Lab 3: Dynamic Convolutional Filter Networks for CIFAR-10 Classification

Originally from Yicong Hong, modified by Shidong Pan

September 18, 2021

1 Introduction

The goal of this lab is to introduce you a state-of-the-art idea called the Dynamic Filter Networks, which has been widely applied in many machine learning models.

Quote from the original Dynamic Filter Networks paper:
"*In a traditional convolutional layer, the learned filters stay fixed after training. In contrast, we introduce a new framework, the Dynamic Filter Network, where filters are generated dynamically conditioned on an input. We show that this architecture is a powerful one, with increased flexibility thanks to its adaptive nature, yet without an excessive increase in the number of model parameters.*"
— Jia, Xu, et al. "Dynamic filter networks." [1]

Intuitively speaking, consider there is a convolutional network trained for extracting image features. Then, each convolutional layer will have a number of learned kernels (filters) where each kernel can extract certain features from the image. However, for some kernels, e.g. kernels which can detect round-shape objects, they are likely to provide much more useful information for an input image "Car" than a "Pyramid". Because for a "Car", those kernels could tell the sizes, positions and textures of the wheels, but for a "Pyramid", those kernels can only inform the network about the absence of a round-shape object. Now, think about this: is it possible if we provide some prior knowledge about the input to the convolutional network, so that the network can learn to apply more suitable kernels for certain inputs, therefore obtain more informative image representations and benefit the downstream tasks?

In this lab, we are going to implement a very simple dynamic convolutional layer in a classification network, please read and follow the instructions carefully to complete the lab tasks. If you want to have a deeper understanding of the knowledge behind the lab tasks, please refer to the papers in the reference list. Or use Google Scholar to search for papers that apply dynamic filter networks.

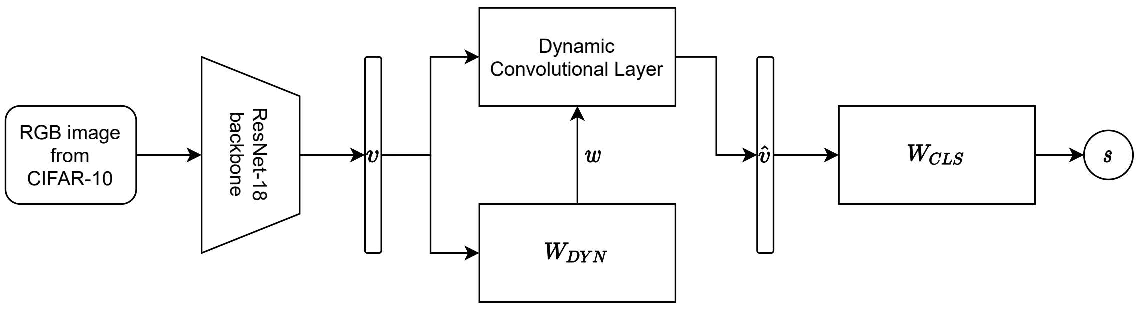

Figure 1: Network schematic.

The network that you are going to build in this lab is shown in Figure 1. In brief, the network takes images from the CIFAR-10 Dataset as input. It encodes the images using the first several layers of a pre-trained ResNet-18 (backbone) and a 3-by-3 dynamic convolutional layer (DC). The weights $\boldsymbol{w}$ for the dynamic convolutional layer are generated by a non-linear network W_{DYN}, which is different for each different input image. Finally, the encoded image features $\hat{\boldsymbol{v}}$ is fed to a linear network W_{CLS} to produce the classification score $\boldsymbol{s}$ for each class.

Note that there are 10 classes in the CIFAR-10 Dataset, and the order is as follows: 0: plane, 1: car, 2: bird, 3: cat, 4: deer, 5: dog, 6: frog, 7: horse, 8: ship, 9: truck. We are going to use PyTorch torchvision to load the dataset, you can also find the dataset in our shared data folder in StuGPU2 server or download it directly online.

2 Task 1: Implement the Dynamic Convolutional Network (10 marks)

In this task, you are asked to implement a Dynamic Convolutional Network presented in the Introduction section. Bear in mind that it is important to maintain your code quality to a high standard. You may find many excellent examples in PyTorch Tutorial.

2.1 Fill in the code for the BaseModel() class in model.py (4 marks)

For the BaseModel() initialization: (2 marks)

1. Initialize a pretrained ResNet-18 model from torchvision.models, however, only use the model up to (Layer 1 - BasicBlock 1) as your Backbone.

   Hint: You can print out the ResNet-18 model to see how each layer is named. Be careful, do not freeze the weights for the Backbone (i.e. **MUST NOT** set `params.requires_grad=False`).

2. Initialize the weights generator network W_{DYN} as FC - Tanh, where FC maps input Dim4096 to Dim576, Tanh is the Tanh activation function.

3. Initialize the Dynamic Convolutional (DC) layer as a 3-by-3 2D convolutional layer which maps 64 input channels to 1 output channel with stride 1 and padding 1.

4. Initialize the classification network W_{CLS} as a single FC layer which maps input Dim64 to output Dim10.

For the forward path: (2 marks)

1. Pass the input image to the backbone convolutional network to get feature map $\boldsymbol{v}$, and conduct a sanity check ($\boldsymbol{v}$ should be dimension of BatchSize $\times$ 64 $\times$ 8 $\times$ 8).

2. Flatten the feature map $\boldsymbol{v}$ and feed it to the weights generator network W_{DYN} to obtain the dynamic network weights $\boldsymbol{w}$.

3. For each sample i in the mini-batch, pick out its dynamic network weights $\boldsymbol{w_i}$, and apply L2-normalization on $\boldsymbol{w_i}$.

4. Set the weights of the dynamic convolutional layer (DC) as the normalized and reshaped $\boldsymbol{w_i}$, $\boldsymbol{w_i}$ after reshaping should be of size 1 $\times$ 64 $\times$ 3 $\times$ 3. You might need to search in PyTorch Documentation or Pytorch Forums about how to do this.

5. Feed the feature map $\boldsymbol{v_i}$ to the DC layer to get the final image representation $\boldsymbol{\hat{v}_i}$.

6. Reshape $\boldsymbol{\hat{v}_i}$ and pass it to the W_{CLS} layer to get the classification scores $\boldsymbol{s_i}$ for each class.

7. Repeat steps 3 - 6 for all samples in the mini-batch.

2.2 Fill in the missing code in train.py as indicated by the comments (1.5 mark)

1. Save the checkpoints. For each evaluation(), you need to save the states of the model and the optimizer whenever you trained a better network (e.g. whenever the new testing accuracy is higher than the previous best testing accuracy). (0.5 marks)

2. In regard to saving the PyTorch model, compare a) saving all learnable parameters of the model and b) saving the whole model. (0.5 marks)
3. Fill in the code for resume() to load the saved model and optimizer, so that you can resume your network to a state with the highest performance anytime, and continue training or run testing. (0.5 marks)

   Hint: You may need the deepcopy() function.

2.3 Train the network and compare to without dynamic filter (4.5 marks)

1. Train the network which you have built, record the training and testing statistics (any information that you think can best reflect the training process and the result). (1.5 marks)
2. Compare the training stats to the same network without the dynamic filters, i.e. stop using the weights generator network W_{DYN} and directly train the 3-by-3 convolutional layer (which applies the same kernels on all the inputs), and explain your findings. (1.5 marks)
3. Based on the better structure between with or without the dynamic filters, can you further improve the performance? You need to specifically state what kind of modification do you make, how your modification is performed, and how it impacts results. (1.5 mark)

Please Note: For 2.3, marks are awarded here for presentation and explanation. For example, simply showing screen shots without explanation is not adequate.

3 Task 2: Understand the Dynamic Convolutional Network (5 marks)

3.1 Visualize the dynamic kernels

As shown in Figure 2, we visualize the weights of the 64 kernels of the dynamic convolutional layer, which is produced by the weights generator network W_{DYN} for three different plane images. To be specific, each red box is a 3-by-3 kernel, the order of the kernels in three plots are the same.

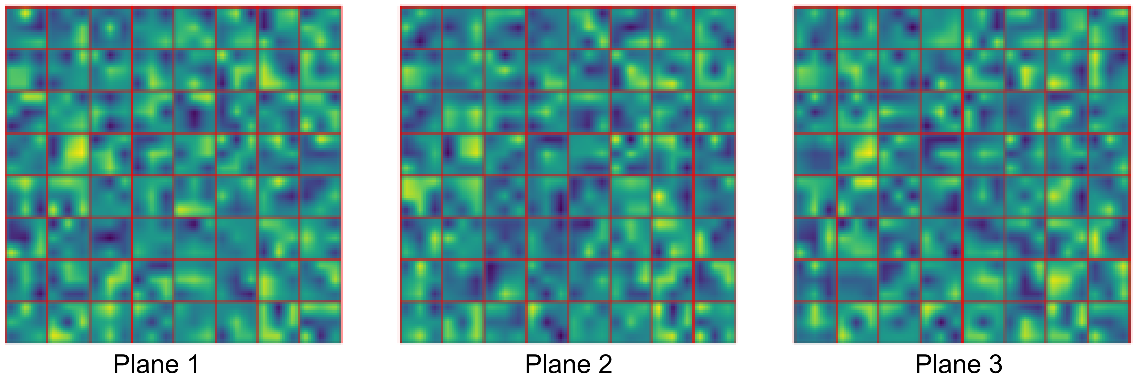

Figure 2: Visualization of the dynamic convolutional kernels.

1. What can you observe by comparing the three sets of kernels? Can you see some similarity among them? Explain your findings. (2 marks)
2. Visualize the kernels of some input images from different classes using your own network. Compare the visualizations and explain your findings. (2 marks)

3.2 Discussion

Please Note: For 3.2, you only need to select one question to answer. Please clearly indicate which one you choose.

1. [1] also introduces the idea of dynamic local filtering layer. Assuming you are asked to apply the Dynamic Local Convolutional Network to replace the Dynamic Convolutional Network, to conduct an image classification task, please draw a pipeline graph or a flow chart to specifically illustrate the main task stream. (1 mark)

   Hint: You can find two excellent examples in Appendix (Section 5) as your reference. Also, there are many other great "art" works that you can find on top conferences such as CVPR, ICCV, etc.

2. Recall that in Lab2, you were asked to:
   Perform the following data augmentation during training: randomly flip the image left and right with 50% probability; zero-pad 4 pixels on each side of the input image and randomly crop 224x224 as input to the network.

   Will data augmentation (flip, rotate, crop, etc.) process contribute to better performance in this task? Justify your answer. (1 mark)

4 Submission

1. The lab will be due on Sunday, 6 Oct 2021, 6:00pm. Please submit a lab report that clearly documents all the experimental results and answers to the questions, and attach all your code in Appendix. Name your submission as Lab_3_u0000000.pdf, replacing u0000000 with your uni-ID.

2. The full grade of this lab is 15 marks. This Lab worth 7.5% of the total course assessment.

References

[1] Xu Jia et al. "Dynamic filter networks". In: *Advances in neural information processing systems.* 2016, pp. 667–675.

[2] Dongxu Li et al. "Tspnet: Hierarchical feature learning via temporal semantic pyramid for sign language translation". In: *arXiv preprint arXiv:2010.05468* (2020).

[3] Shi Qiu, Saeed Anwar, and Nick Barnes. "Semantic Segmentation for Real Point Cloud Scenes via Bilateral Augmentation and Adaptive Fusion". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2021, pp. 1757–1767.

5 Appendix

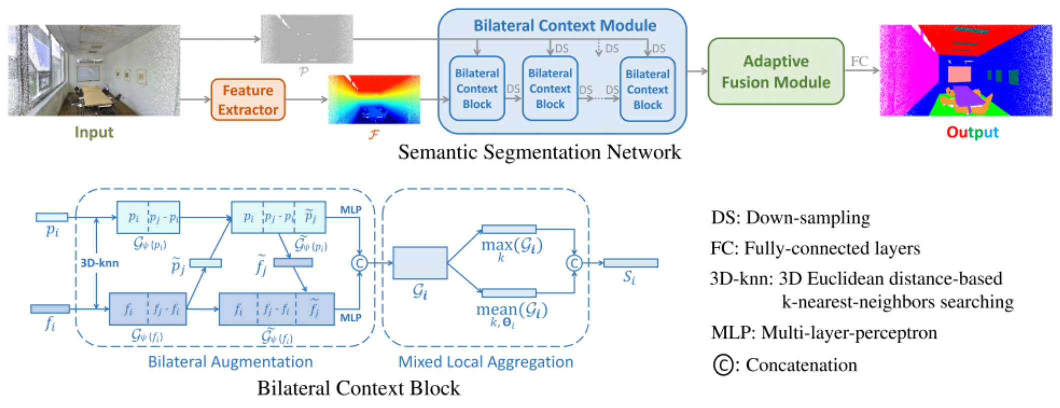

Figure 2: The details of our semantic segmentation network and the Bilateral Context Block (the annotations are consistent with the items in Sec. 3.1). Firstly, the Feature Extractor (Sec. 4.1) captures the preliminary semantic context $\mathcal{F}$ from the input data. Then, the Bilateral Context Module (*i.e.*, a series of the Bilateral Context Blocks) augments the local context of multiple point cloud resolutions. Generally, the Bilateral Context Block requires both semantic and geometric context as bilateral inputs. In particular, the first block inputs the preliminary $\mathcal{F}$ and the original 3D coordinates $\mathcal{P}$; while each of the others inputs its previous one's downsampled output and coordinates $\mathcal{P}$, as the semantic and geometric context respectively. Afterward, our Adaptive Fusion Module (Sec. 3.2) upsamples the Bilateral Context Blocks' outputs, then adaptively fuses them as an output feature map. Finally, we predict semantic labels for all points via fully-connected layers.

Figure 3: A figure from [3].

qiu2021semantic

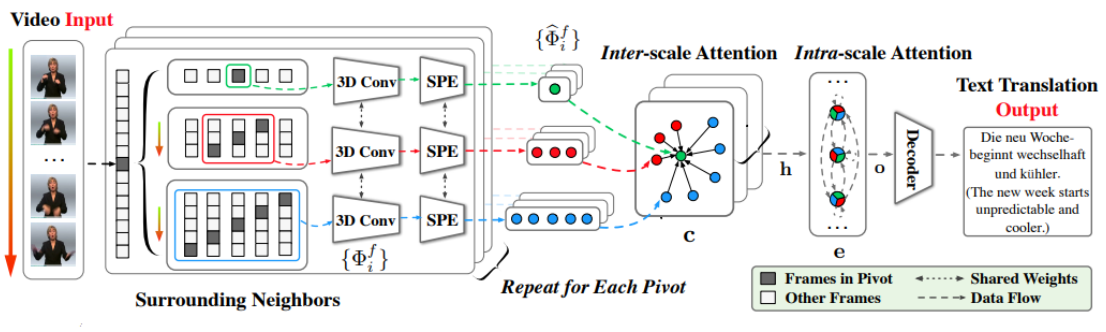

Figure 1: Overview of the workflow of our proposed TSPNet, which generates spoken language translations directly from sign language videos.

Figure 4: A figure from [2].