Deep Learning Assignment II

Chia-Ming Lee
Institute of Data Science,
National Cheng Kung University
zuw408421476@gmail.com

Abstract

This document provides a detailed description of the assignment for the Deep Learning class (113-2 semester) at the Institute of Data Science, National Cheng Kung University. Our code and pre-trained models are available at https://github.com/ming053l/DL-Assignment-2.

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

1.1. Description

Design a special convolutional module that is spatial size invariant and can handle an arbitrary number of input channels. You only need to design this special module, not every layer of the CNN. After designing, explain the design principles, references, additional costs (such as FLOPS or #PARAMS), and compare with naive models. To simulate the practicality of your method, use the ImageNet-mini dataset for training, and during the inference process, test images with various channel combinations (such as RGB, RG, GB, R, G, B, etc.) and compare the performance.

1.2. Design Principles

Dynamic Convolution: Design a dynamic convolution module that can adjust its weights based on the number of input channels. This can be achieved by learning a weight-generating network that takes the input channels as an input and generates corresponding convolution kernels.

Implementation Details:

- Convolution Module: Use a dynamic weight-generating network. This network receives the number of input channels and generates convolution kernels accordingly. These kernels can then be used for standard convolution operations.
- Reference: The concept of dynamic convolution can be referenced from the paper 'Dynamic Convolution: Attention over Convolution Kernels' by Wu et al., CVPR 2020. Cost Analysis: Compared to naive models, this design might increase additional computational costs (such as FLOPS) and parameters but can significantly reduce the need for training and storing models for different channel numbers.

Experiment Design: Training and Inference: Use the ImageNet-mini dataset for training and test the model on images with various channel combinations during inference.

Comparison: Compare the performance of the model using the dynamic convolution module with naive models across different input channel combinations, evaluating accuracy and computational cost.

2. Task II: Designing a Two-Layer Network for Image Classification

2.1. Description

Design a (2-4)-layer CNN, Transformer, or RNN network that can achieve 90% performance of ResNet34 on ImageNet-mini (i.e., with no more than 10% performance loss). There are no restrictions on parameter count or FLOPS, but the maximum number of input and output layers is limited to 4-6. Explain the design principles, references, and provide experimental results. We suggest you DO NOT use pre-trained models for ResNet34.

2.2. Design Principles

Increasing Receptive Field: Use techniques such as RRDB, self-attention mechanisms, or Graph Convolutional Networks (GCN) to increase the network's receptive field and enhance feature extraction capabilities.

Implementation Details:

- RRDB Module: Implement a Residual-in-Residual Dense Block (RRDB) to increase the receptive field and enhance feature extraction.
- Reference: The design concept can be referenced from 'ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks' by Wang et al., ECCV 2018.
- Attention Mechanism: Incorporate self-attention mechanisms to improve the network's ability to capture global

features.

- Cost Analysis: This design might increase computational costs (such as FLOPS) and parameter count but can improve performance within the constraint of limited layers.

Experiment Design:

- Training and Inference: Use the ImageNet-mini dataset for training and evaluate the network's performance on image classification tasks.
- Comparison: Compare the designed two-layer network with a ResNet34 trained from scratch on ImageNet-mini, evaluating accuracy and computational cost.

Additional Tips:

- Increasing Receptive Field: Use any block to enhance the network's receptive field.
- Attention Mechanism: Incorporate self-attention mechanisms to improve the capture of global features.
- GCN: Use Graph Convolutional Networks to handle structured information in images and enhance feature extraction.

3. Solution I

3.1. Model Design Concept

The provided code outlines the design of a modified DenseNet model for classification tasks, incorporating a dynamic convolutional layer. Below are the main components:

Dynamic Convolutional Layer: This custom layer, DynamicConv2d, dynamically generates weights for convolution based on the input. It uses a small neural network to generate these weights, allowing the model to adapt more flexibly to various inputs.

Modified DenseNet: The model is based on DenseNet-121, a popular convolutional neural network known for its dense connectivity pattern. The first convolutional layer of DenseNet-121 is replaced with the DynamicConv2d layer to enhance the model's ability to handle variable input characteristics.

Channel Shuffling: To enhance the robustness of our model and prevent attacked data (resulted from different shuffled channel attack), we use channel shuffling augmentation during training time, thereby enlarging source domain to make our model robust.

3.2. Optimizer and Hyperparameters

Optimizer: The model uses the Adam optimizer (optim.Adam), which is well-suited for training deep learning models due to its adaptive learning rate capabilities.

Learning rate: 0.001

Loss Function: The loss function used is Cross-Entropy Loss, which is standard for multi-class classification problems.

Training Hyperparameters:

Batch size: 16 for training, 1 for validation

Number of epochs: 200 for deep learning models.

3.3. Data Augmentation

Data augmentation techniques are employed to enhance the training data and improve the model's robustness. The code uses the transforms module from torchvision to apply various transformations:

Training Data Transformations:

- Random Resized Crop: Randomly crops a part of the image and resizes it to the specified size.
- Random Horizontal Flip: Flips the image horizontally with a probability of 0.5.
- Channel Shuffling: Randomly changes the input channel of the image.
- Random Rotation: Rotates the image randomly within a specified degree range.
- Normalize: Normalizes the image tensor using the given mean and standard deviation values.

Validation Data Transformations:

- Resize: Resizes the image to a specified size.
- Center Crop: Crops the center part of the image to a specified size.
- Channel Shuffling: Randomly changes the input channel of the image.
- Normalize: Same normalization as applied to the training images.

4. Solution II

4.1. Model Design Concept

Integrating Recurrent Neural Networks (RNNs) with Residual-in-Residual Dense Blocks (RRDBs) can significantly enhance the ability of a model to process and reuse feature maps effectively. Therefore, we can archeive good performance even with fewer parametes compared with ResNet34.

Residual-in-Residual Dense Block for stablizing information flow: Provides dense feature extraction and residual connections that help in capturing intricate details and maintaining gradient flow during training.

Recurrent Convolution for shallow feature map enhancement: Adds the capability to improve shallow convolutional layer, making it possible to exploit more important feature dependencies in the features extraction.

DenseBlock: Within each RRDB, multiple dense layers are used where each layer receives input from all preceding layers. This allows for comprehensive feature extraction and reuse.

4.2. Advantages of Integration

- Enhanced Feature Reuse: The dense connections in RRDBs ensure that features are reused effectively, leading to richer representations.

- Temporal/Spatial Context: The RNN component captures dependencies across the feature maps, adding context that can improve performance in tasks requiring sequential understanding.
- Improved Training Stability: Residual connections help in mitigating the vanishing gradient problem, while the RNN's gating mechanisms further stabilize training.

4.3. Optimizer and Hyperparameters

Optimizer: The model uses the Adam optimizer (optim.Adam), which is well-suited for training deep learning models due to its adaptive learning rate capabilities.

Learning rate: 0.001

Loss Function: The loss function used is Cross-Entropy Loss, which is standard for multi-class classification problems.

Batch size: 16 for training, 1 for validation.

Number of epochs: 200 for deep learning models.

4.4. Data Augmentation

Data augmentation techniques are employed to enhance the training data and improve the model's robustness. The code uses the transforms module from torchvision to apply various transformations:

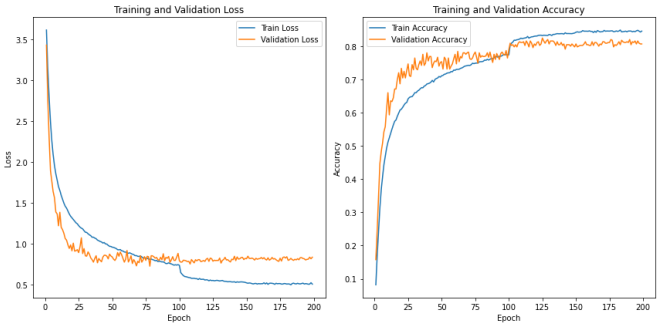
Training Data Transformations:

- Random Resized Crop: Randomly crops a part of the image and resizes it to the specified size.
- Random Horizontal Flip: Flips the image horizontally with a probability of 0.5.
- Channel Shuffling: Randomly changes the input channel of the image.
- Random Rotation: Rotates the image randomly within a specified degree range.
- Normalize: Normalizes the image tensor using the given mean and standard deviation values.

Validation Data Transformations:

- Resize: Resizes the image to a specified size.
- Center Crop: Crops the center part of the image to a specified size.
- Channel Shuffling: Randomly changes the input channel of the image.
- Normalize: Same normalization as applied to the training images.

By combining the dense feature extraction capabilities of RRDBs with the sequence modeling power of RNNs, this integrated approach can significantly enhance the model's ability to handle complex tasks, providing both detailed feature reuse and contextual understanding.



(a) DWGNet

(b) DWGNet with Channel Shuffling

Figure 1. The training/validation loss and accuracy with and without Channel Shuffling Augmentation for Task I.

5. Experiment Results

5.1. Ablation Studies and Results for Task I

The use of Channel Shuffling Augmentation led to a substantial increase in accuracy, from 65.78% to 76.00%. This demonstrates the augmentation's effectiveness in improving the model's ability to generalize better to unseen data. (Abitrary shuffled channel input.)

The augmentation technique significantly improved the model's performance across all evaluated metrics. This highlights the importance of data augmentation strategies in enhancing model robustness and accuracy.

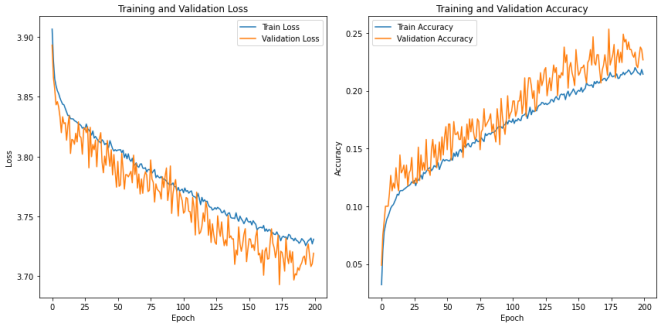
5.2. Ablation Studies and Results for Task II

The RRDB and RRDB-RNN models demonstrate that it is possible to achieve high performance with a significantly reduced number of parameters compared to the ResNet34 model. The RRDB-RNN model, in particular, shows that the integration of RNNs can provide substantial performance gains with a modest increase in parameters.

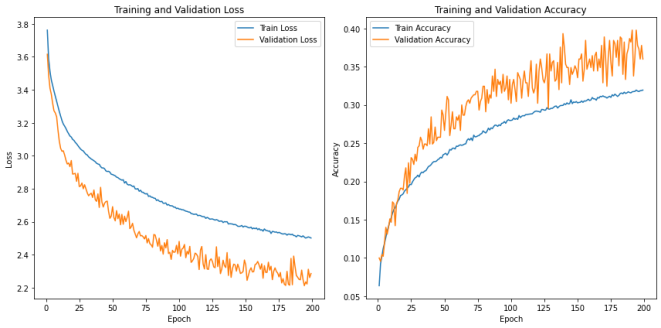
The substantial improvements in accuracy, precision, recall, and f1-score of the RRDB-RNN model over both the ResNet34 baseline and the RRDB-only model indicate the effectiveness of combining dense and recurrent architectures.

Model	Params	Multi-Adds	Forward	FLOPs	Accuracy	Precision	Recall	f1-score
DWG with DenseNet-121	32.84M	14.13G	174.19M	2.65G	65.78%	69.79%	65.78%	65.47%
DWG with DenseNet-121*	32.84M	14.13G	174.19M	2.65G	76.00%	76.61%	76.00%	75.20%

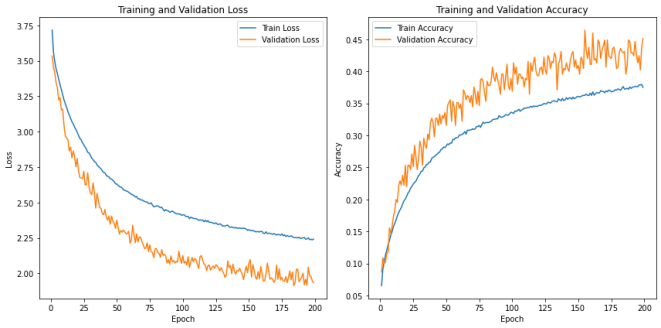
Table 1. Performance and complexity comparison of dynamic weight-generating networks with DenseNet-121 on mini-ImageNet. (* means Channel Shuffling Augmentation is used.)



(a) ResNet34



(b) RRDB



(c) RRDB-RNN

Figure 2. The training/validation loss and accuracy for Task II.

Model	Params	Multi-Adds	Forward	FLOPs	Accuracy	Precision	Recall	f1-score
ResNet34	90.16M	3.66G	59.81M	3.41G	26.67%	11.81%	26.67%	15.83%
RRDB	0.17M	0.20G	20.07M	0.19G	35.11%	44.25%	35.11%	34.35%
RRDB-RNN	0.30M	0.42G	20.07M	0.39G	42.67%	46.73%	42.67%	41.33%

Table 2. Performance and complexity comparison of different models on mini-ImageNet.