classifier Convolutional Neural Network (CNN). However, a closer look into the architecture reveals that the CNN, although giving satisfactory final performance, is not the best choice for real-time navigation. Detailed analysis is presented in next section.

2. Comparison with Convolutional Neural Network

Convolutional Neural Network (CNN) [6] is one of the leading image classification architectures for hierarchical feature extraction. CNNs have been reported to have state of the art performance on many image recognition and classification tasks, including hand written digit recognition [5], house number recognition [10], traffic sign classification [1], and 1000 class ImageNet dataset classification and localization [4, 11].

Although CNNs are proven effective in batch image classification and feature extraction, the architecture and learning mechanism suffers from the following drawbacks in mobile device based outdoor navigation:

1. Inconsistent on-line learning performance. Outdoor navigation settings are dynamic with meandering sidewalks and numerous obstacles. On-line learning,

Figure 1. Sample outdoor navigation settings. The samples show the environment in which we train and test our outdoor navigation application. The application faces dynamic environment with unpredictable obstacles and pedestrians. The application also needs to handle different lighting conditions.

(a) (b)

Figure 2. Blindfold testing. The user is completely reliant on the audio output of the navigation application during blindfold testing. A Go-Pro camera is mounted at eye-level of the user to record the environment and the output of the application for performance analysis. Refresh rate of the application is 5-8 frames per second. Audio instruction is broadcast to the user every 1.5 seconds. The application uses Google Directions API, Google Maps API and Google Location Service to grab the GPS input.

which allows the network to learn unfamiliar environments on the fly with few training samples, appears to be a more sensible option. However, CNN, trained with stochastic gradient descent (SGD), goes through many (usually hundreds of) iterations of fine-tuning before it reaches satisfactory performance. The slow convergence speed is also demonstrated in our experiment results presented in Fig.7(b).

Hebbian learning (implemented as Lobe Component Analysis [14] in our network), on the other hand, uses input examples to initialize weights and find the fine-tuning direction. Convergence speed of a hebbian learning based network is usually much faster than gradient based network of the same size, making it a better learning mechanism for on-line learning scenarios. Comparison of convergence speed is shown in Fig.7(b) and Sec.4.1.

2. Computationally costly update on resource limited mobile device. Unlike other neural network based applications (e.g. voice recognition and image classification) which often have cloud backends, real-time navigation requires fast computation and immediate response, making it impractical to wait for network communications to finish. Thus the computation and learning process must be solely carried out on the mobile device. To update its network weights, CNN relies on error-back propagation to find out the partial derivative of the error function with regard to each weight. The tuning process uses too many resources on a mobile device, making it impractical for the real-time navigation task.

   In contrast, competitive learning (modeled as top-k competition in our network) only updates a small portion of the entire network. Neurons with the highest firing values, meaning that their stored patterns are most relevant to the current input, inhibit firing in the other 'irrelevant' neurons. Weight fine-tuning takes place in only these firing neurons, which is usually set to be a small portion of the entire network, saving computational resources and preserving the stored pattern in the other inhibited neurons.

Based on these two observations, we present our hebbian learning neural network with top-k competition for mobile device based outdoor navigation. The network is proven to be able to perform at around 95% action accuracy in indoor environments [15] and this is our first set toward real-world outdoor navigation. As presented in Table 1, our DN uses hebbian learning to achieve stable performance at early stages of training (1-2 epochs), which makes the network an ideal candidate for on-line learning. Top-k competition minimizes the number of weight updates for each learning sample, speeding up the computation to real-time response (5-8 frames per second) for outdoor navigation. With the help of top-down recognition information at training stage, the final action accuracy of DN is 2% higher compared to the action accuracy of CNN. The detailed discussion of performance evaluation is presented in Sec.4.1.

Although the Developmental Network draws its inspiration from Hebbian learning and shares similarity with Kohonen's network (also know as self-organizing map) [3], there are major conceptual and architectural differences between a DN and an SOM (i.e. self-organizing map). While SOM uses purely bottom-up input to perform clustering without supervision, top-down connections in DN extract motor discriminative features as shown in the work of Luciw and Weng [8]. Top-down connections from motors allow the network to generate context based attention (e.g. when turning right the neurons focusing on the right half of the input would be more likely to fire). When expanded along the time axis, the top-down connections can be viewed as a recurrent connection from motors for the network to generate one-step predictions according to the current firing patterns in both hidden area and the motor area.

Table 1. Comparison between CNN_GPS_Recog and DN_Full architectures.

	CNN_GPS_Recog	DN_Full
Num. of weights	276500 weight sharing	1309232 local weights
Num. of neurons	46986	3900
Num. of updated weights per sample	276500	**3081**
Learning mechanism	SGD	Hebbian
Convergence speed	slow 450 epochs	fast 1-2 epochs
Online learning	no	**yes**
Best performance (error action rate)	24.70%	**22.80%**

3. Developmental Network for outdoor navigation

3.1. Developmental Network (DN)

A Developmental Network has three areas: a sensory area denoted as X, a hidden area denoted as Y, and a motor area denoted as Z. Neurons, located in the hidden area on a two-dimensional grid, accept global or partial input from their receptive fields. The connection between the hidden area and the motor area is bidirectional.

DN, an on-line learning system, constantly updates its Y area from the input in X and Z area. The Z area serves as additional input when the motors are supervised. When the agent is in testing phase, then Z area serves as area where the agent performs movement corresponding to the firing pattern in Z.

Firing of neurons in the network goes through the following stages:

1. Similarity measure. At this stage, each neuron compares its received input with its stored pattern and uses the calculated similarity as the neuron's firing value. The preresponse of the bottom-up response in each neuron is calculated as follows:

$$\hat{r}_{u,i} = \langle \frac{\mathbf{x}_t}{||\mathbf{x}_t||}, \frac{\mathbf{w}_{u,i}}{||\mathbf{w}_{u,i}||} \rangle \qquad (1)$$

where $\mathbf{x}_t$ is the sensory input vector from X area at time t, and $\mathbf{w}_{u,i}$ is the corresponding bottom-up weight of neuron i. The brackets indicate inner product of two unit vectors. $r_{u,i}$ is then calculated from $\hat{r}_{u,i}$ by prescreening, modeled as top-k competition. Similarly, $\mathbf{z}_t$ is be used to calculate the top-down response $r_{d,i}$ for each neuron by replacing the $\mathbf{x}_t$ and $\mathbf{w}_{u,i}$ in Eq. (1) with $\mathbf{z}_t$ and $\mathbf{w}_{t,i}$.

2. Inhibition and competition. Neurons are then competing to fire. Top-k competition is used as a simulation of global dynamic inhibition among neurons. After each neuron i computes its bottom-up response value, $r_{u,i}$, and top-down response value, $r_{d,i}$, the neuron's preresponse value is set to be the average of the two values:

$$r_i = \frac{1}{2}(r_{u,i} + r_{d,i}) \qquad (2)$$

The final neuron response in the Y area is given by top-k competition. The k neurons with the highest preresponse value will fire with the adjusted responses, while other neurons will be suppressed. To adjust the response values based on their ranking:

$$r'_i = \begin{cases} r_i \cdot (r_i - r_{k+1})/(r_1 - r_{k+1}) & r_{k+1} \le r \le r_1 \\ 0 & otherwise \end{cases}$$

where r_1 is the highest response value; r_{k+1} is the $k+1$th highest response value, r_i is the original response and r'_i is the adjusted response. In our experiment, we set $k = 3$ for each Y layer.

3. Learning and updating. Hebbian learning takes place in firing neurons. The input which triggers firing in winning neuron is remembered as an incremental average to the existing weight vector. If a neuron wins in the multistep lateral competition described above (its firing rate is larger than zero), its bottom-up weight and top-down weight would be updated using the following Hebbian learning rule:

$$\mathbf{w}_{u,i} \leftarrow \beta_1 \mathbf{w}_{u,i} + \beta_2 r'_i \mathbf{x}_t$$

where $\beta 1$ and $\beta 2$ determine retention and learning rate of the neuron, respectively:

$$\beta_1 = \frac{m_i - 1 - \mu(m_i)}{m_i}, \beta_2 = \frac{1 + \mu(m_i)}{m_i} \qquad (3)$$

with $\beta_1 + \beta_2 \equiv 1$, m_i is the neuron's firing age (i.e. $m_i = 1$ in the beginning of training, and increments by one each time the neuron wins lateral competition), and $\mu(m_i)$ is the corresponding learning rate:

$$\mu(m_i) = \begin{cases} 0, & \text{if } m_i < t_1 \\ c(m_i - t_1)/(t_2 - t_1), & \text{if } t_1 < m_i < t_2 \\ c + (m_i - t_2)/\gamma, & m_i > t_2 \end{cases} \qquad (4)$$

We used typical value $t_1 = 10, t_2 = 10^3, c = 1$ and $\gamma = 10^4$ in the experiment.

3.2. DN with multiple concept zones

In this paper, our outdoor navigation DN_Full uses one sensory area and four motor areas, as shown in Fig.3.

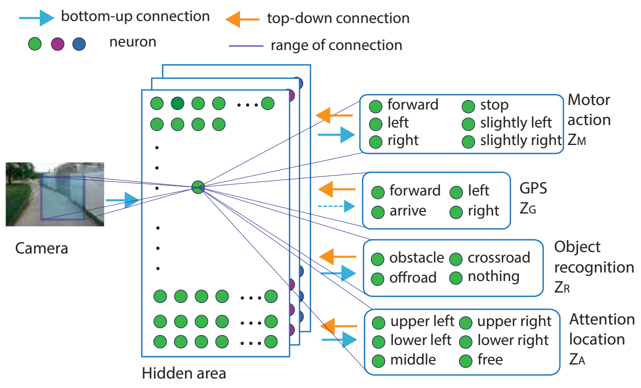


Figure 3. DN used for outdoor navigation. The hidden layer contains three sub-layers with neurons of different receptive field sizes (15 pixels,19 pixels and 21 pixels). The GPS motor zone Z_G is always supervised by the mobile phone GPS, thus the bottom up connection to that zone is not used in computation (dashed). Detail about the network is presented in Table 2. DN_No_Recog shares the same architecture as this DN (denoted as DN_Full) in the hidden layer and the input layer. The difference is that DN_No_Recog does not have concept zone Z_R and Z_A.

One sensory area (X) and four motor areas (Z_M for motor action, Z_G for GPS input, Z_R for object recognition, and Z_A for attention of the agent) are used in the navigation experiment presented in Sec.4. The network also has four sets of top-down weights for each of the motor concepts. The top-down response calculated in Eq. (2) would then be calculated as:

$$r_{d,i} = \frac{1}{4}(r^M_{d,i} + r^G_{d,i} + r^R_{d,i} + r^A_{d,i}) \quad (5)$$

To show that object recognition modules (Z_R and Z_A) are needed to improve the performance of DN, we removed the two concepts zones related with object recognition and constructed DN_No_Recog. DN_No_Recog would thus have two motor areas (Z_M and Z_G). Eq. (5) changes accordingly.

Worthy of notice is that the recognition concepts (i.e. Z_R and Z_A) are not supervised during testing. This means the top down response would not contain $r^R_{d,i}$ and $r^A_{d,i}$ when the network is in validation or testing mode.

3.3. Hardware platform

The mobile device we used in the experiments is a HTC One M8 with a 2GB Ram and a Quad-core 2.3 GHz Krait 400 CPU. To facilitate training and testing, a Moga Pro Bluetooth controller was paired with the application to feed in supervised motor information and track the number of errors made. The mobile device is equipped with a 4MP autofocus camera, which serves our purposes well. The phone also has a Adreno 330 GPU installed, which makes computation parallelization with OpenCL possible for future extension. Currently all code is written in JAVA with Android SDK. The captured image is resized into a 38 by 38 grayscale image using OpenCV4Android. The current update speed is 5-8 frames per second in learning and testing mode depending on the status of the mobile device.

Table 2. Navigation experiment network detail

Zone	Number of neurons	Detail
X	38x38 matrix, real value	reshaped to 1x1444 and normalized
Y	layer 1: 20x20x3 neurons layer 2: 18x18x3 neurons layer 3: 24x24x3 neurons	receptive field size 19 receptive field size 21 receptive field size 15
Z_M	6 neurons	6 motor actions
Z_G	4 neurons	4 GPS inputs
Z_R	4 neurons	4 obstacle categories
Z_A	6 neurons	6 attention regions

4. Experiment

The goal for this application is to help people navigate in outdoor environments. So far we have performed experiments by walking around the university campus. While our application is designed to learn on-line and in real-time, the performance of the network is compared with other visual recognition methods using collected batch data as is shown in the next subsection.

The outdoor navigation for the agent is simplified into nine different general settings, shown in Fig.5.

4.1. Batch training and performance validation

To validate the performance of our network and make comparisons, 4109 training samples are collected at different times around the university campus for batch training and testing. The data collection process consists of 4 different sessions with 2 lighting conditions: mid-afternoon when direct sunlight causes sharp contrast of shadows on the road, and late afternoon when the contrast is not as obvious. One training sample consists of an input image, the GPS signal at that time, the correct action, the most relevant object that needs to be recognized, and the location of that object in the input image (i.e. attention). The first two concepts were recorded in real-time using the paired controller and the application interface. The latter two concepts (i.e. type and attention) can also be specified using the interface by stopping the recording session and selecting the specific items. In the experiments these two concepts were manually labeled after we collected the data.

4-fold cross validation was performed after data was collected and labeled. The tested network was trained for 10 epochs, with performance evaluated at the end of each epoch.

To make comparisons, the performance of our network without the object recognition concept zones is also re-

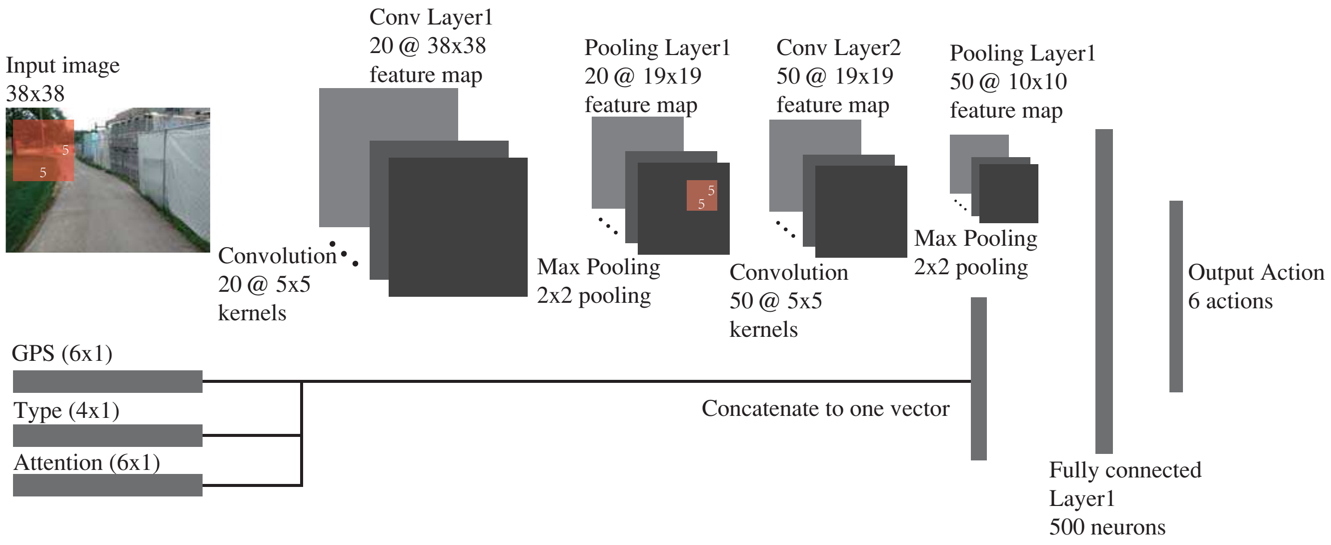


Figure 4. CNN_GPS_Recog architecture for outdoor navigation. The three subsidiary signals are concatenated into one feature vector, which is then combined with the high level feature vector generated by two rounds of convolution and max-pooling over the input image. A two layer fully connected neural network is then deployed on top of the feature vector to generate the final output action. This architecture can be trained using stochastic gradient descent (SGD) and error-back propagation. We compare the performance of this architecture with our Developmental Network in Sec.4.1. The CNN_GPS network shares the same architecture as the CNN_GPS_Recog in convolution and pooling parameters, but there are no available type and attention information for CNN_GPS during training.

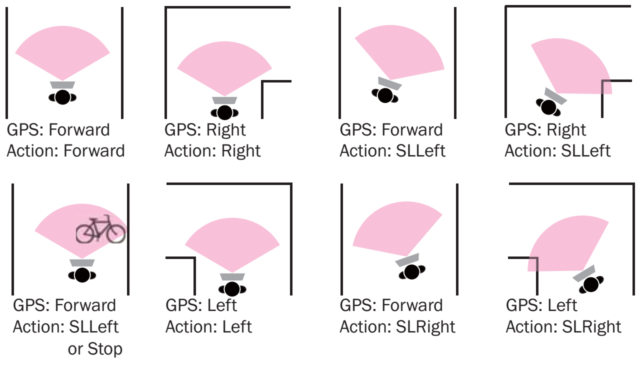


Figure 5. Navigation settings faced by the application. The agent needs to learn the corresponding actions on-line in real-time without prior knowledge of the environment. Regardless of the situation, GPS input of arrive corresponds to an action of stop, which is not plotted in the figure. DN_Full is also required to learn the most relevant object in the current setting. *SLLeft* is short for *slightly left*. *SLRight* is short for *slightly right*.

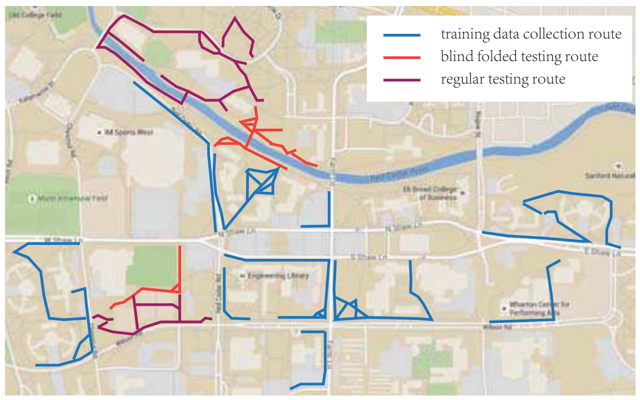


Figure 6. Campus map of Michigan State University. Data collection routes, blindfold testing routes and regular testing routes are marked in this figure. There is no overlap in these routes.

ported, as shown in Fig.7(a). Please note that our DN already demonstrated performance close to its best performance among the 10 epochs after only 1 epoch of training. This speed of convergence is significantly faster compared to typical gradient decent based methods like Convolutional Neural Networks as shown in Fig.7(b). Both the CNN_GPS and CNN_GPS_Recog reach performance comparable to DNs at about the 400th epoch.

The additional recognition information helps the network to generalize learned concepts to unfamiliar environments, boosting the performance by almost 4 percent. The additional information does not help the CNN to gain better performance. This can be explained by the fact that CNN relies on the calculated errors to fine tune the internal weights, and the error is only calculated from the action of the agent. The recognition information is not affecting the formation of convolutional layers, and the recognition information is not affecting testing results since no such information is available during testing. However, in DN, recognition information is utilized as the top-down input from motor concept. The neurons with better recognition results are more likely to fire and win in the top-k competition. Information in those concept zones helps to select the most appropriate neuron to learn the current input, thus increasing the performance by almost 4 percent.

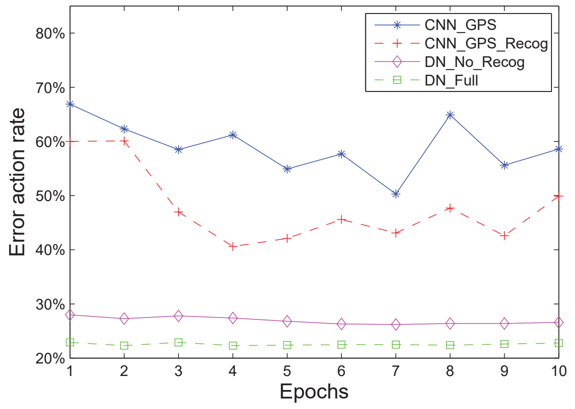
Another thing to notice is an inconsistent action in cross validation does not necessarily mean a wrong action in real world environment. The network is designed to able to recover itself from the wrong facing directions and return to the correct position. Thus in real world testing, we report performance in two categories: inconsistent with the supervisor's intention, and definite errors. Errors are defined as hitting the obstacle, stuck in stop action, or step off road.

4.2. Autonomous navigation in novel settings

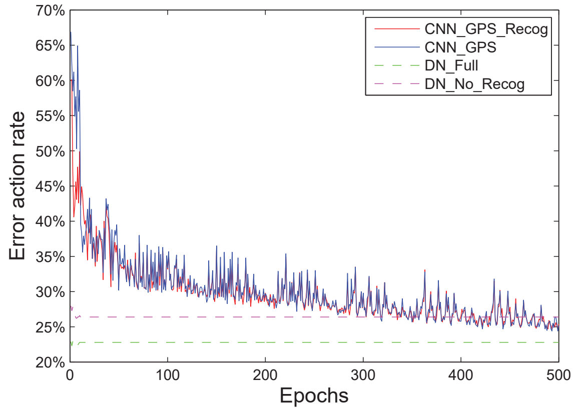
In this experiment the network is put in real-world outdoor environments and its performance is tested. 1516 actions were tested around the university campus. We made sure to choose testing routes different from the training routes where batch training data was collected as described in Sec.4.1. The testing routes and the training routes are shown in Fig.6. Testing was carried out in real-time with the user checking the correctness of output at each step, which takes about 1.5 sec per step. The user would then carry out the action instructed by the application and move on to the next step. If the output action is different from the intention of the user, then the count indicating this difference (the Diff count) would increment by 1. If the result of the output action leads to a collision into obstacles, or stepping off the road, or missing the arrive point, or if the user is stuck in stop output at the wrong location, then the Error count would increase by 1.

To check the influence of sunlight and shadows, the experiment was carried out in 2 different sessions within four days. Results of this testing is presented in Table 3. Session 1 was carried out in mid-afternoon when the degree of contrast between shadow and bright surfaces was at its peak. Session 2 was performed at late afternoon when the contrast was not as sharp. The results show that our network gives almost perfect performance at the testing routes. All of the errors observed in the testing scenarios occurred in session 1 testing, when the strong shadows were recognized as obstacles by the network. Testing outputs are presented in Fig.8.

It can be seen from the samples in Fig.8 and Table 3 that the biggest challenge for the current network is the shadows on the ground. In session 2 testing when the shadows are less observable, the network gives flawless action accuracy, with no errors observed in the 736 actions tested. Because only grayscale images are used in the internal computation, the shadow can be easily mistaken as obstacles blocking the path. One solution is to add shadows as another recognition type in the recognition motor, and use RGB images instead of grayscale images for our network.



(a) Error action rate of 10 epochs of training.



(b) Error action rate of 500 epochs of training.

Figure 7. Error action rates of four network architectures. Hebbian learning based networks like DN_Full and DN_No_Recog approach their best performance after only one epoch of training. The SGD based architectures like CNN_GPS and CNN_GPS_Recog only reaches comparable performance after 400 epochs of fine-tuning. With the help of type information and location information, DN_Full learns to recognize the most relevant objects in the current setting, thus achieving better performance compared to other architectures. However, CNN_GPS_Recog achieves slightly better performance at the beginning of training compared to CNN_GPS. Type information and location information does not seem to be helpful in the process of gradient descent. Detailed discussion is presented in Sec.4.1.

4.3. Autonomous navigation with user blindfolded

The real-world testing described in Sec.4.2 was performed with the user carrying out the output action with the ability to see the environment and adjust his movement. In this testing we blindfold the user and change the output of the application to audio output. The instructions generated by the application is broadcasted to the user every 1.5 second. Three blindfolded test sessions at different places around the campus were performed. Routes of these three

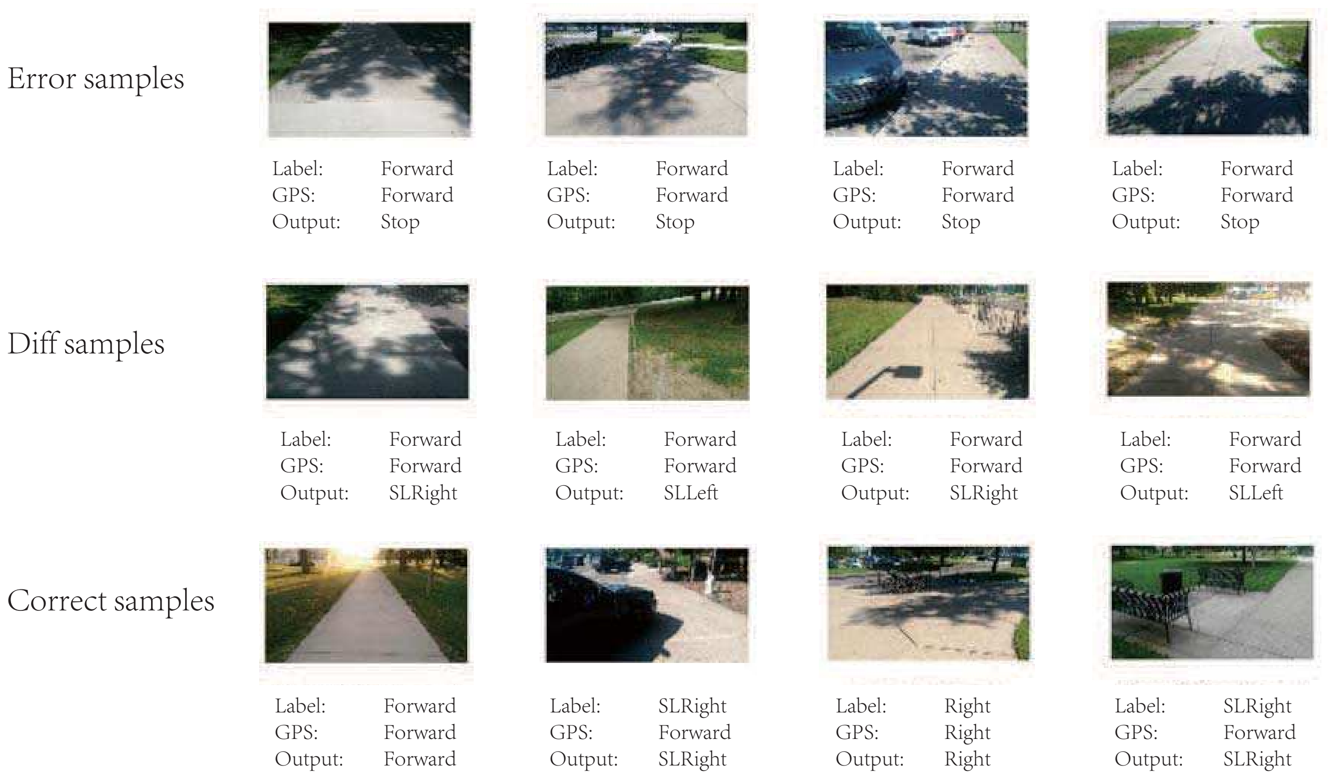


Figure 8. Testing samples from regular testing session. Sharp contrast caused by over-casting shadows was the biggest challenge to the performance of the network. The observed errors are all caused by shadows with sharp contrast and irregular shapes. Sec.4.2 discusses how this issue can be resolved. *Diff* is short for *different from supervisor's intention*.

Table 3. Real-time testing result with DN_Full. SLLeft is short for slightly left, and SLRight is short for slightly right.

	Summary			Session 1 14:00-15:30 (2 days)			Session 2 18:30-20:00 (2 days)		
	Total Actions	Total Diff	Total Error	Actions	Diff	Error	Total Actions	Diff	Error
Right	155	6 (3.87%)	1 (0.65%)	84	4	1	71	2	0
SLRight	172	18 (10.47%)	1 (0.58%)	82	16	1	90	2	0
Forward	788	124 (15.74%)	11 (1.40%)	438	72	11	350	52	0
SLLeft	214	24 (11.21%)	0 (0.00%)	103	16	0	111	8	0
Left	128	10 (7.81%)	0 (0.00%)	73	10	0	55	0	0
Stop	59	6 (10.17%)	0 (0.00%)	0	0	0	59	6	0
Total	1516	188 (12.40%)	13 (0.86%)	780	118	13	736	70	0

blindfolded tests are presented in Fig.6. Because the user cannot check the output at real-time and evaluate the correctness of movement at each step, accuracy of the output is not precisely evaluated.

Detailed description about this navigation experiment is presented in Table 4. The agent failed at the second blindfolded session, leading the user into a glass door at the library entrance. Glass door was a novel obstacle which is not included in the training set. This failure shows that the network would fail to recognize untrained objects, which means (1) incremental learning is important for real-world applications, and (2) novel object detection and learning mechanism is needed for a full-fledged commercial product.

Demonstration video of this process is available at: https://youtu.be/2XK69kAkFcQ?list=PLVs0MJh9CrcF -rqzfYAluiZtfs_y6mvD0. Note that when copying and pasting the url address from pdf to browser, the dash and underscore need to be added manually. Full length recording of the demonstration is available upon request.

Table 4. Blindfold testing detail

Steps	Detail Description	Weather condition	End Result
425	Forward on shadowed road ->Turn right ->Avoid pedestrians ->Go over bridge ->Bikes and pedestrians ->Avoid bushes ->Fork road ->Bushes ->Trees ->Garage building ->Shadows ->Destination	Sunny with shadows	Success
217	Forward on open settings ->Turn left ->Fork road ->Cross road ->Poster signs ->Litter bin ->Building entrance	Cloudy with no obvious shadows	Failed at building entrance
480	Forward on shadowed road ->Turn right ->Turn right ->Avoid Pedestrians ->Avoid lamp posts ->Tilt to right ->Destination	Sunny with shadows	Success

4.4. Real-time training

Our network learns on-line with all the computation carried out on the phone with no cloud back-end. A demonstration of on-line learning can be found at https://youtu.be/2XK69kAkFcQ?list=PLVs0MJh9CrcF -rqzfYAluiZtfs_y6mvD0. Note that for debugging purposes, we are using supervised GPS instead of phone built-in automated GPS in the real-time training video.

5. Discussion

We have described an outdoor navigation application for mobile devices based on the on-line learning object recognition neural network. The network operates in real time in a dynamically varying outdoor environment. All learning and computation takes place on the mobile device in less than 0.2 sec per frame with no cloud backend. Hebbian learning helps the network to converge at a significantly faster speed compared to stochastic gradient descent approach. Top-k competition minimizes the number of updates required per learning sample and thus boosts the refresh rate of our application.

There are several limitations to our current experiment. Testing of our network was performed under only sunny or cloudy weathers. We have not tested or trained the network in rainy weathers. However, with enough neural resources (number of neurons, and more epochs of learning) and training samples, our network will be able to demonstrate success in other weather conditions. The navigation action tested was limited to obstacle avoidance and direction correction. To perform complicated action sequences requires detailed design in motor zones. DN has already been proven to be capable of simulating the behavior of a Finite Automaton error free, given enough computational resources [13]. Thus our network should be able to handle sequential movements, but this goes beyond the scope of this paper. Furthermore, the input image was resized to a much smaller size so that the computation can be carried out in real time. However, this may leads to loss in available information and downgrade the performance. We are currently moving costly computation to OpenCL parallelization to exploit the computational power of the mobile device.

References

[1] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[2] J. D. Crisman and C. E. Thorpe. Scarf: A color vision system that tracks roads and intersections. *Robotics and Automation, IEEE Transactions on*, 9(1):49–58, 1993.

[3] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1–6, 1998.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] B. B. Le Cun, J. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[7] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[8] M. Luciw and J. Weng. Top–down connections in self-organizing hebbian networks: Topographic class grouping. *Autonomous Mental Development, IEEE Transactions on*, 2(3):248–261, 2010.

[9] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, DTIC Document, 1989.

[10] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3288–3291. IEEE, 2012.

[11] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[12] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[13] J. Weng. *Natural and Artificial Intelligence: Introduction to Computational Brain-Mind*. BMI Press, East Lansing, MI, 2012.

[14] J. Weng and M. Luciw. Dually optimal neuronal layers: Lobe component analysis. *IEEE Transactions on Autonomous Mental Development*, 1(1):68–85, 2009.

[15] Z. Zheng, X. He, and J. Weng. Approaching camera-based real-world navigation using object recognition. *Procedia Computer Science*, 53:428–436, 2015.