1 Introduction to the problem and methodology

Task of simulations below is to use a LeNet-like neural network to do classification task on CIFAR-10 dataset. The network provided in example file provides an acceptable baseline for further exploration. Experiments are done on choice of hyperparameters including batch size, optimizer, activation function, percentage of neuron dropout, number of convolution kernels and structure of neural network.

1.1 Batch size choice

Common starting point of trying it out is 32, 64, 128. A power of 2 is used due to SIMD paradigm used by GPUs. This is often called Data Parallelism. Since the networks I work on are not that complex, larger batch sizes are also tried out.

1.2 Optimizer

RMSProp and Adam optimizer is experimented. RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster, being an advanced version of basic MBGD; Adam, also integrates the advantage of computing learning rates for each parameter adaptively, is an even more advanced version. Also, it has a better performance on suppressing fluctuation of loss function.

1.3 Activation function

Relu is used as baseline and more advanced elu and selu are tested to see if the problem of dead neuron brought by using relu appears in simulation of the network.

1.4 Dropout

A dropout too small will obviously hinder the ability of generalization of our model, and 0.5 is a common value in many practices, so generally values between 0.125 to 0.5 are chosen. Several value combinations of dropout are tested to see how each of it can affect final accuracy, especially to what extent can it improve validation and test accuracy, as they show generalization ability of the model. When accuracies grow more rapidly, larger value intervals are used; finetuning is done among values where higher accuracies are gained and increasing speed of accuracies start to go down.

1.5 Number of convolution kernels

Intuitively, number of convolution kernels represents the number of features we can learn. Basically, larger the number we set; somehow more complete we can represent a picture. However, when the number gets too large, almost no new features can be newly picked by extra filters but more burdens are added for training. For this lab, value between 4 and 150 are experimented, considering the input size of 32*32.

2 Contextualization of deep learning methodologies within the state of the art of deep learning for robotics

2.1 Robotics vision

deep learning is widely involved in robotics vision, and Ruiz-del-Solar et al. [1] gave a rough categorization of its application, namely Object Detection and Categorization, Object Grasping and Manipulation, Scene Representation and Classification, and Spatiotemporal Vision.

2.1.1 Object Detection and Categorization

Tao et al. [2] used Mask-RCNN for target detection and instance segmentation when building their robot for VSLAM. Firstly, an ORB algorithm is used to extract image feature points. Secondly, a RANSAC algorithm is employed to eliminate mismatched points and estimate camera position-pose changes. Then, the Mask RCNN is applied to make partial adjustments to its hyper parameter.

Jia et al. [3] depends heavily on neural network in his research on detecting and segmenting overlapping fruits. ResNet combined with DenseNet can reduce input parameters and is used as a backbone network for feature extraction. Feature maps are input to the RPN for end-to-end training to generate the RoI, and finally the mask is generated by an FCN to get the region where the apple is located.

Bogun et al. [4] use LSTM to get motion aid object recognition in short videos. They implement each gate of LSTM as a convolution and show that convolutional-based LSTM models are capable of learning motion dependencies and are able to improve the recognition accuracy when more frames in a sequence are available. The whole system consists of merely the NN, 3 continuous frames are used as input and object label as output.

2.1.2 Object Grasping and Manipulation

Redmon et al. [5] used CNN for predicting grasp coordinates. A five-dimensional representation for robotic grasps proposed gives the location and orientation of a parallel plate gripper before it closes on an object. Recognition and grasp detection is combined to build a task pipeline. A self-proposed NN based on Alexnet is used for the whole pipeline.

Sung et al. [6] present a novel approach to manipulation planning based on the idea that many household objects share similarly-operated object parts. They formulate the manipulation planning as a structured prediction problem and learn to transfer manipulation strategy across different objects by embedding point-cloud, natural language, and manipulation trajectory data into a shared embedding space using a deep neural network. In order to learn semantically meaningful spaces throughout our network, they introduce a method for pre-training its lower layers for multimodal feature embedding and a method for fine-tuning this embedding space using a loss-based margin.

2.1.3 Scene Representation and Classification

Maeda et al. [7] used a Time-Contrastive Network to compare current state of a task and its state of completion. This distance is represented as a metric of distance-to-go and the metric is used for describing distance towards the goal of the task.

Cadena et al. [8] explore the capabilities of Auto-Encoders to fuse the information available from cameras and depth sensors, and to reconstruct missing data, for scene understanding tasks. They seek to generate complete scene segmentations and depth maps, given images and partial and/or noisy depth and semantic data and formulate this objective of reconstructing one or more types of scene data using a Multi-modal stacked Auto-Encoder.

2.1.4 Spatiotemporal Vision

Hwang et al. [9] utilized deep learning in vision part of a vision-based learning from demonstration (LfD) system to train a mimetic robot. In this system, YOLO network is used for object recognition, while multiaction recognition is carried out by an Inflated 3D ConvNet (I3D) deep learning network followed by a proposed statistically fragmented approach to enhance the action recognition results.

Yin et al. [10] presents a nonlinear time alignment method with deep autoencoder. The spatio-temporal features obtained from the neural network contain the metric information for feature comparison.

2.2 Robot tactile sensing

Melnik et al. [11] use deep reinforcement learning (DRL) for object manipulation. Tactile information from sensors on robot hands are used as input in addition to visual information. An Actor and Critic network is used as network architecture.

Lepora et al. [12] use poseNet (a CNN designed by them) to estimate pose 3D pose from optical tactile image and they see hyperparameters also as normal ones needed to be tuned, which is this first time the technique is used in tactile information.

2.3 Robot motion & navigation

Wang et al. [13] utilized deep Q-network to learn implicit control policy for robot motion planning in their research. They modeled the robot as a labeled Markov decision process (MDP) with continuous state and action spaces. Linear temporal logics (LTL) are used to specify high-level tasks. We then train deep neural networks to approximate the value function and policy using an actor-critic reinforcement learning method. The LTL specification is converted into an annotated limit-deterministic Buchi automaton (LDBA) for continuously shaping the reward ¨ so that dense reward is available during training.

Liang et al. [14] also use deep reinforcement learning in their research of collision avoidance, and their network architecture is also basically a CNN. Their simulator models realistic crowd and pedestrian behaviors, along with friction, sensor noise and delays in the simulated robot model. They also describe a technique to incrementally control the randomness and complexity of training scenarios to achieve better convergence and generalization capabilities. A novel reward function is proposed for the NN training.

3. Simulations

The method of describing NN is partially borrowed from [15]. Hyperparameters in red are those to be player around in each test, those in blue will change only within configurations that gave best results in the previous round. Green results are best ones in current test and their hyperparameter configuration will be used for the next round of simulation.

3.1 batch size & optimizer & activation function

Batch size = (batch size), Optimizer = (optimizer)

Input: 32*32

C1: f.maps 32@32*32 (Activation function) C2: f.maps 32@32*32 (Activation function)

S3: f.maps 32@16*16-Dropout=0.25

C4: f.maps 64@16*16 (Activation function) C5: f.maps 64@16*16 (Activation function)

S6: f.maps 64@8*8-Dropout=0.25

F7: layers 512-Dropout=0.5

OUTPUT:10

1-Activation function = relu

Accuracy (train, val, test) in % @epoch num / Optimizer / batch size	RMSprop			Adam		
1024	96.10	78.35	76.55@120	95.89	78.82	78.72@100
512	92.10	77.86	77.36@60	94.67	79.49	78.78@75
256	88.88	79.09	77.93@45	94.30	79.95	79.14@65
128	82.24	72.53	78.08@50	93.13	80.31	79.76@60
64	75.20	75.07	75.19@12	91.97	79.02	78.72@50
32	68.33	70.92	70.27@6	88.41	78.67	78.84@50

2-Activation function = elu

Accuracy (train, val, test) in % @epoch num / Optimizer / batch size	RMSprop			Adam		
1024	95.95	78.77	78.58@110	96.02	78.96	78.56@120
512	95.51	78.55	78.69@90	95.36	79.01	78.36@90
256	94.92	77.55	77.53@90	94.99	78.31	77.81@90
128	93.50	77.45	76.68@85	93.85	77.37	77.21@90
64	91.11	76.58	75.61@85	92.19	76.28	76.84@90
32	85.41	75.14	74.68@85	88.50	75.41	74.52@50

3- Activation function = selu

Accuracy (train, val, test) in % @epoch num / Optimizer / batch size	RMSprop			Adam		
1024	93.02	70.30	70.75@120	94.17	75.80	74.96@250
512	94.24	75.57	75.13@115	95.26	76.43	76.06@210
256	93.00	75.02	74.31@90	94.50	76.13	75.11@160
128	91.40	74.56	74.63@85	92.49	76.34	74.99@145
64	88.10	74.67	74.03@100	89.67	74.63	74.29@90
32	81.20	72.74	71.89@100	84.37	72.85	71.99@50

3.2 dropout

Batch size = (batch size), Optimizer = Adam

Input: 32*32

C1: f.maps 32@32*32 (Activation function=relu) C2: f.maps 32@32*32 (Activation function=relu)

S3: f.maps 32@16*16-Dropout=a

C4: f.maps 64@16*16 (Activation function=relu) C5: f.maps 64@16*16 (Activation function=relu)

S6: f.maps 64@8*8-Dropout=b

F7: layers 512-Dropout=c

4- Testing on different dropout values

Accuracy (train, val, test) in % @epoch num val of a,b,c \ batch size	128			256			512		
0.0625, 0.0625, 0.125	99.11	76.08	75.43@80						
0.125, 0.125, 0.25	98.08	75.88	74.98@70						
0.375, 0.375, 0.5	91.38	81.16	79.66@70						
0.4, 0.4, 0.4	93.26	81.43	80.35@110	94.25	81.46	79.99@110	94.24	80.75	79.67@105
0.4, 0.4, 0.5	91.42	81.82	80.30@110	91.71	81.67	80.59@110	92.77	81.80	81.33@120
0.45, 0.45, 0.45	91.87	82.16	80.94@110	90.12	82.26	80.90@105	92.63	81.88	80.67@115
0.45, 0.45, 0.5	90.33	81.77	81.29@120	90.55	81.85	81.23@120	91.67	81.55	80.59@115
0.5, 0.5, 0.5	87.13	81.20	80.63@120	88.48	81.84	81.63@115	88.82	81.93	80.23@115
0.6, 0.6, 0.6	69.41	67.61	66.79@120						

3.3 number of convolution filters

Batch size = (batch size), Optimizer = Adam

Input: 32*32

C1: f.maps w@32*32 (Activation function=relu) C2: f.maps 32@32*32 (Activation function=relu)

S3: f.maps x@16*16-Dropout=a

C4: f.maps y@16*16 (Activation function=relu) C5: f.maps 64@16*16 (Activation function=relu)

S6: f.maps z@8*8-Dropout=b

F7: layers 512-Dropout=c

5- Testing on different numbers of convolution filters (part 1)

Accuracy (train, val, test) in % @epoch num val of w,x,y,z \ other config	batch size=128 a=0.45, b=0.45, c=0.45			batch size=128 a=0.45, b=0.45, c=0.5			batch size=512 a=0.4, b=0.4, c=0.5		
4, 4, 4, 4				53.05	57.00	56.98@55			
8, 8, 8, 8				62.73	66.97	66.43@85			
16, 16, 16, 16				63.14	69.37	68.46@70			
32, 32, 32, 32				85.18	80.01	79.36@110			
64, 64, 64, 64				91.77	82.40	81.37@135			
80, 80, 80, 80				92.61	82.53	81.95@120			
100, 100, 100, 100	93.44	82.53	82.23@120	93.27	82.93	82.38@130	95.52	82.65	81.68@125
128, 128, 128, 128	95.08	82.78	81.93@140	94.86	83.07	82.73@135	96.32	83.27	81.78@125
150, 150, 150, 150	95.24	83.31	82.78@120	94.42	83.23	82.29@120	95.24	83.03	82.00@110

6- Testing on different numbers of convolution filters (part 2)

Accuracy (train, val, test) in % @epoch num val of w,x,y,z \ other config	batch size=256 a=0.45, b=0.45, c=0.45			batch size=256 a=0.45, b=0.45, c=0.5			batch size=256 a=0.5, b=0.5, c=0.5		
100, 100, 100, 100	95.48	82.82	82.39@125	94.09	83.17	82.76@130	93.42	83.64	82.93@140
128, 128, 128, 128	95.24	83.22	82.18@105	95.19	83.38	82.63@120	94.77	83.97	82.99@140
150, 150, 150, 150	96.11	83.56	82.86@120	94.96	84.04	83.50@150	94.47	83.87	83.02@140

4. Description, interpretation and assessment of the results

Initial accuracy of lab example is train-82.24%, validation-79.14%, test-78.95%, after parameter tuning, they improved to train-94.96%, validation-84.04%, test-83.50% respectively, on average around 4%~5% of accuracy improvement is gained, proving that hyperparameter tuning is indeed meaningful in improving results.

For this network architecture, Adam optimizer always gain a better result than RMSprop regarding validation and testing accuracy, this can give A primary conclusion that Adam optimizer is more suitable for the network (or other similar networks),

yet the conclusion cannot be generalized. The pre-knowledge that Adam and RMSprop integrate more techniques when doing back propagation neither can give reference like “of course Adam and RMSprop can outperform primitive MBGD on evert network” so further tests still need to be done with MBGD and other optimizers selected.

When batch size increases, accuracy will first increase and training time of each epoch goes down. For this network the best values are 128, 256 and 512 as they give highest results. Furthering increase it will no longer benefit the result obviously r cause accuracy to drop-as the network worked with selu. The tendency of “first increase then decrease” of accuracy when increasing batch size needs to be validated under other network configurations.

For activation function, relu works better than elu and selu for this network. Elu and selu are originally proposed to solve the problem of dead neurons, yet in this network this problem did not occurred, and it can be inferred that linear activation works better than nonlinear ones here (elu and selu has non-linear shape when x>0). Sigmoid, tanh and softsign can still be used with batch normalization in CNN and this is also further possible experiments to do.

Increasing dropout value will give better results on validation and testing accuracy while decrease training accuracy, or to say, prevent overfitting. However, when dropout value gets too large (maybe especially when dropout value near input layer gets too large, as too few information is passed into the network), all 3 accuracies starts to decrease. Further experiments are expected to validate the guess.

Regarding number of convolution filters, increasing the number can improve accuracy, as more information from the input is extracted. However, when it reaches a certain value (it may depend on input size, information complexity of the picture and target of the task), further increase will give no extra improvement to the result. Further experiments shall be done on different input size to check if the guess is correct.

Reference

[1] Ruiz-del-Solar, Javier, Patricio Loncomilla and Naiomi Soto. “A Survey on Deep Learning Methods for Robot Vision.” *ArXiv* abs/1803.10862 (2018): n. pag.

[2] C. Tao, Z. Gao, J. Yan, C. Li and G. Cui, "Indoor 3D Semantic Robot VSLAM Based on Mask Regional Convolutional Neural Network," in IEEE Access, vol. 8, pp. 52906-52916, 2020.

[3] Jia, Weikuan & Tian, Yuyu & Luo, Rong & Zhang, Zhonghua & Lian, Jian & Zheng, Yuanjie. (2020). Detection and segmentation of overlapped fruits based on optimized mask R-CNN application in apple harvesting robot. Computers and Electronics in Agriculture. 172. 105380. 10.1016/j.compag.2020.105380.

[4] Ivan Bogun, Anelia Angelova, Navdeep Jaitly. Object Recognition from Short Videos for Robotic Perception. arXiv:1509.01602v1 [cs.CV] 4 Sep 2015

[5] Joseph Redmon, Anelia Angelova. Real-Time Grasp Detection Using Convolutional Neural Networks. arXiv:1412.3128v2 [cs.RO] 28 Feb 2015

[6] Jaeyong Sung, Seok Hyun Jin, Ian Lenz, and Ashutosh Saxena. Robobarista: Learning to Manipulate Novel Objects via Deep Multimodal Embedding. arXiv:1601.02705v1 [cs.RO] 12 Jan 2016

[7] Maeda, Guilherme & Vaatinen, Joni & Yoshida, Hironori. (2020). Visual Task Progress Estimation with Appearance Invariant Embeddings for Robot Control and Planning.

[8] Cadena, C., Dick, A.R., & Reid, I.D. (2016). Multi-modal Auto-Encoders as Joint Estimators for Robotics Scene Understanding. *Robotics: Science and Systems*.

[9] P. Hwang, C. Hsu and W. Wang, "Development of a Mimic Robot—Learning From Demonstration Incorporating Object Detection and Multiaction Recognition," in IEEE Consumer Electronics Magazine, vol. 9, no. 3, pp. 79-87, 1 May 2020.

[10] Xiaochuan Yin and Qijun Chen. Deep Metric Learning Autoencoder for Nonlinear Temporal Alignment of Human Motion. IEEE International Conference on Robotics and Automation (ICRA). May 2016.

[11] Melnik, A., Lach, L., Plappert, M., Korthals, T., Haschke, R., & Ritter, H. Tactile Sensing and Deep Reinforcement Learning for In-Hand Manipulation Tasks.

[12] Lepora, Nathan & Lloyd, John. (2020). Optimal Deep Learning for Robot Touch.

[13] Wang, C., Li, Y., Smith, S. L., & Liu, J. (2020). Continuous Motion Planning with Temporal Logic Specifications using Deep Neural Networks. arXiv preprint arXiv:2004.02610.

[14] Jing Liang, Utsav Patel, Adarsh Jagan Sathyamoorthy, Dinesh Manocha. “CrowdSteer: Realtime Smooth and Collision-Free Robot Navigation in Dense Crowd Scenarios Trained using High-Fidelity Simulation” *ArXiv* abs/2004.03089 (2020): n.

pag.
[15] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.