Incorporating Side-Channel Information into Convolutional Neural Networks for Robotic Tasks

Yilun Zhou[1] and Kris Hauser[2]

***Abstract*— Convolutional neural networks (CNN) are a deep learning technique that has achieved state-of-the-art prediction performance in computer vision and robotics, but assume the input data can be formatted as an image or video (e.g. predicting a robot grasping location given RGB-D image input). This paper considers the problem of augmenting a traditional CNN for handling image-like input (called main-channel input) with additional, highly predictive, non-image-like input (called side-channel input). An example of such a task would be to predict whether a robot path is collision-free given an occupancy grid of the environment and the path's start and goal configurations; the occupancy grid is the main-channel and the start and goal are the side-channel. This paper presents several candidate network architectures for doing so. Empirical tests on robot collision prediction and control problems compare the the proposed architectures in terms of learning speed, memory usage, learning capacity, and susceptibility to overfitting.**

I. INTRODUCTION

Deep learning is a powerful machine learning technique that uses many-layered neural networks. One particularly successful network structure in computer vision and image processing domains is the Convolutional Neural Network (CNN), where each layer performs a feedforward convolutional transformations from an input tensor (an image-like, multi-dimensional dense array) to an output tensor of different shape. CNNs are particularly suited for image-like input due the spatial coherence of images and the advent of general purpose graphics processing units (GPUs) which make training fast on arrays up to 3- or 4-D. Exploiting these image-like properties leads to superior empirical performance compared to other learning methods such as support-vector machine (SVM) or multi-layer perceptron (MLP).

Deep network architectures for incorporating non-tensor information have been less well studied. Multi-modal deep learning architectures use several predictive channels in a unified manner to boost prediction performance [1], [2], [3]. By contrast, we consider a situation in which no channel is significantly predictive on its own, but rather the target concept exhibits strong coupling between channels.

This type of problem setting is characteristic of robotics and other control problems, in which other information about the robot's state and/or task parameters should be incorporated with image-like information. This *side-channel input* usually has much lower dimensionality than the *main-channel* image-like input, and does not exhibit spatial coherence. Furthermore, the side-channel input has equal or sometimes greater importance than the main-channel input in determining the outcome. For example, in predicting whether an autonomous vehicle should steer left, right, stay straight, accelerate, or brake, the outcome depends both on whether obstacles exist in the vehicle's way (from imaging) as well as rules of the road (which are state dependent) and vehicle's destination (which is task dependent).

This paper proposes and compares four generic deep architectures for augmenting traditional CNNs with side-channel information. The *activation map* architecture uses the side-channel to predict a modulation of the relevance of main-channel elements; the *mix-in* architecture folds the side-channel information into the first fully-connected layer of network; the *stacking* architecture augments each element of the main-channel with the entire set of side-channel variables; and the *input-modulated kernel* architecture computes CNN kernel weights using the side-channel.

Experiments are performed on classification and regression problems in the domain of robot collision detection and compliant simulation, in which the image-like main-channel is an occupancy map of the environment and the side-channel describes task-specific parameters. These toy problems were chosen to be similar to real-world problems in high-speed collision avoidance, but also easy to generate millions of training examples with perfect ground truth. The strengths and weaknesses of each architecture are compared on several metrics, including training time, space complexity, flexibility, and susceptibility to overfitting. Results suggest that activation map and mix-in architectures are most practically viable in terms of space requirements and training time, with activation map achieving the best overall performance.

Datasets, software, and extended results for all experiments can be found at `http://motion.pratt.duke.edu/sidechannel/`.

II. RELATED WORK

Our benchmark problems are in the domain of predicting collision with arbitrary environments, which represents fundamental operations in robot planning and control. Some authors have considered similar problems. For example, Pan et al [4] used locality-sensitive hashing to predict probability of collision-free connection in sampling-based motion planners. Jetchev and Toussaint [5], [6] used metric learning and support vector regression for trajectory quality prediction, including environmental features. In contrast we do not

*Y. Zhou is supported by a Pratt Undergraduate Research Fellowship. K. Hauser is partially supported by NSF grant #1218534.

[1] Y. Zhou is with the Department of Electrical & Computer Engineering and Department of Computer Science. `yilun@cs.duke.edu`

[2] K. Hauser is with the Department of Electrical & Computer Engineering and Department of Mechanical Engineering & Materials Science. `kris.hauser@duke.edu`

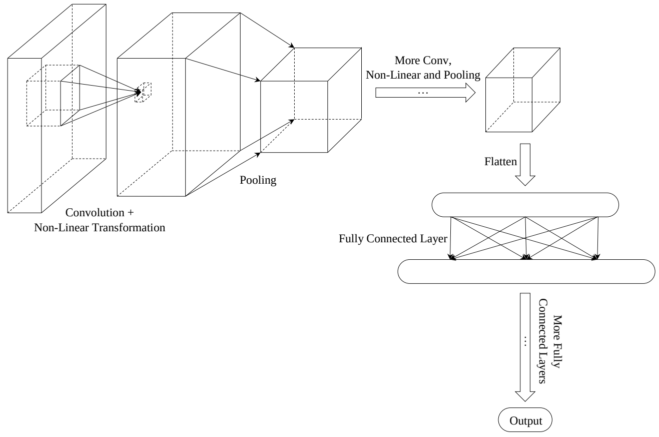


Fig. 1. Standard CNN Architecture

design features but instead feed raw occupancy maps to the learner.

CNNs have been used for handwritten character recognition [7] and face recognition [8] for decades. However, recently it gained a resurrection of people's interest by its superior performance [9] on the ImageNet challenge [10]. Since then, it has been applied to various other image-related tasks such as image generation [11], image captioning [1], and visual game playing [12].

CNNs have also been applied to many robotic tasks that require computer vision. Most of the work uses 2D image-like input. For example, Lenz et al [13] used a CNN to detect grasping location for a parallel gripper from RGB-D image; Schenck and Fox [14] combined a CNN and recurrent neural network (RNN) to track liquid hidden in container for robot manipulation tasks.

While most CNN work uses input of single modality: an image-like densely structured matrix (or tensor), there has also been some work on multi-modal deep learning. For example, Ngiam et al [2] studied restricted Boltzmann machine autoencoder architectures in combining audio and video data for recognizing syllables pronounced in a video. Wu et al [3] devised a deep learning architecture for learning image similarity from combination of different features (modalities).

In these and related works, correlations between modes improves prediction performance. In contrast, our work considers a scenario in which neither mode is predictive alone, and instead the information contained in both modes must be fused to obtain reasonable performance (e.g., similar to a MUX function). This setting is beginning to be studied by other researchers, with recent [15] usually employing, in our terminology, the mix-in architecture.

III. METHOD

We assume the learner is given examples of the input and output of a function $y = f(x_m, x_s)$ where the image-like main-channel input x_m is a $c \times h \times w$ tensor, in which c is the number of channels of input (e.g. RGB image has 3 channels), and h and w are the height and width of the image, and the side channel x_s is a d-dimensional vector. For example, a collision detection problem may have the main-channel input being an occupancy grid of the environment, stored as a dense image-like data structure, while the side-channel information denotes the robot's start and goal configuration. The output may be binary for classification tasks or real-valued for regression tasks. We are concerned only with problems that obey the following assumptions: 1) $d \ll chw$, and 2) both x_m and x_s are critical in predicting y, and omitting either will lead to a poor result.

Our proposed architectures are designed to adhere to two principles:

1) The structure of x_m is preserved in its original form to exploit the favorable properties of CNNs. In other words, it is fed into the neural network without flattening or other transformation; and
2) The side-channel input is not structured, so the network should treat each input entry symmetrically. This principle ensures that an architecture can be applied to a wide variety of tasks. Although it may be useful to study architectures that exploit the structure of side-channel inputs, this is left for future work.

A. Standard CNNs

A standard CNN architecture (Fig. 1) consists of two stages, each composed of multiple layers. The first is the convolutional stage, in which each layer performs convolution, non-linear transformation, and (optionally) max-pooling. The convolution layer takes a tensor of $c_{in} \times h_{in} \times w_{in}$, and c_{out} kernels with each of $c_{in} \times h_k \times w_k$, and produces an output tensor of $c_{out} \times (h_{in} - h_k + 1) \times (w_{in} - w_k + 1)$. Then a non-linear transformation is applied element-wise to the convolution output. Typical choices include the rectifying linear unit (ReLU), sigmoid, and hyperbolic tangent (tanh) functions. We use ReLU in our training.

Pooling is a down-sizing process that replace every $m \times n$ adjacent block by a function of its entries. We mainly use 2×2 and 3×3 max-pooling, which replaces each 2×2 or 3×3 blocks by its maximum value. Other pooling techniques include average-pooling and softmax-pooling [16].

The chosen values of c_{out}, h_k, and w_k and m, n of the optional pooling step define the structure of each layer in the network, and the parameters of the kernel are learned via backpropagation.

The second stage is fully connected stage. Its input is a flattened output of the convolutional stage. Then each layer computes $h(W \cdot \mathbf{x} + \mathbf{b})$, where h is a non-linear function (also ReLU in our experiment), W is a weight matrix and b is a bias vector, both of which are learned. The structure of the fully connected stage is defined by the number and size of intermediate layers.

For classification, the final output layer minimizes the negative log softmax likelihood loss. For regression, the output layer minimizes mean-squared error.

The discussion above is for CNN with 2D input with c channels. The extension to 3D input is straightforward. Briefly, volumetric input is a 4D tensor with dimensions $c \times l \times w \times h$, with l being the additional dimension, length.

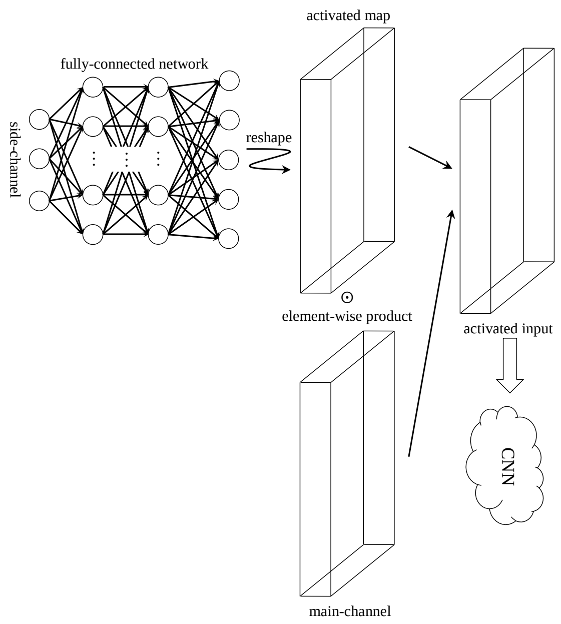


Fig. 2. Activation Map Architecture

Each individual kernel is a 4D tensor swept in a 4D tensor, resulting in a 3D tensor (the channel dimension is collapsed because it is not swept across). Pooling is modified to operate on a cuboid for each channel.

B. Activation Map

The activation map architecture (Fig. 2) makes the assumption that the side-channel information "modulates" the main input in a clearly hierarchical way in that the relevance and usage of main input is determined by the side-channel information.

It is structured to first build an "activation map" which has the same dimensions as the main-channel input. This stage uses a fully connected deep network to learn a transformation from x_s to an "activation list" of length $c \cdot h \cdot w$, which is then reshaped to a $c \times h \times w$ activation map. The element-wise product of the activation map and main-channel input is then fed into a standard CNN to produce the output.

C. Mix-In

The mix-in architecture (Fig. 3) has a standard CNN structure in the convolutional stage and uses only the main-channel input. Immediately after the convolutional stage, the side-channel information is mixed into the flattened neurons and propagates through the fully-connected stage.

D. Stacking

The stacking architecture (Fig. 4) augments the main-channel input by stacking it with d layers of side-channel input, in which d is the dimension of side-channel information. Each layer contains replicated values of x_s for each element of x_m, and there are as many new layers as number of side-channel input sources. Thus, if side-channel sources contain 4 numbers denoting the 2D start and goal position, and the main-channel input is a $1 \times h \times w$ occupancy grid, then the augmented input is of size $5 \times h \times w$.

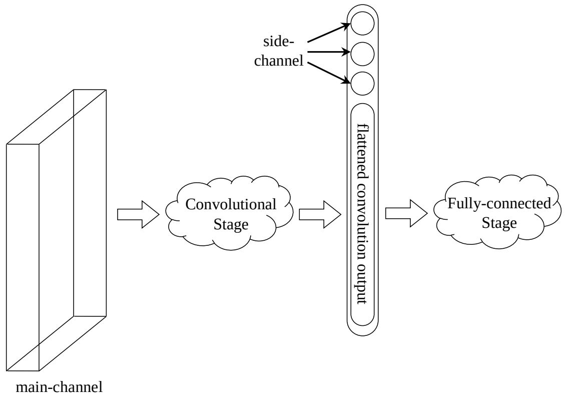


Fig. 3. Mix-In Architecture

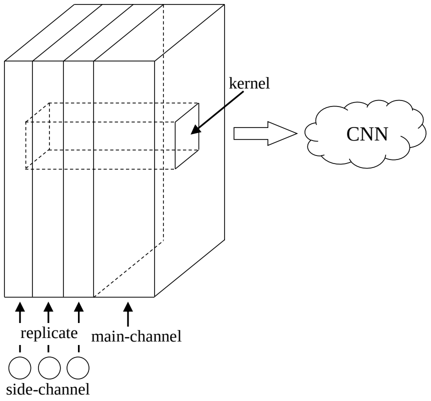


Fig. 4. Stacking Architecture

E. Input-Modulated Kernel

The input-modulated kernel architecture (Fig. 5) uses side-channel information to determine the kernel weights used for convolution. For each layer in the convolutional stage, a fully-connected network maps the side-channel input to the kernel weights. Each kernel is computed from an independent fully-connected network.

F. Training and Implementational Details

To train our model, we use the mini-batch-based Adam optimization algorithm [17] in all examples, and the batch size is selected from 5 to 200 so that the GPU memory is not exhausted (and thus 3D tasks have smaller batch size). We use the Python package Theano [18] to implement our architectures, which uses NVIDIA CUDA Deep Neural Network library (cuDNN) to perform feedforward and back-propagation of convolution operations. Most of the critical computation jobs are done on a NVIDIA GeForce GTX970 with 4GB memory.

IV. TEST PROBLEMS

We designed several toy collision prediction problems so that each architecture can be tested on an "idealized" domain where exact ground truth is available and millions of examples are available for training. Motion feasibility prediction can serve as a primitive operation in motion planners (e.g. collision detection along a line is needed for tree growth

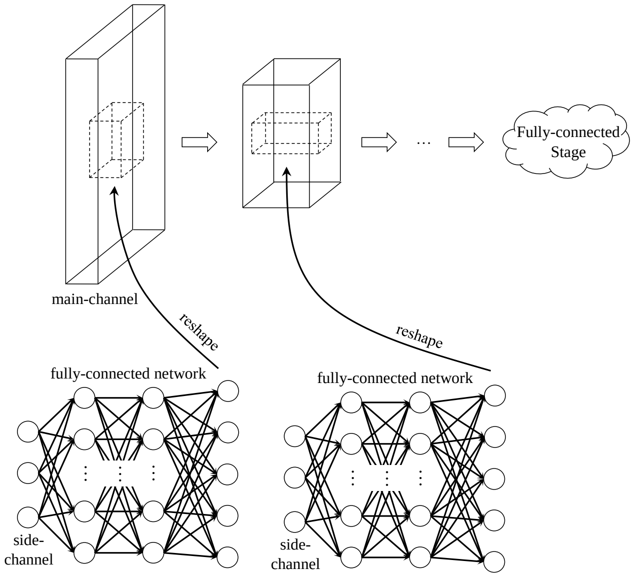


Fig. 5. Input-Modulated Kernel Architecture

Problem	MC	SC	Type	Architecture
2D Block	20 × 20	4	Classification	C3/50-M2-C4/30-M2-F100-F100
2D Indep	20 × 20	4		
3D Block	20 × 20 × 20	6		
Simulation	100 × 100	4	Classification	C15/50-M2-C14/30-M3-C5/20-M2-F100-F100
Col 2D F	100 × 100	2	Regression	C7/50-M2-C6/50-M2-C4/50-M2-F1000-F100
Col 2D V	100 × 100	4		
Col Arm	52 × 52 × 34	1	Regression	C3/50-M2-C(2,2,3)/50-M2-C(3,3,2)/50-C3/50-F1000-F500

TABLE I

SUMMARY OF TEST PROBLEMS. MC MEANS MAIN-CHANNEL DIMENSION; SC MEANS SIDE-CHANNEL DIMENSION.

in RRT), and collision distance/time prediction are required of autonomous vehicle navigation and safety in industrial robotic assembly. Table I summarizes key parameters of each problem. All main-channel inputs are in the form of an occupancy grid.

For the architecture column, "Cx/y" means a convolution layer of with y kernels each of size x, which is a scalar giving side length for equilateral kernel or a tuple giving kernel shape; "Mx" a means max-pooling layer with down-sampling rate of x on each side; "Fx" means a fully-connected layer with x output variables. The output layer is not specified as its shape is fully determined by the number of output variables of the second-to-last layer.

a) Straight-line collision prediction: In 2D Block, 2D Indep, and 3D Block, the task to learn is whether a line segment is collision-free in a grid of obstacles. The side-channel information is the start and end of the segment, comprising 4 numbers in 2D and 6 numbers in 3D. For the 2D cases, start positions and goal positions are randomly selected within the workspace, while for the 3D case, they are selected from top and bottom face respectively. In 2D Block and 3D Block, obstacles are sampled as random rectangles/cuboids, while in 2D Indep, occupied grid cells are sampled independently at random (Fig. 6).

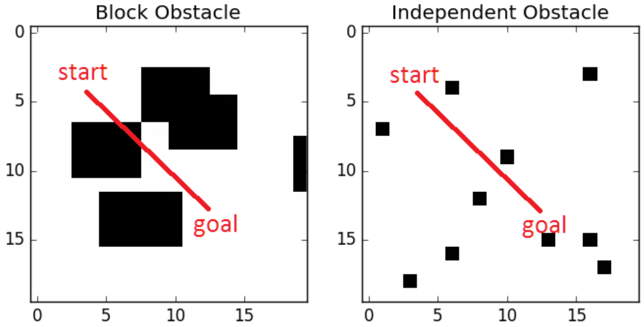


Fig. 6. 2D Block and 2D Indep collision prediction problems: Black denotes obstacles.

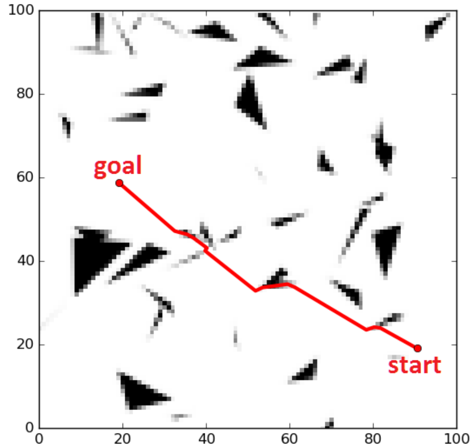


Fig. 7. Simulation problem: the prediction is whether a point mass sliding along obstacles can reach the goal, or settles into a local minimum. Here the connection is successful.

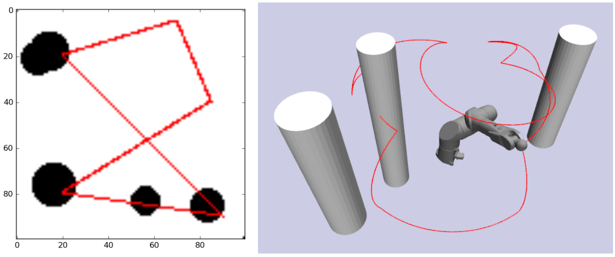
b) Simulation Connectivity Prediction: In Simulation, the task is to predict whether a point mass can move to a goal using compliant motion (Fig. 7). Start and goal positions are generated uniformly at random and random triangular obstacles are generated. Then, a physics simulation is run to move a point mass from start position to goal position, assuming obstacles are fixed and frictionless. The point can slide against obstacles and still reach the goal configuration, or it could be caught in a local minimum.

c) 2D Collision Distance/Time Prediction: In Col 2D, the task is to predict the time at which a point mass, moving in a 2D workspace, would collide with obstacles. Random disk obstacles are scattered in the workspace. Two variations of the problem are considered.

1) Fixed Path (Col 2D F): The robot moves on a fixed path with uniform speed (Fig. 8 (a)). The side-channel input is the current (x, y) position of the robot. Our training and testing sets only maintain obstacle configurations that at least collide with part of the robot path.
2) Variable Path (Col 2D V): The robot moves in a random direction with a uniform speed. The side-

channel input is four numbers (x, y, v_x, v_y) denoting current position and velocity with $v_x, v_y \in \pm[1, 5]$. The output is the time to nearest collision.

In both cases, the distance to nearest collision is calculated by simple geometry.



(a) Col 2D F (b) Col Arm

Fig. 8. Col 2D F and Col Arm problem: path is traced out in red.

d) Arm Collision Distance Prediction: In Col Arm the task is similar to Col 2D Fixed Path, but rather than using a point mass robot, a 3D industrial robot model (Staubli TX90L) is used (Fig. 8 (b)). The robot moves along a fixed path that spans much of its workspace. The obstacles are approximately human-sized cylinders simulating humans walking around the robot. The robot is also included in the occupancy grid, as though it were sensed by cameras.

The robot path is discretized to 4726 steps such that consecutive configurations have very similar but different occupancy grid. The side-channel is simply the index of the configuration. The output to be learned is the number of steps until first collision with an obstacles.

V. RESULTS

A. Summary

Table II shows the testing performance for each architecture on each task, reporting error rate for classification problems and root-mean-squared error for regression. During testing, a random subset of a pool of test problems is selected and error is measured. Then we smooth the test error across iteration using moving average, and the number to average is much less than total number of test iterations so that the initial high error stage does not affect the final test result. The Base column displays error of a baseline CNN trained only on main-channel input. The Range column shows the minimum and maximum possible error. The Kernel architecture trained extremely slowly, and after a day of training time with no sign of convergence, we marked results with T!. For 3D Block and Col 2D V, stacking side-channel input to the original main-channel input will exceed the 4GB memory on the GPU. For these tasks (denoted with *) we stack the side-channel with the output of the first max-pooling layer.

B. Typical results

	Act	Mix-In	Stack	Kernel	Base	Range
2D Block	**1.7%**	5.8%	3.8%	4.6%	27.2%	[0, 100%]
2D Indep	**6.8%**	22.6%	18.3%	T!	33.8%	[0, 100%]
3D Block	**4.4%**	5.4%	4.7%*	T!	25.2%	[0, 100%]
Simulation	**23.9%**	26.8%	35.0%	T!	47.0%	[0, 100%]
Col 2D F	16.8	**14.8**	19.5	T!	72.2	[0, 331]
Col 2D V	**2.20**	2.82	2.25*	T!	5.72	[0, 100]
Col Arm	**84**	139	280	T!	104	[0, 4725]

TABLE II

ERRORS OF EACH ARCHITECTURE ON VARIOUS TASKS. "ACT" COLUMN REPORTS BEST PERFORMANCE AMONG DIFFERENT PARAMETER SETTINGS (NUMBER OF HIDDEN NEURONS IN COMPUTING THE ACTIVATION MAP). "T!" MEANS TRAINING TIME IS TOO LONG TO REACH CONVERGENCE. "*" INDICATES THAT, DUE TO MEMORY ISSUES, THE SIDE-CHANNEL WAS STACKED TO THE OUTPUT OF THE FIRST MAX-POOLING LAYER RATHER THAN THE MAIN CHANNEL.

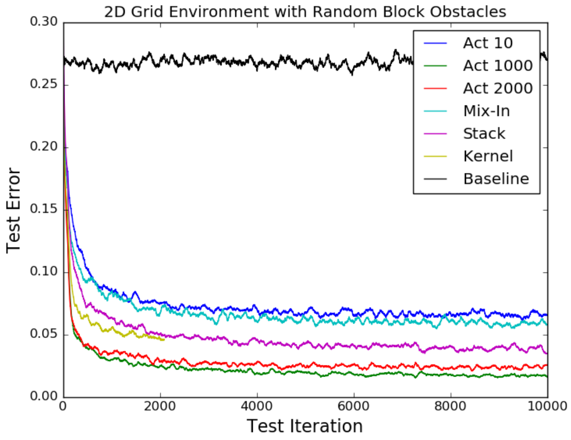
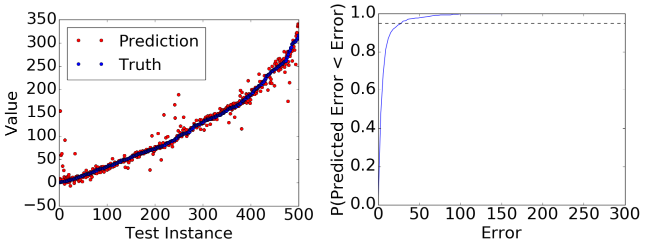


Fig. 9. A typical learning curve, here shown for 2D Block. (Best viewed in color)

A typical learning curve is shown on 2D Block (Fig. 9). Three activation map parameters are tested, with X hidden neurons per layer in the activation stage (Activation X). Activation 1000 performs best, followed by Activation 2000, stack, kernel (stopped at around 2000th iteration due to slow training), mix-in, and Activation 10.

For the 2D Col F regression problem, Fig. 10 plots ground truth vs. prediction for each test instance and a cumulative distribution function (CDF) of error for the mix-in architecture, which performed the best on this task. The prediction is generally close to ground-truth, with 95% of errors less than 28.2 units (<10% of output range).

For the activation map architecture, the map can directly reveal how side-channel information is incorporated into the prediction. In collision detection tasks, the learned activation



(a) Ground truth vs. prediction (b) CDF of errors

Fig. 10. Mix-in architecture test results on Col 2D. (Best viewed in color)

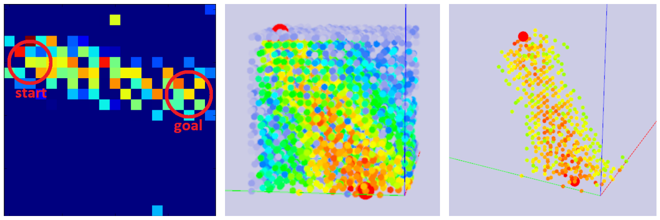


Fig. 11. Activation maps for certain start and end points (indicated as circles) for collision prediction problems. Warmer color indicates higher activation. Left: 2D Blocks. Middle: 3D Block. Right: 500 largest values of the same 3D Block example. (Best viewed in color)

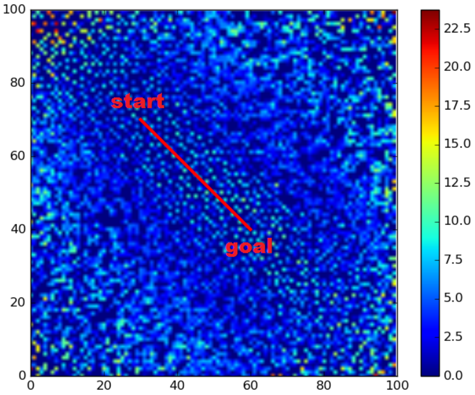


Fig. 12. Activation map for Simulation problem: The activation pattern is not as distinct as collision prediction, because the existence of an obstacle can "steer away" a path from a straight line. (Best viewed in color)

map approximates the swept volume of the robot (Fig. 11). However, it is less interpretable on the Simulation problem (Fig. 12) where relevance is intimately tied with the main-channel input.

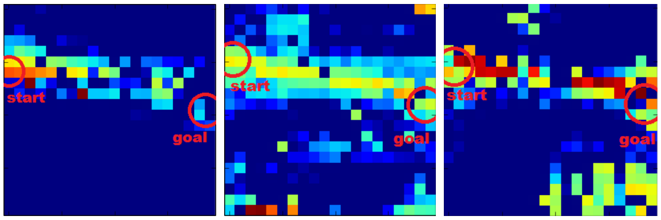
C. Interpretation

In general, we observe the following trends.

1) Activation map performs well all around, but works best when side-channel input solely and uniquely determines the relevance of main-channel input;
2) Mix-in is good when the main-channel input also affects the relevance of itself but may suffer from loss of information due to over-compression in convolutional stage;
3) Stacking generally does not perform as well and scales poorly with the number of side-channel inputs d;
4) Input-modulated kernel takes too long for even moderately-sized problems, and thus is not practical; and
5) Baseline CNNs are usually unable to achieve good performance. The exception is when the side-channel information is "naturally" included in the main-channel information, as observed in the Col Arm task.

The advantages and disadvantages of each architecture are discussed in more detail below.

a) Activation Map: Besides its good performance, an advantage of the activation map is that it is highly interpretable via visualization. It also performs the best when the side-channel input can by itself determine the relevance of main-channel input, e.g., in collision detection where the activation approximates the swept volume (Fig. 11). It performs worse (but still fairly well) on the simulation feasibility problem (Fig. 12) where relevance is intimately tied with the main-channel input. Similarly, for collision time prediction (Fig. 14), it predicts that the whole path is potentially relevant, but an even more minimal activation map could be determined by the configurations of obstacles on the path: if an obstacle appears early on the path, then the remainder is actually irrelevant.



(a) Activation 10 (b) Activation 1000 (c) Activation 2000

Fig. 13. Effect of the number of neurons per hidden layer for activation map. For 2D Indep, (a) 10 neurons are not expressive enough to represent the straight-line swept volume, (b) 1000 neurons per hidden layer gives best performance, and (c) 2000 neurons overfit the training examples, exhibiting activation in the lower right region of the environment regardless of start and goal positions. (Best viewed in color)

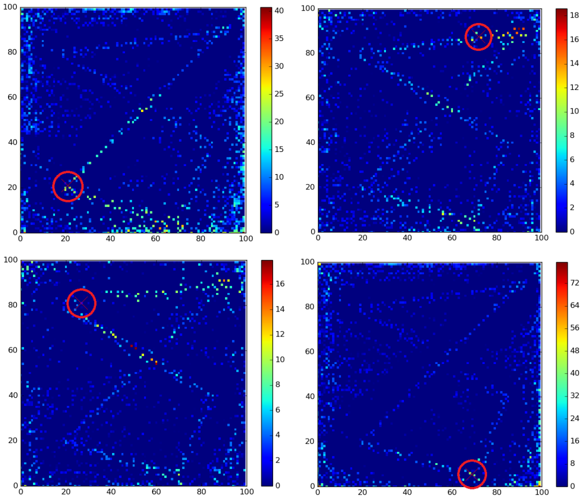


Fig. 14. Activation map for fixed-path 2D collision time prediction task. The entire path is activated because side-channel information alone cannot uniquely determine the relevance of features. Robot current position is circled in red. (Best viewed in color)

There are two main disadvantages of the activation map. First, its performance is relatively sensitive to the chosen number of neurons in the activation stage. Second, the map is computed through a fully-connected network and requires many hidden neurons per intermediate layer (Fig. 13) to induce a flexible-enough function. However, since the number of main-channel inputs is large, learning such a fully-connected network may require a lot of data and be prone to over-fitting (note the extraneous pattern on Fig. 13 (c) and Fig. 14).

b) Mix-In: Mix-in offers the best space efficiency and no tuning requirement, and performs reasonably well. The convolutional stage acts as a feature detector that compresses main-channel input down to a small set of features, which may destroy important information that should be mixed with the side-channel input. For example, in 2D Indep, the environmental variation is quite large, and it is hard for a compressed representation to capture all subtleties of the original environment.

c) Stacking: Space complexity is the biggest problem with stacking, in that it augments the initial $O(chw + d)$ input to $O((c+d)hw)$. It also appears to have poor capacity and/or fall into local minima, performing quite poorly on the 2D Indep, Simulation, and Col Arm examples.

d) Kernel: Training of the input-modulated kernel architecture is prohibitively expensive even for moderately sized problems, and thus its characteristics are not fully studied. The cuDNN library is optimized for computing simultaneous convolution of multiple images with the same kernel. In this architecture, each kernel is dependent on side-channel information, which is different for each training example. Thus, batch-based training cannot be performed in parallel.

VI. Conclusion

This paper empirically compared four candidate architectures for incorporating side-channel information into CNNs. Experiments on simulated collision prediction problems suggest that the Activation Map architecture, which maps the side-channel to a pixel-by-pixel modulation of the image-like main-channel, generally performs well. However, its performance is sensitive to the tuning of the number of neurons per layer, and may be susceptible to overfitting in the presence of huge numbers of main-channel inputs due to the use of fully-connected networks.

To address this latter problem it may be possible to borrow the idea of local receptive field of CNNs, in which spatial coherence is used to gradually reduce the size of the input data. In our case, we can "reverse" the process to gradually build larger and larger maps, until the map has the same size as the main-channel input. This idea makes sense intuitively because an activation map is also likely to exhibit spatial coherence much like the main-channel (i.e. if one position is relevant, its neighbors tend also to be relevant).

More study is needed to assess the quality of the Kernel architecture, which is currently impractical on large problems due to the inability to perform batch convolution. New software and/or hardware support of fast individual convolution should be developed to study this idea further. It may be also possible to improve prediction performance even further with hybrid network structures, such as activation + mix-in. In addition, a more compact stacking layer could be learned from a fully connected network (similar to the activation architecture) and stacked to augment the main-channel input.

References

[1] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3128–3137.

[2] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proc. 28th Intl. Conf. on Machine Learning (ICML)*, 2011, pp. 689–696.

[3] P. Wu, S. C. Hoi, H. Xia, P. Zhao, D. Wang, and C. Miao, "Online multimodal deep similarity learning with application to image retrieval," in *Proc. 21st ACM Intl. Conf. Multimedia*, ser. MM '13. New York, NY, USA: ACM, 2013, pp. 153–162.

[4] J. Pan, S. Chitta, and D. Manocha, "Faster sample-based motion planning using instance-based learning," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 381–396.

[5] N. Jetchev and M. Toussaint, "Trajectory prediction: learning to map situations to robot trajectories," in *Proc. 26th Annual Int. Conf. Machine Learning (ICML)*. ACM, 2009, pp. 449–456.

[6] ——, "Trajectory prediction in cluttered voxel environments," in *Proc. IEEE Intl. Conf. Robotics and Automation (ICRA)*. IEEE, 2010, pp. 2523–2528.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[8] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Trans. Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[11] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "DRAW: A recurrent neural network for image generation," *arXiv:1502.04623*, 2015.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[13] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *Intl. J. Robotics Research (IJRR)*, vol. 34, no. 4-5, pp. 705–724, 2015.

[14] C. Schenck and D. Fox, "Detection and tracking of liquids with fully convolutional networks," in *Proc. Robotics: Science and Systems Workshop on Are the Sceptics Right? Limits and Potentials of Deep Learning in Robotics*, 2015.

[15] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Machine Learning Research (JMLR)*, vol. 17, no. 39, pp. 1–40, 2016.

[16] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition," in *Proc. Interspeech Conference*, 2013, pp. 3366–3370.

[17] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[18] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688