Deep Learning 2024 HW2

資訊所 P76124752 莊上緣

Github: shangyuan191/DL2024_HW2_DCM (github.com)

1. 題目介紹

給定 mini-ImageNet 資料集，我們需要用在這個資料集上完成以下兩個 task:

Task1: Designing a Convolution Module for Variable Input Channels

Task2: Designing a Four-Layer Network for Image Classification

2. 資料集介紹

Mini-ImageNet 資料集內共有 50 個種類的照片，且給定了三個 txt 檔，分別標註了訓練集、驗證集、測試集的每張圖片路徑以及對應的 label。數量如下:

train.txt:63325 張圖片

val.txt:450 張圖片

test.txt:450 張圖片

3. Task1: Designing a Convolution Module for Variable Input Channels

題目描述:

設計一個卷積模塊，該模塊可以處理變動的 input channel，也就是說，我們設計的卷積模塊在面對 input channel 為 RGB、RG、GB、R、G、B 的圖片時，都能正常運作並且因應這些變動的 input channel 動態地調整模塊內部的 data pipeline，使模型能更加 robust 與具備泛化性

思路:

觀察了資料集內的照片分布後，我發現大部分的照片都是 RGB，僅有非常少數的照片是只有 1 個 channel，因此我會先建好一個基礎的 CNN 模塊當成 baseline model，並分別從圖片前處理的角度與設計 layer 的角度來提出方法並進行實驗。

Baseline Model:三層 CNN

將訓練集所有圖片都 reshape 成(3,224,224)，如果遇到只有單個 channel 的圖片，就額外堆疊兩張大小與數值相同的 tensor

Method 1:process_channel_DIY(基於前處理的方式，對訓練集、驗證集、測試集都進行相同處理)

寫好一個 target_channel list 搭配迴圈，包含[RGB,RG,GB,R,G,B]，對每個狀況都進行前處理並訓練一個模型(除了前處理方式不同之外，都是丟進 baseline CNN model，僅根據 target_channel 來改動 CNN input layer 的 input channel 數量)，前處理方式也是先將所有圖片都 reshape 成(3,224,224)，如果遇到只有單個 channel 的圖片，就額外堆疊兩張大小與數值相同的 tensor，接著，根據 target_channel 來保留對應的 channel 數值，其餘的 channel 內的像素就直接將數值變成 0

Method 2:process_channel_sigmoid(基於前處理的方式，對訓練集、驗證集、測試集都進行相同處理)

與 method 1 大同小異，唯一的差別是不使用單純的將 target_channel 以外的 channel 都變為 0，而是根據圖片原有的 channel 數與 target channel 之間的差距，使用 F.sigmoid 來平滑地調整並補齊或砍掉數值

Method 3:PoolingChannelAttention (基於 pooling 與注意力機制的方式)

此方式為在 baseline CNN model 前面加上一個基於 pooling 與 attention 機制的卷積模塊，先使用 average pooling+max pooling 對 input 進行降維，接著加一層 MLP 來處理 pooling 後的特徵，這個方式就不需要透過前處理刻意改變 channel 數量，而是可以自適應地面對不同的 input channel 數量，並通過注意力+pooling 機制將 input channel 都變成 1

通用的前處理:

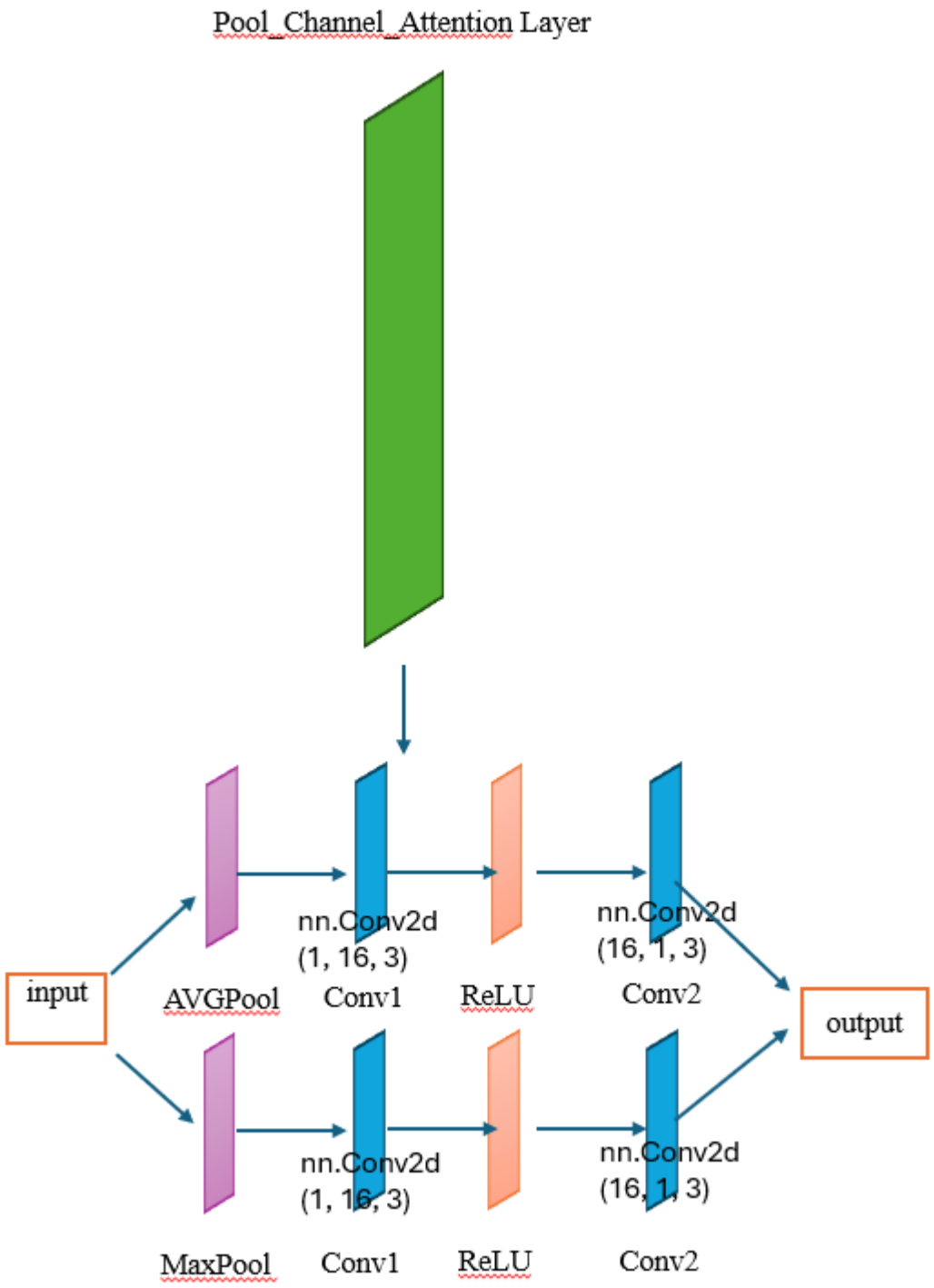
Image augmentation:

```
v2.RandomResizedCrop(size=(224, 224), antialias=True),
v2.RandomHorizontalFlip(p=0.5),
v2.ToDtype(torch.float32, scale=True),
v2.Normalize(mean=mean, std=std),
v2.ToDtype(image_dtype, scale=True)
```

HyperParameter:

Batch_size=64,Num_workers=16,Num_epochs=30,Lr=0.0005

PoolingChannelAttention 架構圖:



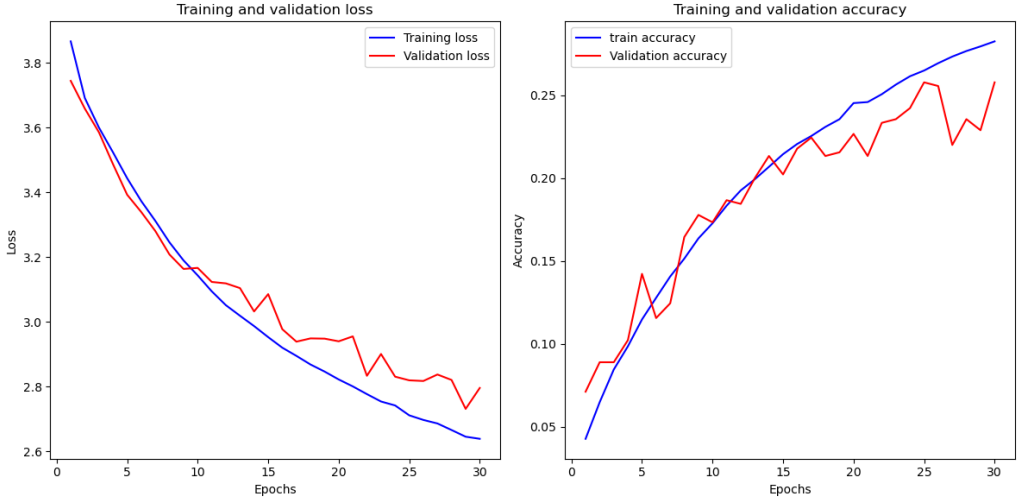
實驗結果:

Test accuracy

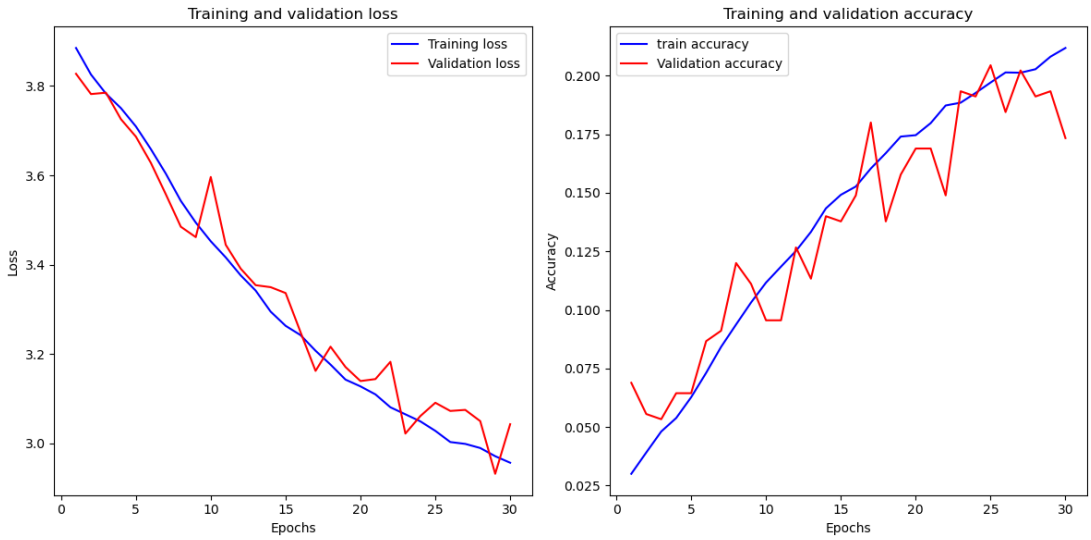
	RGB	RG	GB	R	G	B
Baseline model	0.29	0.06	0.04	0.02	0.03	0.02
Channel_processing_DIY	0.25	0.32	0.28	0.27	0.26	0.24
Channel_processing_Sigmoid	0.09	0.06	0.09	0.04	0.07	0.1
Pool_Channel_Attention	0.2	0.16	0.16	0.13	0.12	0.13

模型訓練過程 plot:

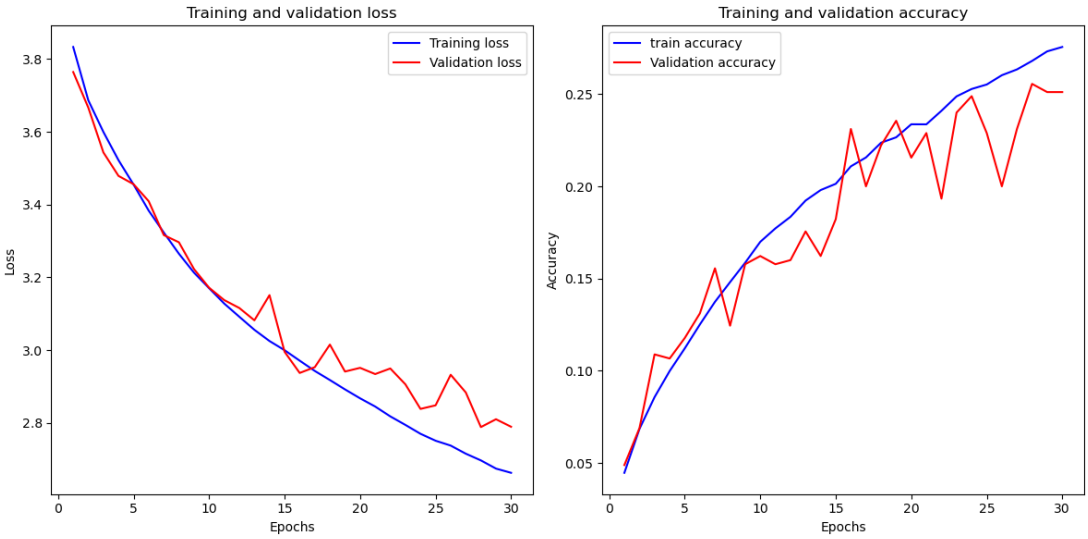
Baseline CNN model



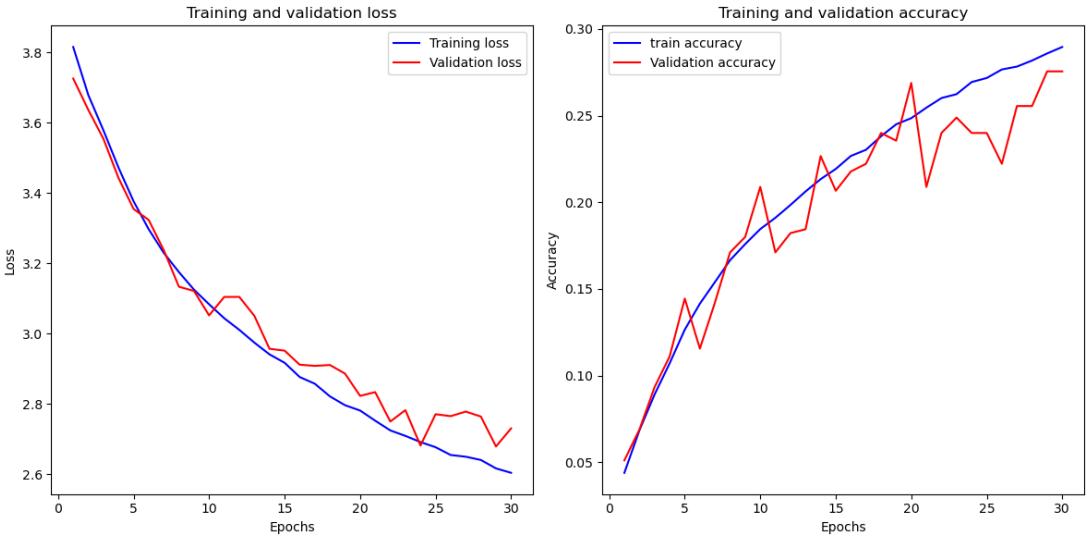
Pool Channel Attention model



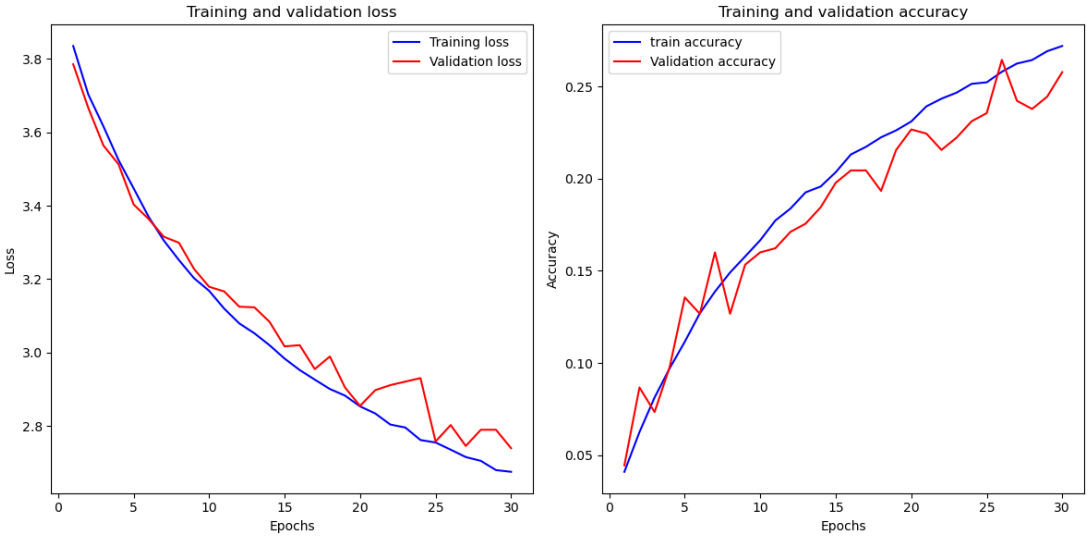
Channel_processing_DIY model in RGB



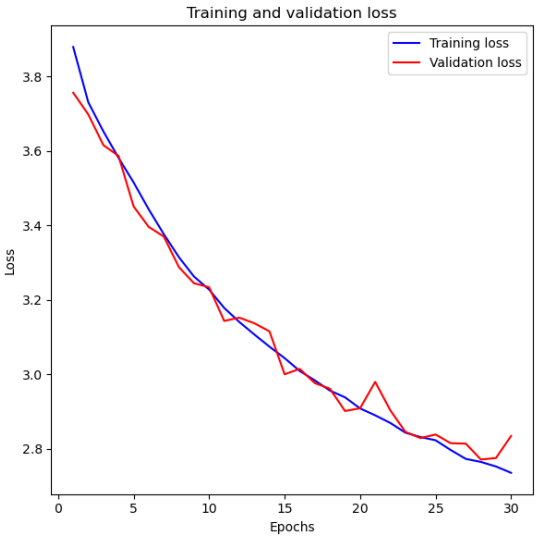
Channel_processing_DIY model in RG

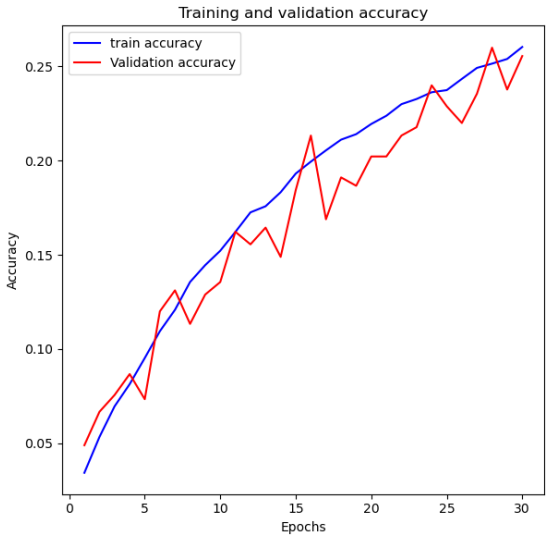


Channel_processing_DIY model in GB

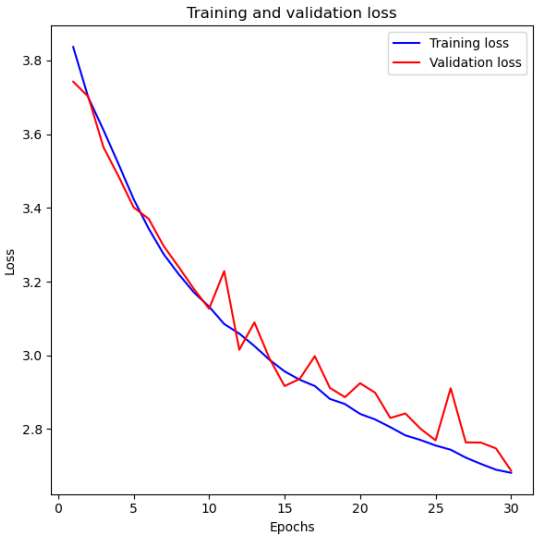


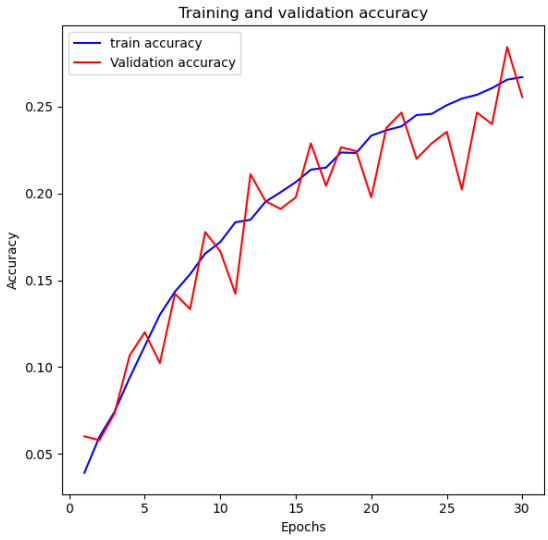
Channel_processing_DIY model in R



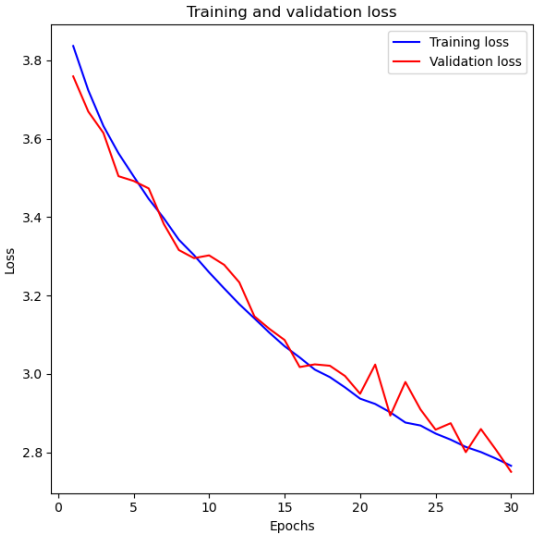


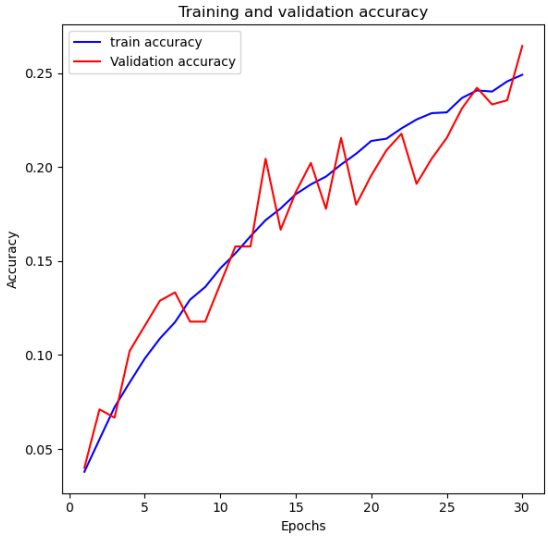
Channel_processing_DIY model in G



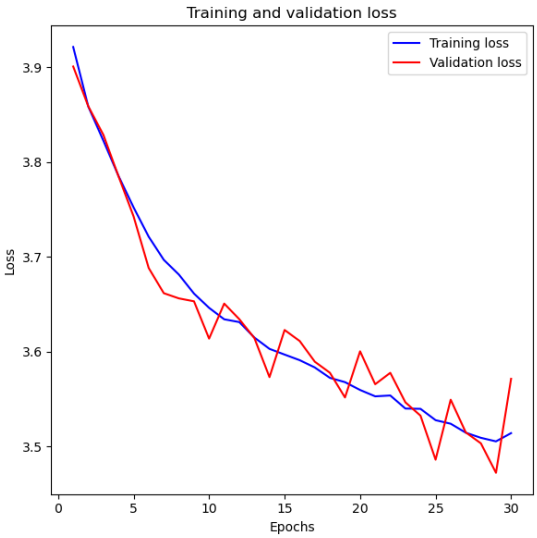


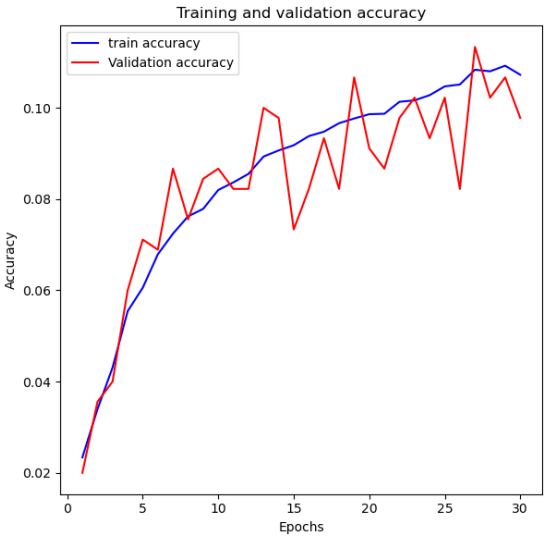
Channel_processing_DIY model in B



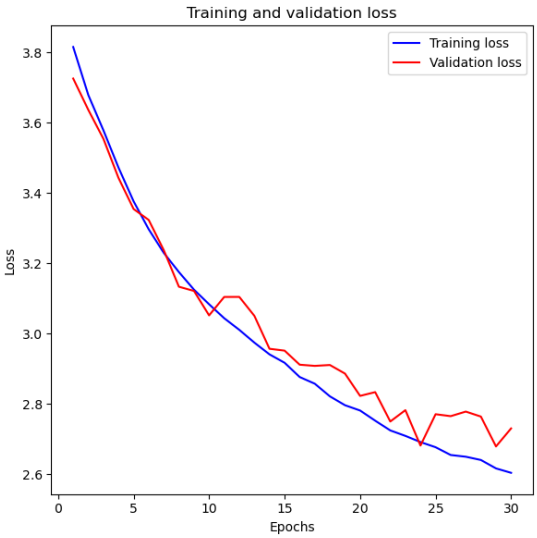


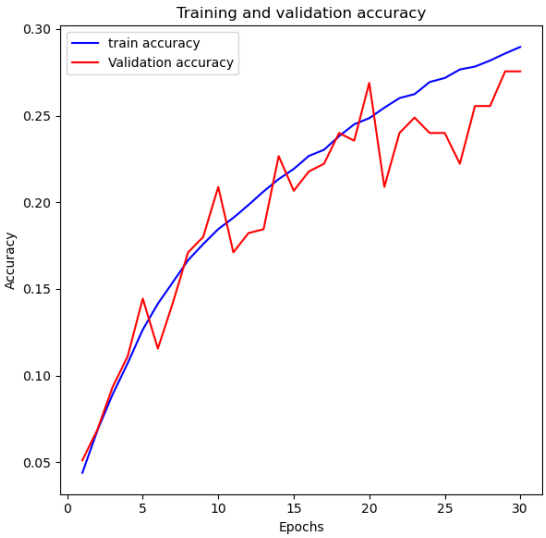
Channel_processing_sigmoid model in RGB



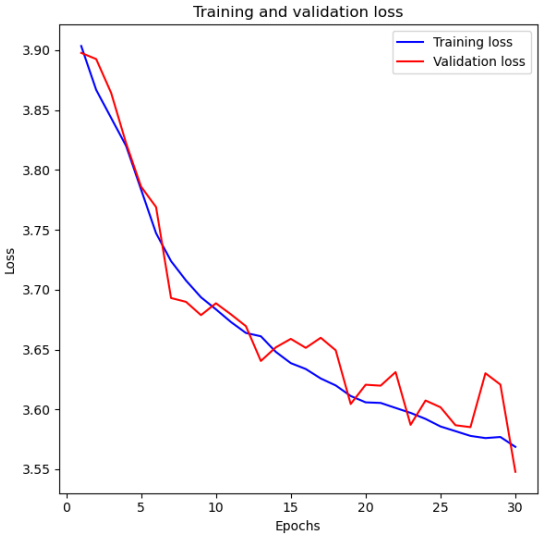


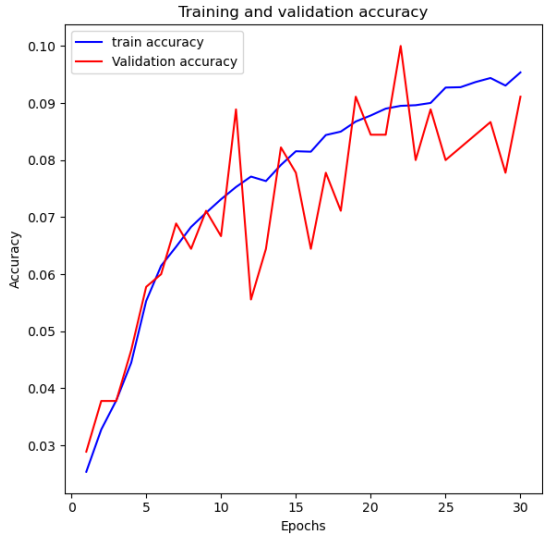
Channel_processing_sigmoid model in RG



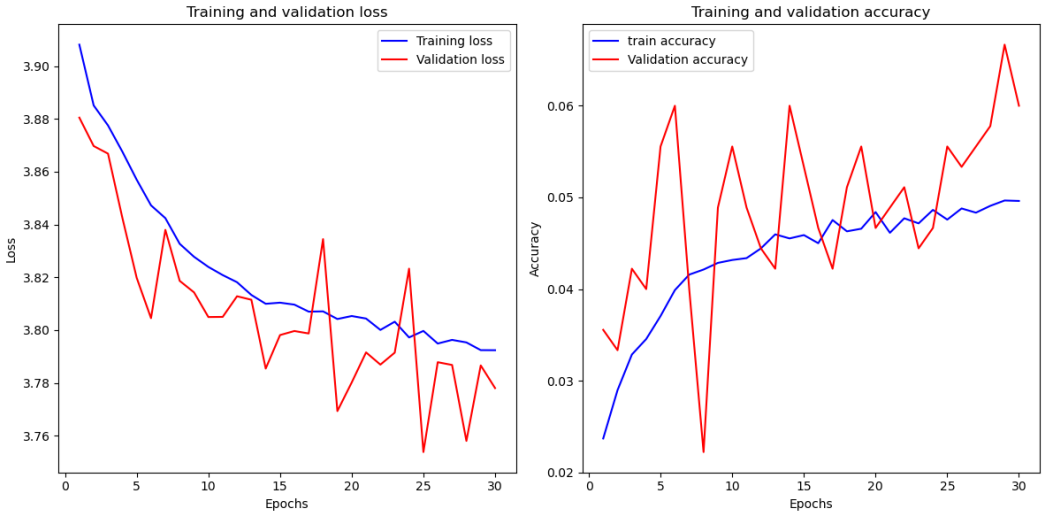


Channel_processing_sigmoid model in GB

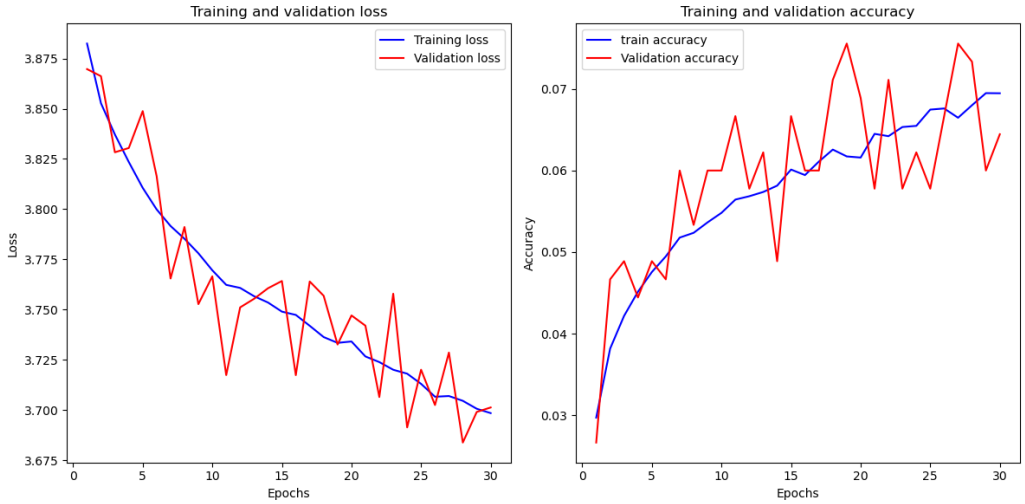




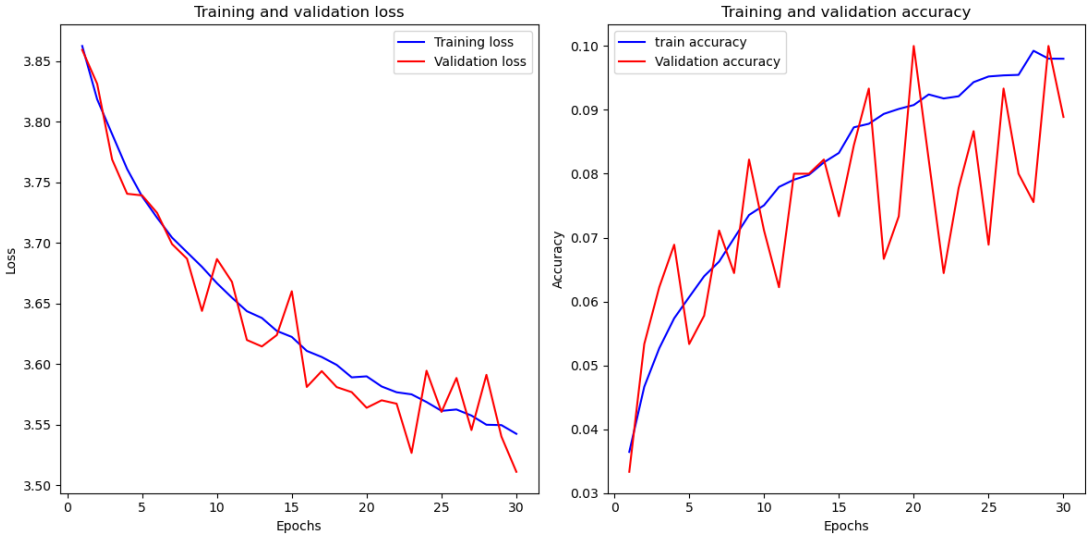
Channel_processing_sigmoid model in R



Channel_processing_sigmoid model in G



Channel_processing_sigmoid model in B



觀察與結論:

(1) 在 target channel 為 RGB 時，baseline model 有最好的表現，但在面對 Variable Input Channels 時，表現就大幅下降，原因可能是因為丟給模型的訓練資料都是 RGB，因此在面對非 RGB 的驗證與測試資料時，表現就大幅下降

(2) Channel_processing_DIY 的方法雖然簡單暴力，但由於這個方式是將訓練集、驗證集、測試集的資料都強制補成(或砍成)符合 target_channel 的樣子，因此不論是在面對何種 target channel 時，都有很好的表現，但此法在實務上較為不可行，因為通常無法預設 input data 長怎樣，且這樣暴力的作法可能會破壞資料本身的特性

(3) Channel_processing_sigmoid 的方法雖然看似採用較為自適應、平滑(比起上面的 DIY 方法而言)的補值或是砍值的方式，但顯然 sigmoid 的算法所產生的資料沒辦法讓模型有很好的表現，還不如像 DIY 的方式這樣把非 target channel 的部分直接補 0。

(4) Pool_Channel_Attention 的方式由於是透過平均池化與最大池化的 channel attention 方式來將 input channel 都經過池化層而變成 size 為 1，雖然可能會因此損失不少資料，但此法顯然比起上面兩個基於圖片前處理的方式還更加合理且具有泛化性，我們可以發現，雖然在 target channel 為 RGB 時，這個 method 的表現比不過 baseline，但在面對 Variable Input Channels 時，這個 method 的表現相對平穩很多，不會像 baseline model 這樣，只在 RGB 表現好。

4. Task2: Designing a Four-Layer Network for Image Classification

題目描述:

設計一個類神經網路模塊，該類神經網路模塊僅能包含 2~4 層 CNN，這個類神經網路模塊必須在相同狀況下達到 Resnet34 的 90%的表現。

關於如何認定 CNN 的層數:

一個 layer 必須等上一個 layer 完全計算完才能拿到數據,這樣算是兩層。但如果一層中做很多事情,例如包含 convolution、attention 機制,並將 attention 結果乘回到 convolution 特徵上,最後輸出,這也可以算是一層(或一個 block)。

思路:

在經過了實驗與觀察後，我發現在 epoch 較小的情況下，我們自己設計的模型比較有機會達成題目要求，且我分別透過了數據增強方法、並行架構方式與 spatial attention 方式進行嘗試，經由 task1 的啟發，我認為如果使用池化+注意力機制，應該可以提升表現，最後發現透過數據增強+pooling+attention 的方式能夠達到題目的要求

HyperParameter:

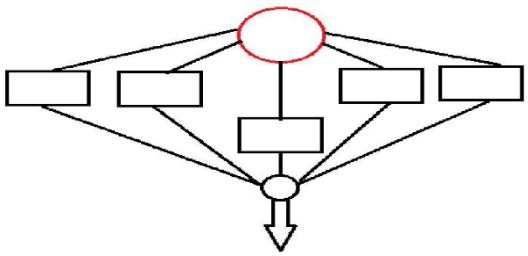
Batch_size=128,Num_workers=16,Num_epochs=10,Lr=0.0005

Baseline model:Resnet34

在僅僅 apply 了圖片 resize 與標準化的 transform 步驟後，不使用官方的 pre-train weight，而是以一個空的 resnet34 模型訓練了 10 個 epoch

Method 1: 並行架構方式

由於題目的說法是「一個 layer 必須等上一個 layer 完全計算完才能拿到數據,這樣算是兩層」，因此我認為，如果一個 input 同時並行傳給多個模塊進行處理，最後再接上一層 MLP 來彙整並得到輸出，應該也是符合題目要求的，示意圖如下:



因此我首先嘗試了將以下模塊並排放置:

1. 三層 CNN 的 block
2. self-attention block
3. RRDB block
4. GCN block

這樣理論上若要算 CNN 層數的話，應該就是上面這四個模塊中具有最多 CNN layer 的那個層數，也就是 3 層。

但最後發現這個方法雖然看似 ensemble 的概念，但是並沒有起到表現增幅疊加的作用，反而因為在僅有 10 個 epoch 的狀況下，模型正確率很低。

test_loss	test_accuracy
3.6418363857269287	0.08

Search this file

	epochs	train_loss	train_accuracy	validation_loss	validation_accuracy
1	1	3.80305492503531	0.04688511646269246	3.8674837642245823	0.05777777777777775
2	2	3.7246624247396225	0.061950256612712196	3.7958793046739365	0.042222222222222223
3	3	3.7052817264659295	0.0677773391235689	3.7521807098388673	0.05555555555555555
4	4	3.6895250660075156	0.07188314251875247	3.79272885216607	0.06222222222222222
5	5	3.6784062868420238	0.07294117647058823	3.8024028725094263	0.07555555555555556
6	6	3.671838374139761	0.07572048953809712	3.9171221256256104	0.042222222222222223
7	7	3.667490608363673	0.07663639952625345	3.804139493306478	0.05333333333333334
8	8	3.6690881611657997	0.07655744176865377	3.698337884479099	0.07111111111111111
9	9	3.6590653991209767	0.07963679431504145	3.6840325111813015	0.09111111111111111
10	10	3.649053894988737	0.08187919463087248	3.733420148425632	0.08444444444444445

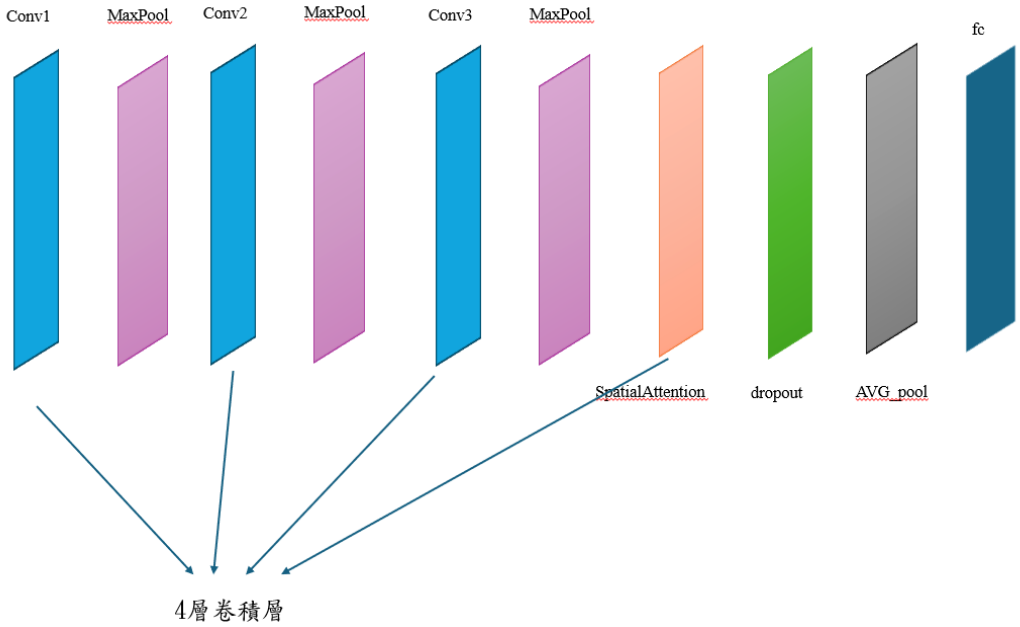
Method 2:

先透過數據強化處理:

Image augmentation:

```
v2.RandomResizedCrop(size=(224, 224), antialias=True),
v2.RandomHorizontalFlip(p=0.5),
v2.ToDtype(torch.float32, scale=True),
v2.Normalize(mean=mean, std=std),
v2.ToDtype(image_dtype, scale=True)
```

接著使用 3 個卷積模塊加上一個 spatial attention 模塊，中間放入最大池化、平均池化層，並加入 dropout 後，可以達到非常接近 resnet34 的表現



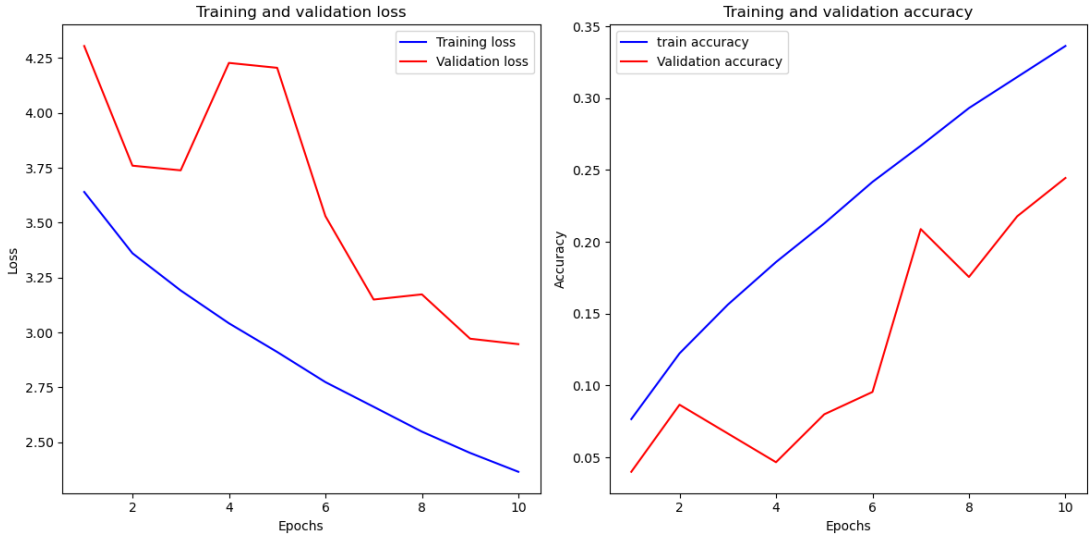
實驗結果:

Test accuracy

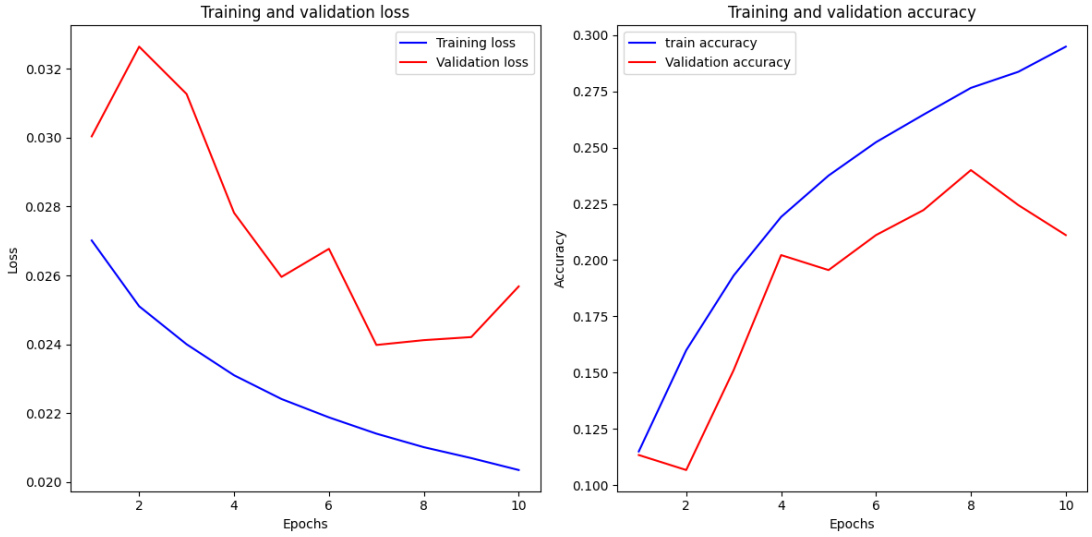
	Test acc
Resnet34	0.255
Task2_cool_model(題目要求的模型)	0.253

模型訓練過程 plot:

Resnet34



Task2_cool_model



觀察與結論:

雖然最終有達到題目的要求，我的模型的表現非常接近 resnet34 了，但是我認為有一點勝之不武的成分，因為我在訓練 task2_cool_model 時，多加了一兩種數據增強的方式，且這是在 epoch 僅有 10 的情形下，才能讓我的模型暫時媲美 resnet34，我自己試過，若在 epoch 為 30、50 甚至更高時，resnet34 的表現可以達到 acc 為 60%以上，我的模型在後期就無法趕上 resnet34 了。但我認為受了 task1 的啟發，我加入了平均池化、最大池化與空間注意力機制(因為這個 task 不需要考慮 variable input channel，所以從 channel attention 改為 spatial attention)，使模型在表現上有很大的改善(比起 method 1 的集成方法而言)

5. 心得

其實在大學期間，我並沒有太多的 DL 的背景，我對於 AI、DL 的認知可能頂多到常見的 ML 方法、資料科學的程度而已，這次作業讓我有了很好的練習機會，比如手建模型，自己寫 dataloader、訓練與驗證的 function，也讓我學到了一般正規的深度學習專案的框架該怎麼寫，例如 utils.py、model.py 裡面應該要寫甚麼等等，也算是對於碩士要研究的深度學習領域的實作方面有了一個全面的入門訓練。

6. reference

Introducing ChatGPT (openai.com)

[1912.03458] Dynamic Convolution: Attention over Convolution Kernels (arxiv.org)

kaijieshi7/Dynamic-convolution-Pytorch: Pytorch!!!Pytorch!!!Pytorch!!! Dynamic Convolution: Attention over Convolution Kernels (CVPR-2020) (github.com)

[1809.00219] ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks (arxiv.org)

xinntao/ESRGAN: ECCV18 Workshops - Enhanced SRGAN. Champion PIRM Challenge on Perceptual Super-Resolution. The training codes are in BasicSR. (github.com)

Learning PyTorch with Examples — PyTorch Tutorials 2.3.0+cu121 documentation

Spatial Attention 和 Channel Attention 的个人理解_from attention import channelattention, spatialatt-CSDN 博客