Deep Learning – HW2

劉仁忠 RE6125015

Github: https://github.com/Kragsane/HW2/tree/main

I. Task: Designing a Convolution Module for Variable Input Channels

Dynamic convolution is an advanced neural network technique that enhances model adaptability and efficiency by allowing convolutional layers to adjust their weights dynamically based on the input data. Unlike traditional convolutional layers that utilize fixed weights once trained, dynamic convolution layers generate weights in real-time during inference, tailoring their responses to the specific features of each input image. This method offers the potential for greater accuracy and generalization by enabling the network to focus on pertinent image attributes and adapt to varying data distributions. It's particularly useful in applications where the input characteristics can vary significantly, such as in image recognition tasks across different lighting conditions or camera angles. Dynamic convolution thus represents a significant shift towards more flexible and context-aware neural networks.

In this task focused on designing dynamic convolution layers, I have developed four variations (v1, v2, v3, v4) to explore the concept of dynamic convolution, using ResNet18 as the foundational model. Each of these layers is engineered to adjust its weights according to the number of input channels, thereby enabling the selection of channels and generation of convolutional weights tailored to the incoming data. This customization enhances the model's ability to process relevant image features effectively. **Each of these layers modifies ResNet18's initial convolution layer into a dynamic one to illustrate the practicality of this layer.** Below is an overview of each layer:

- v1: Implements masking to exclude non-essential channels, selecting only relevant channels for which it dynamically generates weights.
- v2: Instead of creating weights for all possible channels, this version produces weights for each specific subset size. It uses a channel mask to select relevant channels, applying this mask to dynamically generate weights.
- v3: Incorporates the dynamic convolution layer described in “Dynamic Convolution: Attention over Convolution Kernels” by Wu et al., CVPR 2020, leveraging Attention2D and Dynamic2D for a dynamic weight-generation network.
- v4: Similar to v2 but includes a mechanism to randomly drop channels with a controlled probability. This approach encourages the model to learn more robust features by not consistently relying on specific channels or features, thus fostering the development of redundant pathways and enhancing generalization.

Training and validation phases are governed by an early stopping criterion, meaning the number of epochs may vary across models. **For detailed metrics such as precision, recall, F1-score, and class support, as well as confusion matrices and plots of training and validation accuracy and loss, please refer to the accompanying “.ipynb” file.**

Batch size: 64, input image size: (84, 84, 3)

Computational costs:

Model	#PARAMS (M)	FLOPS (GFLOPS)	Memory Usage (Mb)
ResNET18	11.1945	135.887	2026
v1	11.1997	502.252	3510
v2	11.2043	135.889	1888
v3	11.1927	-	1880
v4	11.2043	135.889	1882

#PARAMS and FLOPS were count using ***calflops*** packages provided in *https://github.com/MrYxJ/calculate-flops.pytorch/tree/main*

Experiment results (chosen as best model from early stopping):

Model	# of epochs	Training Acc (%)	Training Loss	Validation Acc (%)	Validation Loss	Testing Acc on RGB (%)
ResNET18	15	63.32	1.1510	59.11	1.4622	54.89
v1	12	60.74	1.2705	53.78	1.5691	56.67
v2	12	59.91	1.2820	48.67	1.7130	54.00
v3	13	65.85	1.0578	51.33	1.6708	52.00
v4	18	61.74	1.2377	48.67	2.0200	46.44

As shown in the table above, version 1 (v1) of the dynamic convolution layers result in higher performance, achieving a 56.67% accuracy, which is a 2% increase over the baseline ResNet18 model at 54.89%. However, v1 has a limitation: it can only infer using the same channel configuration it was trained on. For example, if it is trained with RGB images, it is restricted to inferencing with RGB as well. It cannot process different channel combinations (such as RG, GB, R, G, B) without being retrained for each specific configuration. This limitation prompted the development of additional versions (v2, v3, and v4), allowing for the exploration of how different channel combinations affect performance without needing retraining. The testing accuracies for models v2, v3, and v4, which utilize various channel combinations, are detailed below.

Testing accuracy for v2, v3, and v4 model:

Model	RGB	RG	RB	GB	R	G	B
v2	54.00	1.78	2.00	2.67	2.00	3.11	2.89
v3	52.00	4.89	4.00	3.78	2.89	2.00	2.44
v4	**46.44**	**26.89**	**31.78**	**28.89**	**15.11**	**17.78**	**19.11**

Ablation Studies and Analysis

In our analysis, both versions 2 (v2) and 3 (v3) demonstrated similar performances, which were notably weaker when utilizing different channel combinations during inference. Specifically, accuracy dropped significantly in v2 from 54% to around 2-3% and in v3 from 52% to approximately 3-4% with different channel combinations. Although these versions are capable of inferencing with various channel configurations without the need for retraining, the resultant accuracy is considerably low, highlighting their limitations.

This observation led to the development of version 4 (v4), which incorporates random channel dropping during training to enhance the model's robustness across diverse channel combinations. Despite v4 achieving a lower testing accuracy in RGB at 46.44%, it shows

relatively better performance in handling 2-channel and 1-channel inferences, with accuracies of about 29% and 18%, respectively. This outcome is consistent with the logic behind random channel dropping; by occasionally omitting channels during training, the model does not overly depend on any specific channels, thereby learning to extract useful features from whatever channels are available. This method promotes a more versatile and resilient approach, making v4 an intriguing option for scenarios requiring flexibility in channel usage.

II. Designing a Two-to-Four-Layer Network for Image Classification

In this task, I designed a network with 2 to 4 layers aimed at image classification, employing a CNN architecture that integrates advanced techniques such as RRDB, self-attention mechanisms, and SE blocks. The network comprises six layers, in addition to the input and output layers. Here's a breakdown of the operations within each layer:

- Layer 1: Initial Convolution + BatchNorm + ReLU
- Layer 2: SelfAttention + SEBlock
- Layer 3: RRDB (3 Convolutions + SelfAttention + SEBlock + Residual Connection)
- Layer 4: Transition Convolution + BatchNorm + ReLU
- Layer 5: Adaptive Average Pooling
- Layer 6: Fully Connected Layer

The **SEBlock**, or Squeeze-and-Excitation Block, is a module designed to improve the representational power of a neural network by adaptively recalibrating channel-wise feature responses. It works in two main stages: squeeze and excitation. During the squeeze stage, global information is aggregated using global average pooling, which condenses each channel into a single value. In the excitation stage, these values are passed through a small neural network consisting of two fully connected (FC) layers with a ReLU activation followed by a Sigmoid activation. This network generates weights for each channel, which are then multiplied with the original feature maps to scale each channel's importance, effectively allowing the network to focus on more informative features.

The **SelfAttention** mechanism is designed to capture long-range dependencies in the data by computing the relevance of one part of the input with respect to another. It transforms the input feature map using three convolutional operations to create query, key, and value matrices. The attention scores are calculated by performing a dot product between the query and key matrices, followed by a softmax operation to normalize the scores. These attention scores are then used to weigh the value matrix, allowing the network to selectively focus on important features from the entire input. The result is a new feature map that integrates contextual information across the entire input, which is particularly useful for tasks requiring understanding of complex relationships in the data. **In the given code, this is combined with an SEBlock to further enhance feature recalibration.**

The **RRDB**, or Residual in Residual Dense Block, is an advanced neural network architecture that integrates the principles of both residual learning and dense connections. Within an RRDB, multiple convolutional layers are densely connected, meaning each layer receives inputs from all preceding layers, promoting feature reuse and improving gradient flow. Additionally, the entire dense block is embedded within a residual connection, where the input is added to the output of the block after being scaled, ensuring that the gradient can flow back through the entire block during training. This structure helps in capturing and preserving both fine-grained

and high-level features. **Furthermore, the RRDB in the given code also includes a self-attention mechanism and SEBlock to enhance the network's ability to focus on relevant features and adaptively recalibrate channel-wise feature responses.**

The idea behind layer 2 and layer 3 are counted as one layer:

Layer 2: SelfAttention + SEBlock

The SelfAttention mechanism combined with the SEBlock is considered one layer because they collectively form a single functional unit designed to enhance feature representations in a cohesive manner. The SelfAttention mechanism itself involves multiple steps: it computes the query, key, and value matrices, calculates the attention scores, and applies these scores to the value matrix to generate contextually enriched features. Adding the SEBlock to this process further refines these features by adaptively recalibrating the importance of each channel. The SEBlock performs global average pooling followed by a two-layer fully connected network that outputs scaling factors for each channel, which are applied to the feature map to highlight important features and suppress less relevant ones. Despite the multiple operations involved, **they are tightly integrated to perform a single, unified task of enhancing and recalibrating feature representations**, thereby fitting the definition of a single layer as they rely on each other to complete this task.

Layer 3: RRDB (Residual in Residual Dense Block)

The RRDB is considered one layer because it encapsulates a complex but cohesive set of operations that together improve feature learning and gradient flow in a deep network. Within an RRDB, multiple convolutional layers are densely connected, meaning that each layer takes inputs from all previous layers within the block. This dense connectivity promotes feature reuse and ensures robust gradient propagation. Additionally, the entire block is wrapped in a residual connection, where the input is added to the block's output, allowing the gradient to flow back directly through the identity mapping, which stabilizes training. Furthermore, the inclusion of self-attention and SEBlock mechanisms within the RRDB enhances its ability to focus on important features and recalibrate channel responses. **All these operations are designed to work together synergistically**, contributing to a single, unified goal of enhanced feature extraction and representation. Therefore, despite the internal complexity, the RRDB is considered a single layer because it functions as a cohesive unit.

In this task, the evaluation is segmented across three models. The first model is ResNet34, serving as the baseline. This is followed by two variants, v1 and v2, both of which incorporate the aforementioned layer configuration. **The primary distinction between v1 and v2 lies in the parameters of the first Conv2D layer**. Specifically, v1 utilizes a convolution with a kernel size of 3, a stride of 1, and padding of 1, while v2 employs a larger kernel size of 7, a stride of 2, and padding of 3.

Training and validation phases are governed by an early stopping criterion, meaning the number of epochs may vary across models. **For detailed metrics such as precision, recall, F1-score, and class support, as well as confusion matrices and plots of training and validation accuracy and loss, please refer to the accompanying ".ipynb" file.**

Batch size: 16, input image size: (84, 84, 3)

Computational costs:

Model	#PARAMS	FLOPS (GFLOPS)	Memory Usage (Mb)
ResNET34	21.3026 M	70.1987	3096
v1	205.204 K	46.0907	1998
v2	212.884 K	11.6577	1702

#PARAMS and FLOPS were count using ***calflops*** packages provided in *https://github.com/MrYxJ/calculate-flops.pytorch/tree/main*

Experiment results (chosen as best model from early stopping):

Model	# of epochs	Training Acc (%)	Training Loss	Validation Acc (%)	Validation Loss	Testing Acc on RGB (%)
ResNET34	15	57.22	1.3813	52.67	1.6410	51.78
v1	41	41.75	2.0264	46.00	1.9551	44.67
v2	57	48.22	1.7928	45.11	1.8645	50.00

In my experiments, where the goal was to design a two-to-four-layer network that could achieve 90% of the performance benchmark set by ResNet34, the results indicate that this objective was nearly met (from above table). ResNet34, serving as the baseline, achieved a testing accuracy of 51.78% over 15 epochs, demonstrating rapid convergence. Our custom models, v1 and v2, despite their simplified architectures, made considerable progress towards this goal. Model v2, in particular, reached a testing accuracy of 50.00% after 57 epochs, which is approximately 96.5% of the performance of ResNet34. This performance is noteworthy, especially considering that v2 was designed with a more straightforward layer structure compared to the deeper ResNet34. The number of epochs required for v2 suggests a slower convergence rate, likely due to the model adapting and optimizing its fewer layers to approach the complexity of tasks handled by ResNet34. This finding emphasizes the potential of smaller, efficiently designed networks to nearly match the performance of more complex architectures, offering a promising avenue for environments where computational resources or model interpretability are key constraints.

It's important to highlight that while model v2 requires more epochs to converge compared to v1, indicating a slower convergence rate, it not only achieves higher accuracy but also operates more efficiently. Specifically, v2 completes its training within approximately 5 hours to reach 57 epochs, whereas v1 takes nearly 24 hours to reach 41 epochs. This efficiency in v2 can be attributed to its larger kernel size, which, despite processing fewer details per operation compared to the smaller kernel size of v1, does not compromise on achieving higher accuracy. This suggests that a larger kernel size can effectively balance computational speed and performance, making v2 a more efficient choice in scenarios where both time and accuracy are critical.

References

1. ‘Dynamic Convolution: Attention over Convolution Kernels’ by Wu et al., CVPR 2020.
2. ‘ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks’ by Wang et al., ECCV 2018.
3. ‘Squeeze-and-Excitation Networks’ by Hu et al., CVPR 2018.
4. ‘Attention Is All You Need’ by Vaswani et al., NeurIPS 2017.