

YA-TING TSAI_final_project

訓練一個model會需要甚麼程式和檔案呢？經過兩個月的學習，我覺得應該要有以下的東西

1. 訓練資料(training data, ex: DIV2K、Flickr2K)
2. 評分標準(benchmark: PSNR of Set5)
3. 抓取資料的dataset、將資料形成一個個batch的dataloader
4. train model的檔案，包含每個epoch中做了甚麼事
5. model架構，最最重要的一個模型的核心
6. save checkpoint，也很重要的程式，沒有存到會哭出來的那一種

完整程式碼

<https://github.com/yating52997/SuperResolution>

線上版報告

[YA-TING TSAI_final_project](#)

因為大部分時間都在建構模型，所以將報告的順序設成：

- Final result
- Model
- 在此次final project中學到了甚麼
- 對AI training課的感想
- Dataset、train architecture

Final result

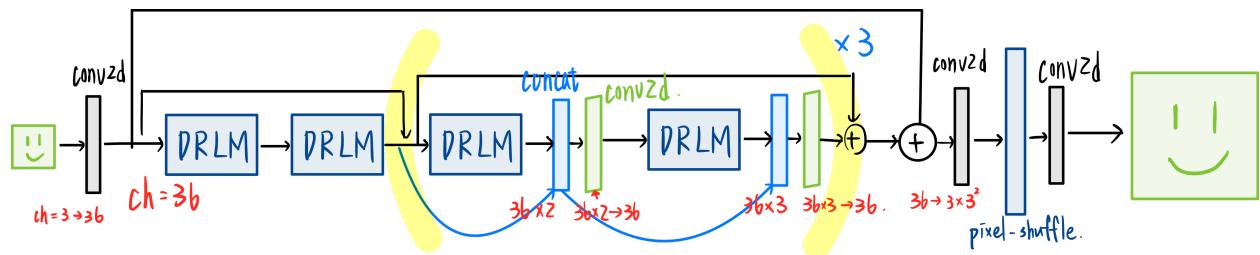
Average PSNR: 32.002dB、FLOPs: 10.88G、PARAMS: 0.1173M

```
● yating52997@MSI:~/ML/final_project$ python3 demo.py
Number of LR imgs (patches): 5 , Number of HR imgs (patches): 5
=====
Tesing: baby.png , PSNR: 34.042938232421875
Tesing: bird.png , PSNR: 35.0545654296875
Tesing: butterfly.png , PSNR: 28.74995994567871
Tesing: head.png , PSNR: 30.8249568939209
Tesing: woman.png , PSNR: 31.338890075683594

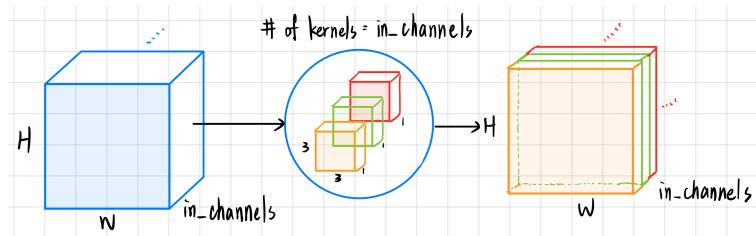
Average PSNR: 32.002 dB

=====
FLOPs : 10.88467488 G
PARAMS : 0.117399 M
=====
Your FLOPs is smaller than 11.1 G.
You will get this 10 points.
Congratulations !!!
=====
Overall Ranking Score: 127654.516
=====
Your score on Kaggle should be 234.200745
=====
```

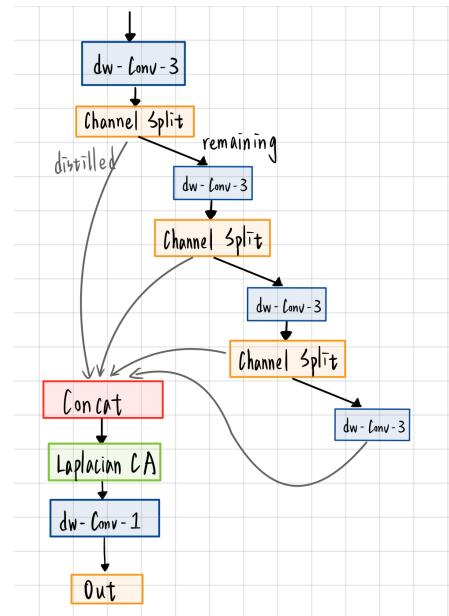
model architecture



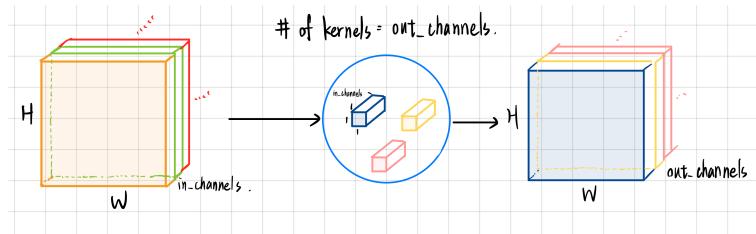
depthwise convolution



DRLM



pointwise convolution



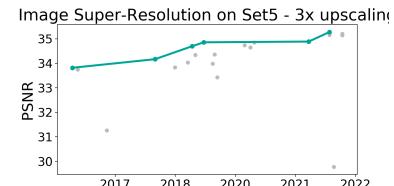
Model

打開排行榜來看一下有甚麼model可以使用

Papers with Code - Set5 - 3x upscaling Benchmark (Image Super-Resolution)

The current state-of-the-art on Set5 - 3x upscaling is HAT-L. See a full comparison of 21 papers with code.

<https://paperswithcode.com/sota/image-super-resolution-on-set5-3x-upscaling>



看到霸榜的HAT，而且是有關super resolution第一名都是他的那種，本來非常的心動，但一點進去Github，就發現

- Note that the default batch size per gpu is 4, which will cost about 20G memory for each GPU.

挖勒，我連10G都沒有根本train不了，而且看起來不只一個GPU

後來發現普遍的transformer好像都非常的胖

Model	Training time (# GPU)
EDSR-baseline-LTE	21h (1 GPU)
RDN-LTE	82h (2 GPU)
SwinIR-LTE	75h (4 GPU)

CNN-based

先轉戰CNN-based，讓自己可以train出一些東西出來

DRLN+

首先DRLN+，第一名的CNN-based，clone到工作站使用demo.py檔測試後

發現model很胖非常胖，為了不聞到顯卡燒起來的味道，還是先不train這一個model了，之後有機會再給他瘦身

```
=====
FLOPs : 4499.812678144 G
PARAMS : 34.614795 M
```

IMDN(Information Multi-distillation Network)

接著看到IMDN，一個在文章開頭就寫出lightweight輕量級的model，肯定不簡單

11 IMDN

34.36

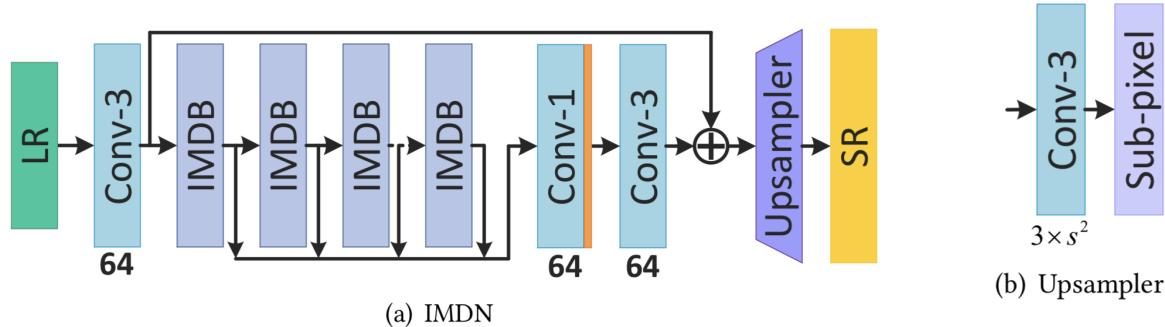
Lightweight Image Super-Resolution with Information
Multi-distillation Network



2019

<https://arxiv.org/pdf/1909.11856v1.pdf>

IMDN architecture



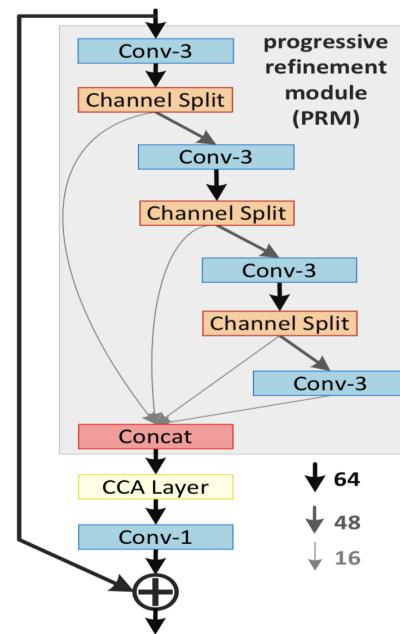
- (a) IMDN model包含前方 3×3 convolution with 64 output channel 作為low resolution照片的特徵提取
然後接著是多個information multi-distillation blocks(信息多重蒸餾塊)堆疊，且每一個IMDB會將輸出結果保存下來，最後Concat在一起
並匯入到後方 1×1 的convolution整合，論文中提到這有助於以較少的參數量收集到較完整的訊息
(b)最後的upsampler只包含一個 3×3 的convolution作為可學習層和不需要參數的pixelshuffle完成super resolution

IMDBlock

information multi-distillation block信息多重蒸餾模塊是由 progressive refinement module(PRM)、 contrast-aware channel attention(CCA)和最後的 1×1 convolution進行減少通道數

其中PRM是將資料先做 3×3 convolution後，利用torch.split 將channel分成蒸餾出來的與剩餘的，剩餘的會再經歷 3×3 convolution，將輸出channel數回復到原先最初輸入channel數，在進行split，所以每一次distillate和remaining channel數都會維持一樣，最後將所有的distillation 資料，沿著channel dimension concat起來

每一次的convolution都是由remaining通道數增加到input通道數，再藉由torch.split蒸餾出distilled channel and remaining channels，這樣可以有效做到使用較少的參數和算量，疊比較多層



CCA(Contrast-aware channel attention)

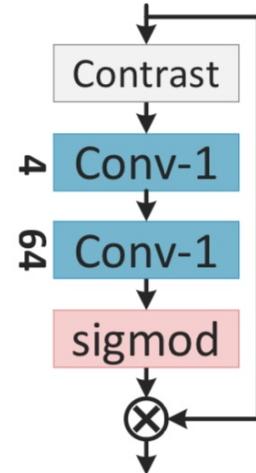
CCA和一般channel attention很不一樣的是他不使用global average pooling，改成使用標準差和平均值來評估特徵圖的對比度

論文中說到global average pooling有助於提升high level vision(e.g., classification or detection)捕捉特徵，提升PSNR，但會降低材質、邊緣的特性(SSIM)，

而CCA對於low level vision(e.g., image super-resolution)非常適用，若輸入是 $X = [x_1, \dots, x_c, \dots, x_C]$ 具有C個 $H \times W$ 的feature maps，

$$\begin{aligned} z_c &= H_{GC}(x_c) \\ &= \sqrt{\frac{1}{HW} \sum_{(i,j) \in x_c} \left(x_c^{i,j} - \frac{1}{HW} \sum_{(i,j) \in x_c} x_c^{i,j} \right)^2 +} \\ &\quad \frac{1}{HW} \sum_{(i,j) \in x_c} x_c^{i,j}, \end{aligned} \quad (5)$$

z_c 是c-th element of output



FLOPs、PARAMs

好像還是有點胖，在同一份檔案發現裡面有很多不同的model，來換一個試試

```
=====
FLOPs : 91.502942976 G
PARAMS : 0.703059 M
```

IMDN_RTC

此為作者為了參加AI in RTC競賽所使用的model，並獲得演算法組的第一名

作者也在Github上面寫到，這是IMDN的改良版，是ultra lightweight model

那就近多輕呢？

```
=====
FLOPs : 2.820538368 G
PARAMS : 0.021825 M
=====
```

真的是在所有的super resolution model中非常輕盈的
爆train一波試試

1st 嘗試

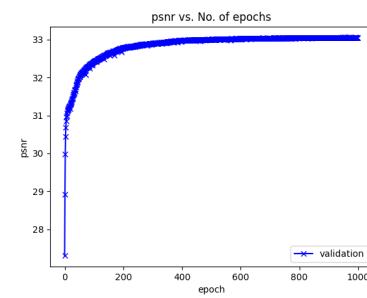
只使用DIV2K dataset，且一個epoch repeat data 20次， train到1000多個epoch，發現他真的上不去了，而且PSNR數值只有31.41

train一個epoch的時間大約是15sec

```
Number of LR imgs (patches): 5      , Number of HR imgs (patches): 5
=====EVALUATION=====
Tesing: baby.png      , PSNR: 33.870361328125
Tesing: bird.png       , PSNR: 34.139957427978516
Tesing: butterfly.png , PSNR: 27.487442016601562
Tesing: head.png       , PSNR: 30.736492156982422
Tesing: woman.png     , PSNR: 30.83127212524414

Average PSNR: 31.413 dB

=====
FLOPs : 2.820538368 G
PARAMS : 0.021825 M
=====
Your FLOPs is larger than 11.1 G.
You will not get this 10 points.
=====
```



數值為PSNR-Y(那時候真的以為自己達標了QQ)

2nd嘗試

覺得可能是資料集問題，DIV2K只有800張訓練圖片，可能需要更多變換性的資料，故在此加上Flickr2K，將DIV2K的資料量改成*10，而Flickr2K的不重複
model的部分維持不變(nf=12, num_modules=5)
train一個epoch的時間大約在21sec

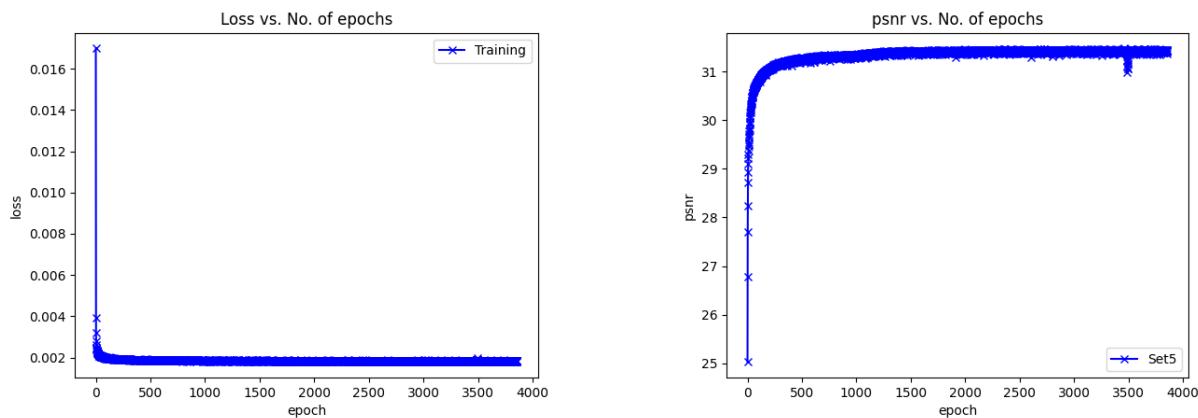
```

=====
=====EVALUATION=====
Tesing: baby.png      , PSNR: 33.896461486816406
Tesing: bird.png       , PSNR: 34.13275146484375
Tesing: butterfly.png , PSNR: 27.67892074584961
Tesing: head.png       , PSNR: 30.755369186401367
Tesing: woman.png      , PSNR: 30.90737533569336

Average PSNR: 31.474 dB

=====
FLOPs : 2.820538368 G
PARAMS : 0.021825 M
=====
Your FLOPs is larger than 11.1 G.
You will not get this 10 points.
=====
Overall Ranking Score: 1562734.000
=====
Your score on Kaggle should be 269.993408

```

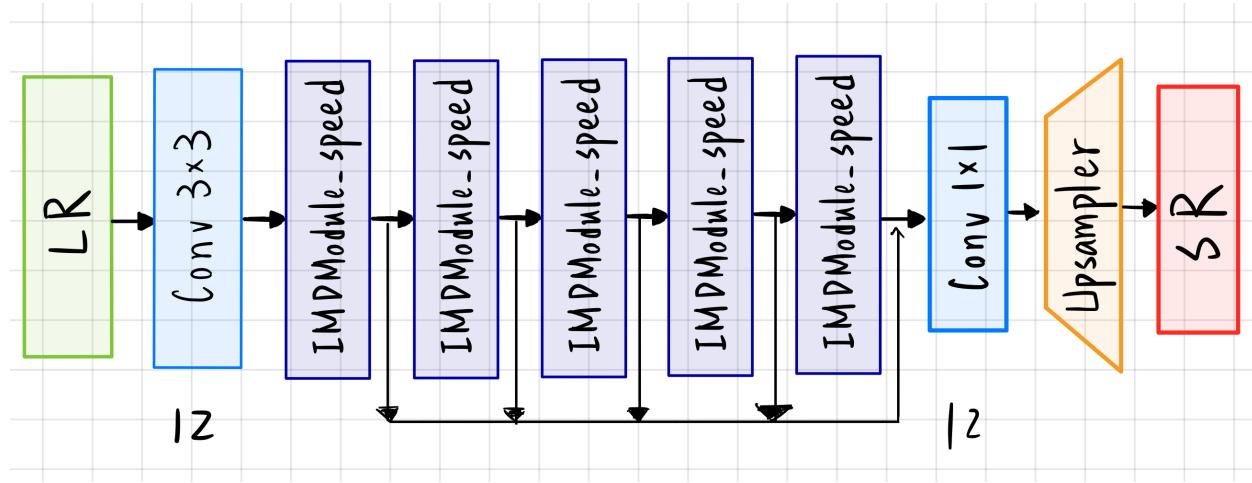


爆train一波，一直以為可能有機會可以上31.5，可以拿到bonus的10分，但真的太難了，後1000個epoch幾乎沒有動，最後的結果停在31.47，從這知道資料及增加有助於增加PSNR數值，但非常的微小，仍然無法上31.5QQ

IMDN_RTC architecture

```
=====
FLOPs : 2.820538368 G
PARAMS : 0.021825 M
=====
```

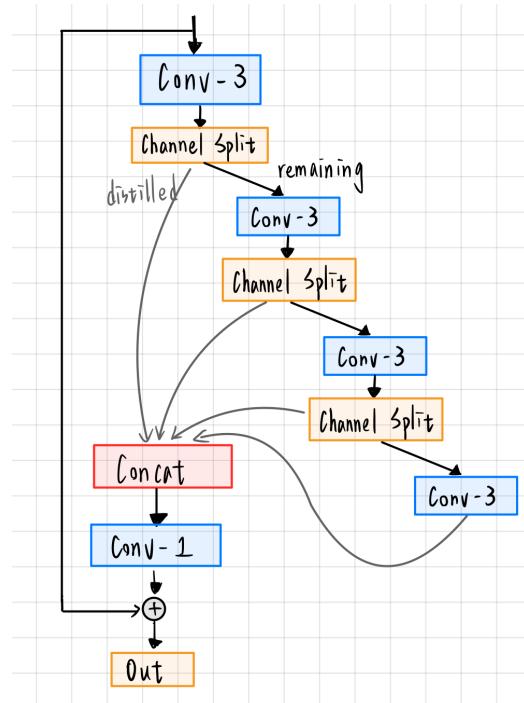
IMDN_RTC怎麼做到這麼輕的呢?來看一下他的架構



從架構、程式碼中可以看出，IMDN_RTC比IMDN還要少了一個IMDMoudel Block，且channel數從64下降到12，並在upsampler前只有一個 1×1 的convolution

在speed up IMDModule Block中，一樣採用訊息蒸餾的方式，較不一樣的地方是，在進行concat後，**不具有 Contrast-aware channel attention**層，而是直接做 1×1 變換channel數量

在進行完所有IMDModule後，speedup版本不具有將所有IMDModule concat起來的步驟，只有拉residual link進行加法而已，所以在後方的convolution層算量也會較輕



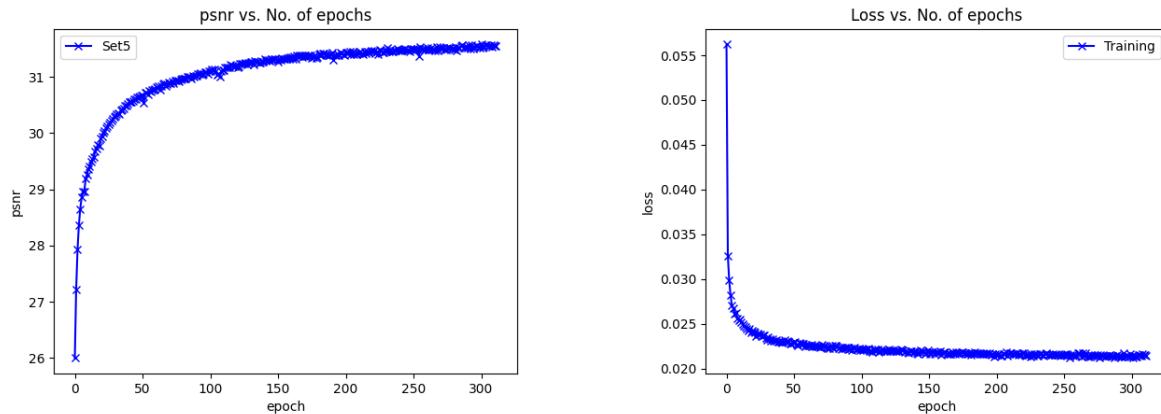
第一二次嘗試心得

沒有改model爆train一波的結果行不太通，看完兩個model的架構後，我覺得IMDN_RTC訓練結果都卡在31.4的原因可能是

1. 不具有Channel Attention層，雖然speedup module的channel數較少，但channel attention具有可以加大感知範圍的能力、與找到較重要通道數的能力
2. 模型本身太淺了，12個channel和5個module雖然可以有效減低算量，但深度和複雜度都不太夠

3rd 嘗試改良

1. 先將channel數增加到16、module數增加到10



在250個epoch左右達標，終於超過31.5且
模型大小有壓在11.1G以下

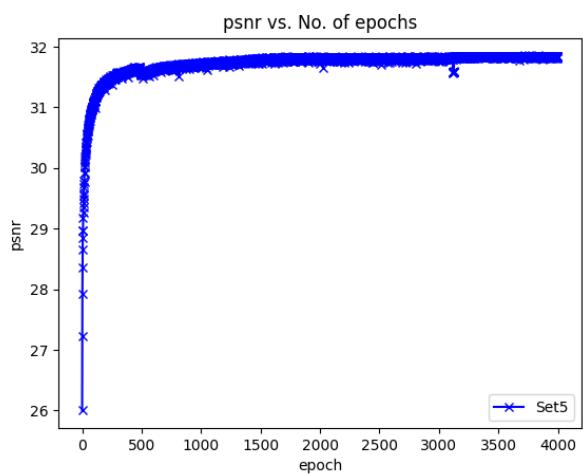
本來想說爆train一波有沒有機會超過32

但最終仍然卡在31.8上不去

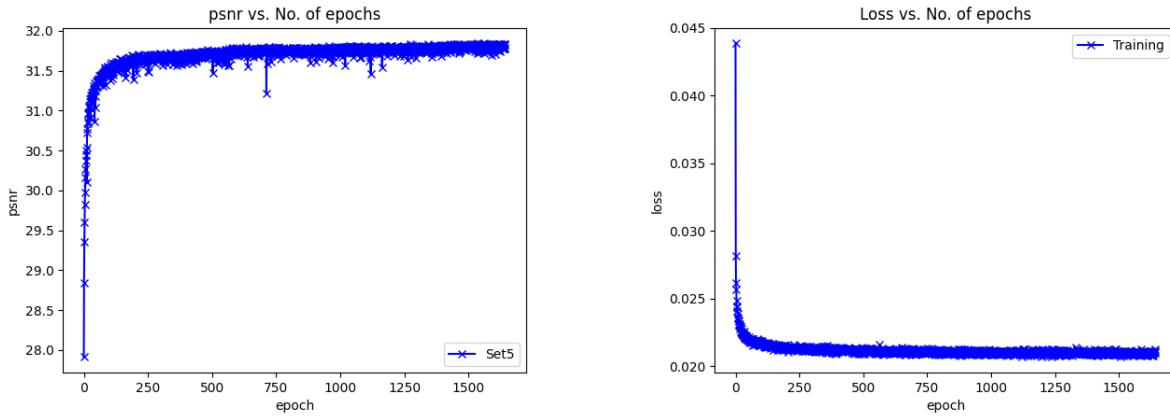
```
Number of LR imgs (patches): 5 , Number of HR imgs (patches): 5
=====EVALUATION=====
Tesing: baby.png , PSNR: 33.9485244750997656
Tesing: bird.png , PSNR: 34.42723846435547
Tesing: butterfly.png , PSNR: 27.741779327392578
Tesing: head.png , PSNR: 38.78426742553711
Tesing: woman.png , PSNR: 38.97205352783203

Average PSNR: 31.575 dB

=====
FLOPs : 9.0513088032 G
PARAMS : 0.069795 M
=====
Your FLOPs is smaller than 11.1 G.
You will get this 10 points.
Congratulations !!!
=====
Overall Ranking Score: 168393.219
=====
Your score on Kaggle should be 265.294556
=====
```



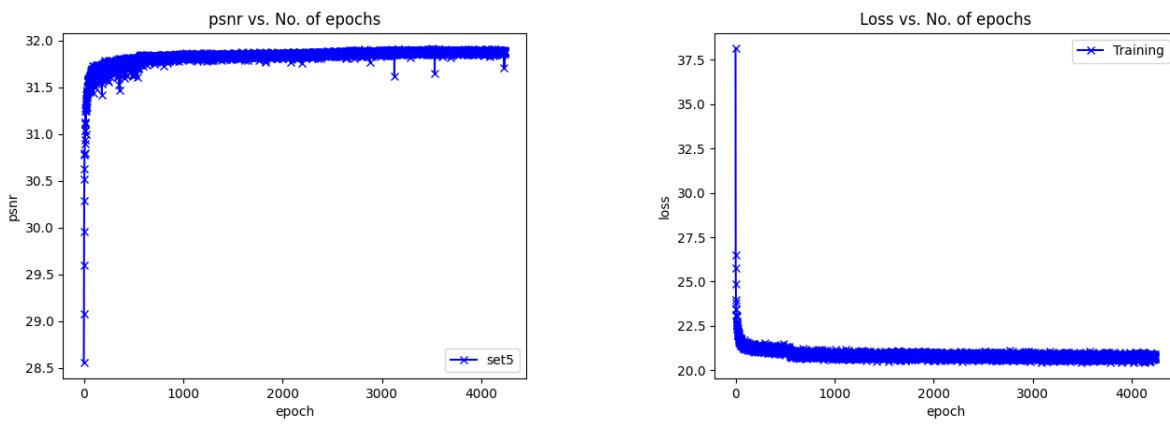
2. 將channel數增加到12、module數增加到16



在大概100個epoch左右達標，超過31.5且模型大小有壓在11.1G以下
較上一個模型深，但channel數較少，可是具有較好的表現，最後的結果停在31.84

4th使用IMDN嘗試改良

將原先64 channels 改成20 channels, module維持在6， 使用Contrast-aware channel attention
算量高好壓在9.43G(還好沒有爆)



```

(coco) [0111] tsai@eng03:~/BN] python -m evalpy
Number of LR imgs (patches): 5 , Number of HR imgs (patches): 5
=====EVALUATION=====
Tesing: baby.png , PSNR: 34.02644729614258
Tesing: bird.png , PSNR: 34.9442253112793
Tesing: butterfly.png , PSNR: 28.540180206298828
Tesing: head.png , PSNR: 30.78606414794922
Tesing: woman.png , PSNR: 31.2586612701416

Average PSNR: 31.911 dB

=====
FLOPs : 9.43325256 G
PARAMS : 0.072813 M
=====
Your FLOPs is smaller than 11.1 G.
You will get this 10 points.
Congratulations !!!
=====
Overall Ranking Score: 216800.578
=====
Your score on Kaggle should be 240.486664
=====
```

這個model之所以會被train到4000個epoch(兩天)是因為，一直想說31.91可能等等就上去了，很可惜後來沒有真的上去，停留在31.91。

在第三次和第四次的改良中學到

1. channel attention有其必要性，像在第四次train時，雖然深度沒有第一次深，卻有較好的表現，而且也是一次碰到31.9
2. 加深加廣有助於增加複雜度，進而增加判斷顏色的成功率與psnr的數值
3. 將所有的IMDBlock concat在一起好像比較有助於training，雖然算量會因此增加很多

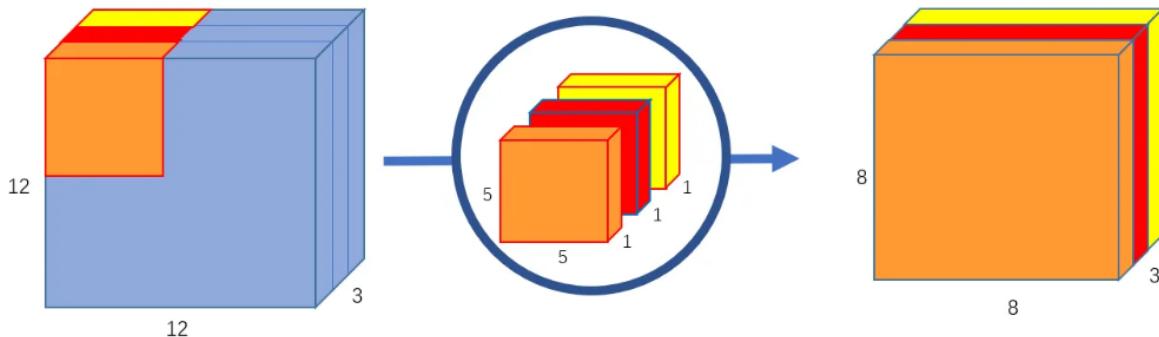
接下來需要將整體再加深加廣，但其實第三次和第四次堆的model都已經到達9.9G FLOPs的數量了，要是增加成channels和增加block數的話，勢必會突破11.1的門檻，沒有辦法拿到bonus項

這時我想到之前期中專題有使用到的depthwise seperable convolution

Depthwise seperable convolution in IMDN

depthwise seperable convolution其實可以分成 depthwise convolution和pointwise convolution，目的是在減少算量的同時，可以維持channel數量不變，且convolution數會從一層變成兩層，不僅算量減少還增加深度

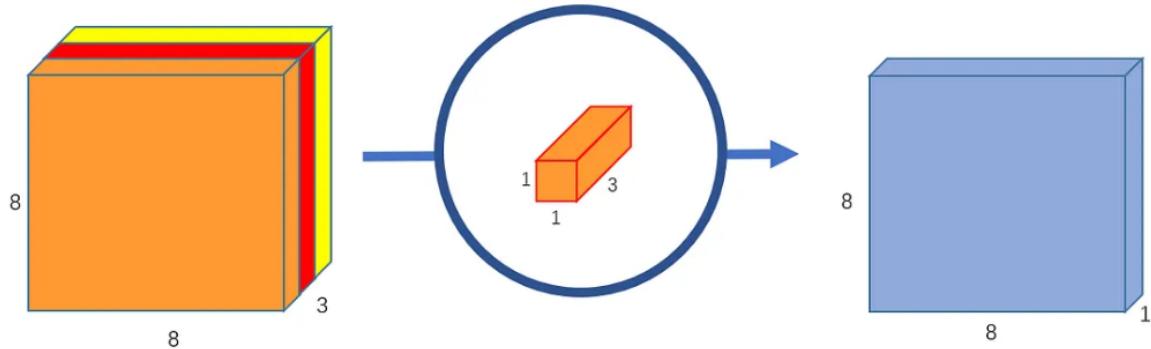
depthwise convoltion



```
nn.Conv2d(in_channels, in_channels, kernel_size=kernel_size, stride=stride,  
padding=kernel_size // 2, groups=in_channels, bias=bias)
```

可以從圖中得知depthwise convolution是分別對每一個channel做convoution，從code可以知道我們將group設定為input channels的數量，所以在算量上會從原先 $H \times W \times C$ 變成 $H \times W \times 1$ ，參數量因kernel的channel數量下降而減少

pointwise convolution

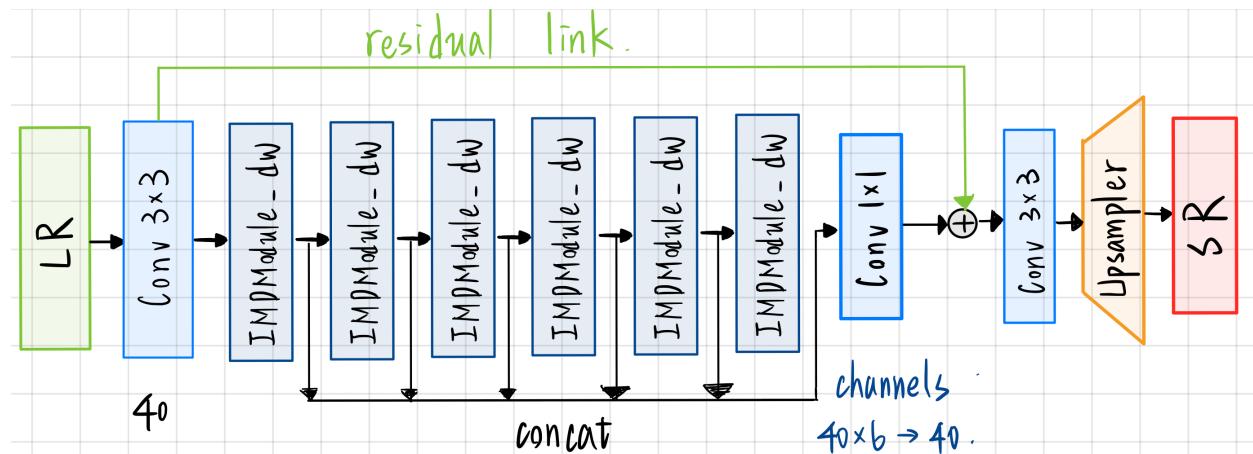


```
nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, padding=0, bias=bias)
```

可以從圖中得知，pointwise convolution是利用 1×1 kernel的數量達到想要的output channel數，此處的 1×1 kernel有助於減少參數量，在進行運算時也會從 $kernel \times kernel \times H \times W \times C$ 變成 $1 \times 1 \times H \times W \times C$

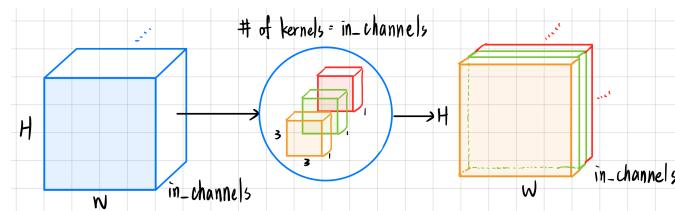
1st嘗試使用depthwise convolution

architecture

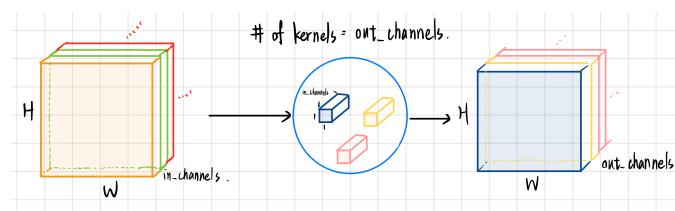


首先在架構的部分將channel的數量增加到40，然後將模型疊到壓在11.1G底下，最終的計算結果是疊6層的IMDModule_dw

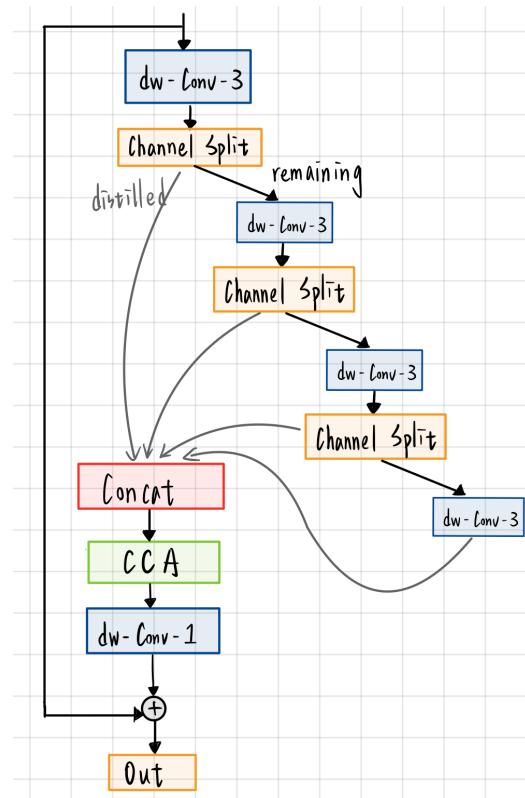
depthwise convolution



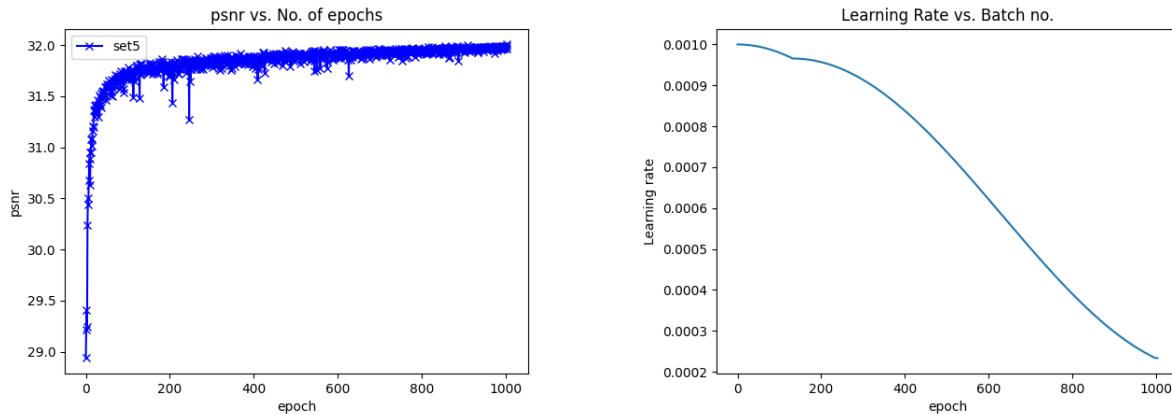
pointwise convolution



而IMDModule內，將原先的conv3全都替換成depthwise seperable convolution，並保留CCA，將輸入與輸出相加形成residual link



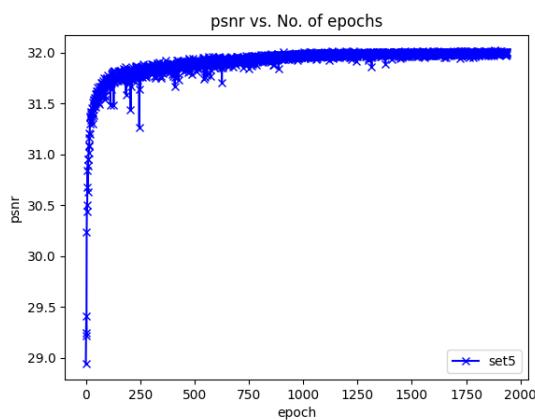
爆train一波的結果，在909個epoch時到達32!!!!



```

epoch = 908 lr =  0.0002924976520259775
100%|██████████| 1000/1000 [02:56<00:00,  5.67it/s]
==> Epoch[908](1000/1000): Loss_l1: 0.02048
==> Valid. psnr: 31.9596
==> Best PSNR: 31.9914
epoch = 909 lr =  0.0002917152014671382
100%|██████████| 1000/1000 [02:55<00:00,  5.68it/s]
==> Epoch[909](1000/1000): Loss_l1: 0.02058
==> Valid. psnr: 32.0027
==> Checkpoint saved to checkpointdnew_x3/IMDN.pth
==> Best PSNR: 32.0027
epoch = 910 lr =  0.0002909356237402855
100%|██████████| 1000/1000 [02:55<00:00,  5.70it/s]
==> Epoch[910](1000/1000): Loss_l1: 0.02047
==> Valid. psnr: 31.9729
==> Best PSNR: 32.0027

```



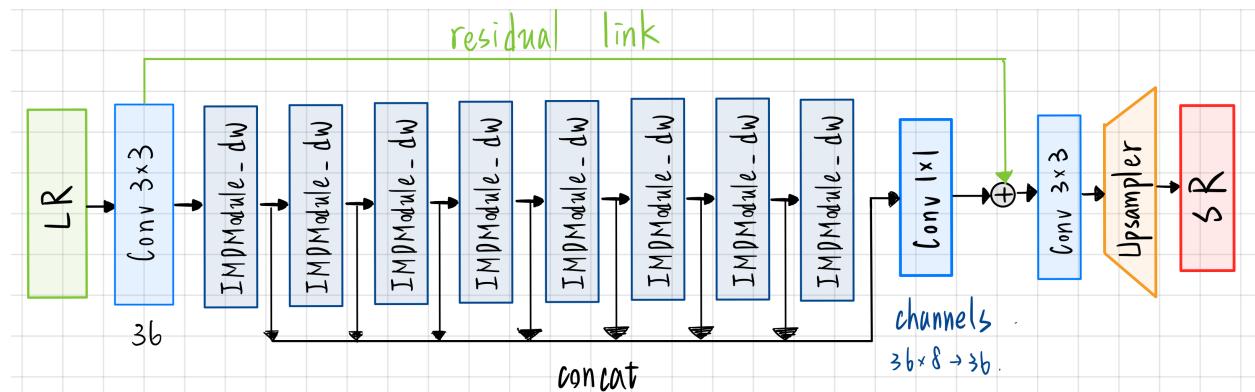
想說試試看模型的極限，於是train到2000，最終到達32.021dB

```

● (torch) [U114yttsai@eng05 IMDN_copy]$ python demo.py
Number of LR imgs (patches): 5 , Number of HR imgs ( patches): 5
=====
=====EVALUATION=====
=====
Tesing: baby.png      , PSNR: 34.034934997558594
Tesing: bird.png       , PSNR: 34.968841552734375
Tesing: butterfly.png , PSNR: 28.924819946289062
Tesing: head.png        , PSNR: 30.811368942260742
Tesing: woman.png       , PSNR: 31.366313934326172
Average PSNR: 32.021 dB
=====
=====
FLOPs : 10.2157584 G
PARAMS : 0.081345 M
=====
Your FLOPs is smaller than 11.1 G.
You will get this 10 points.
Congratulations !!!
=====
Overall Ranking Score: 200061.406
=====
Your score on Kaggle should be 231.098602
=====
```

發現depthwise seperable convolution挺有料的，於是又嘗試形成其他差不多結構的模型出來，算量都壓在11.1G底下

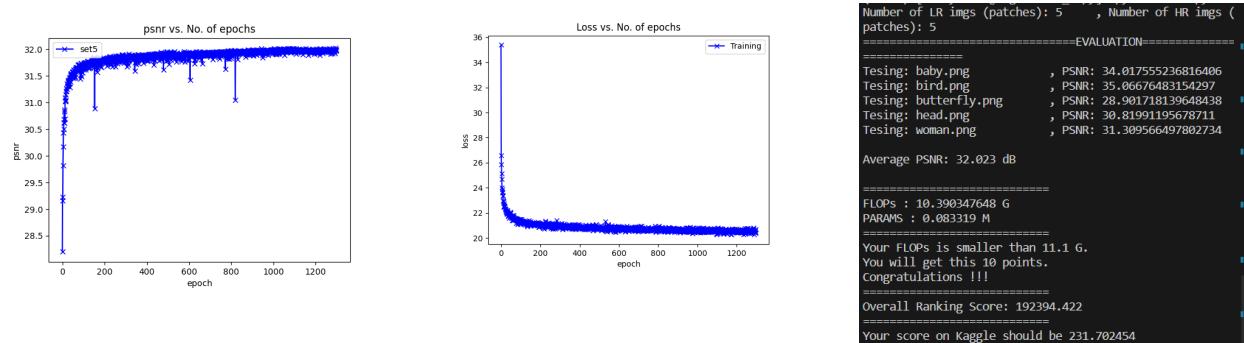
2nd嘗試使用depthwise convolution architecture



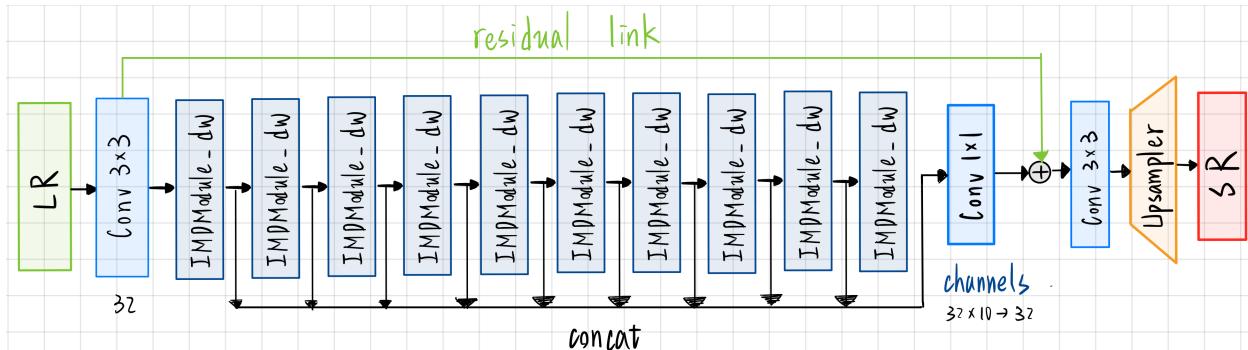
想說再疊兩層IMDModule_dw看看效果，但這樣就會超出11.1G，

最後將channel數改成36、IMDModule_dw疊八層

train一波也32了!!!!



3rd嘗試使用depthwise convolution architecture



再疊兩層呢?主打一個嘗試不同深度寬度的
model、有GPU就train爆的精神

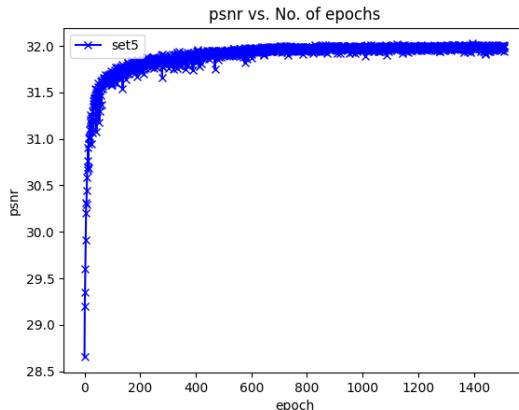
將channel數改成32、IMDModule_dw疊10層
train一波也32了!!!!

大概700個epoch就達標了，只是想說在train遺下
看看結果，一不小心train到1500個epoch，最終
結果大概32.027

```
(torch) [u114ytsai@eng05 IMDN_copy]$ python demo.py
Number of LR imgs (patches): 5      , Number of HR imgs (patches): 5
=====
=====EVALUATION=====
=====
Tesiing: baby.png      , PSNR: 34.050697326660156
Tesiing: bird.png      , PSNR: 35.100624084472656
Tesiing: butterfly.png , PSNR: 28.8053035736084
Tesiing: head.png      , PSNR: 30.799945831298828
Tesiing: woman.png     , PSNR: 31.379642486572266

Average PSNR: 32.027 dB

=====
FLOPs : 9.984547072 G
PARAMS : 0.079999 M
=====
Your FLOPs is smaller than 11.1 G.
You will get this 10 points.
Congratulations !!!
=====
Overall Ranking Score: 209388.531
=====
Your score on Kaggle should be 232.707153
```



在depthwise seperable convolution中學習到:

1. 好像模型必須有一定的算量(約等於10G)和參數量(0.08M左右)，才會達到PSNR = 32，先前有train過channel數沒有那麼多的depthwise(5G)，只止步於31.88

2. 加深深度比加大channel數還要有更好的performance，像第三次嘗試的channel = 10,
of block = 32，不僅算量、參數量較低，還可以更快達到標準

DRLN+

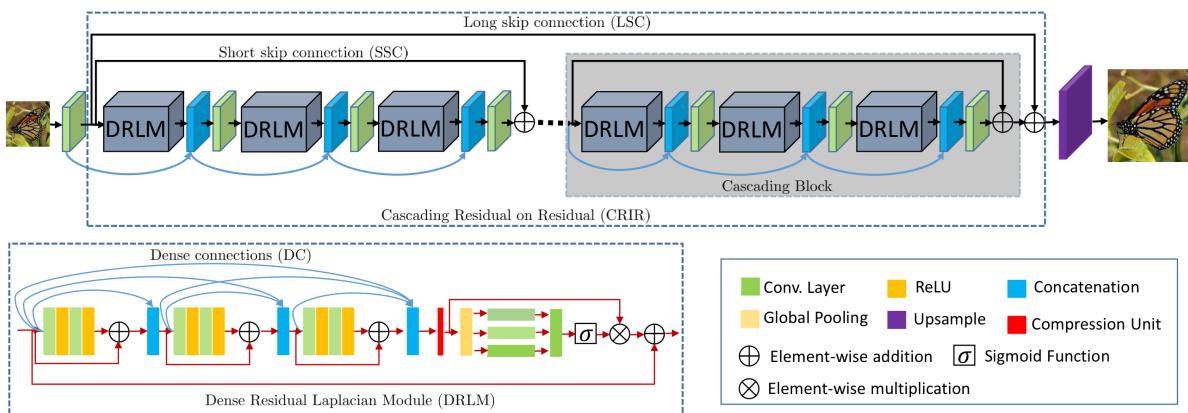
Densely Residual Laplacian Super-resolution

<https://github.com/saeed-anwar/DRLN>

真的很好奇他如何以CNN-based的姿態闖進前幾名的super resolution on set5 3x，於是開始看論文與看code

architecture

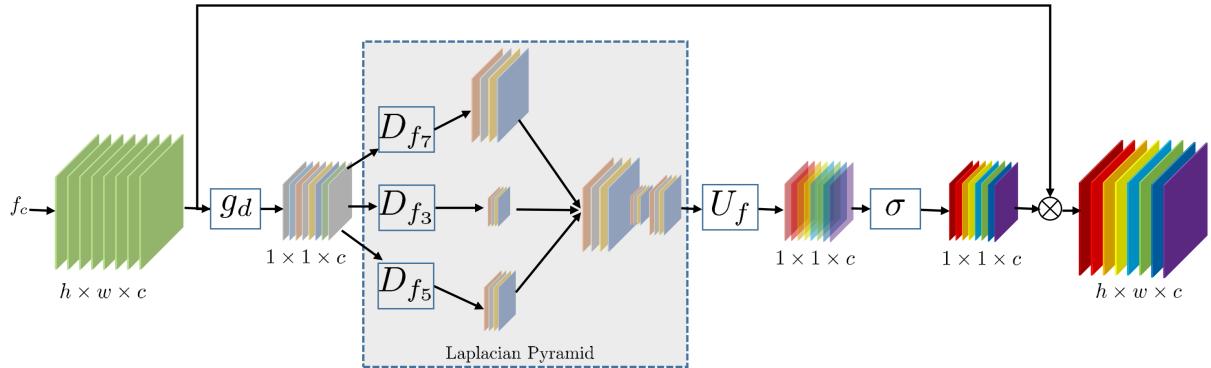
3



在論文中作者有寫到她是由很多CNN-based的model所改良出來的，像CARN、EDSR...，希望可以做到在不要疊到400層的情況下，能有較高的performance

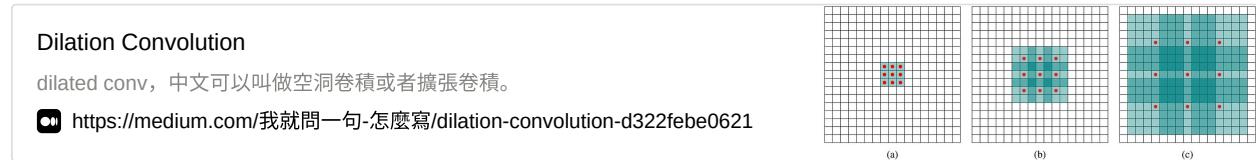
讀完扣後，會發現它是由很多個DRLM堆疊而成，輸入會經過DRLM們然後和上一次concat的結果concat在一起，形成channel數(ch*1 → ch*2 → ch*3 → ch*4)，concat出來的結果還會經過一個正常的convolution將channel數回歸到ch*1，每三個DRLM會進行一次residual link，大概堆20個DRLM

那DRLM是甚麼呢？它是由三組residual block所組成的，每一次block結束都會和上次block的結果進行concat，最後再藉由後方一般的convolution將channel數回歸成input channel數，後面再加上他們獨門的Laplacian channel attention



Laplacian channel attention主要的實作方法是將輸入做AdapativeAvgpool(1)後，分成三組以不同的dilation達到laplacian的目的，整體來說算及清，因為convolution都作用在 $C \times 1 \times 1$ 上

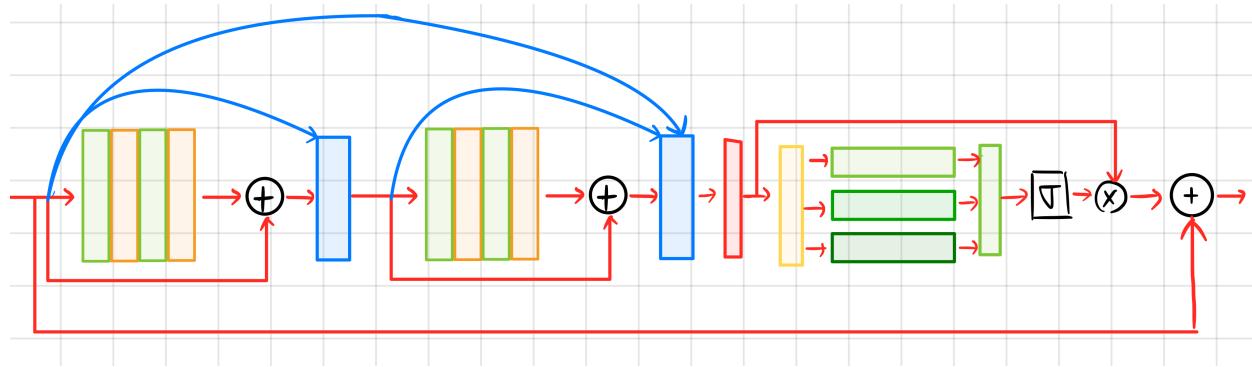
在一般的使用dilation具有可以增加感受視野的目的



```
=====
FLOPs : 4499.812678144 G
PARAMS : 34.614795 M
```

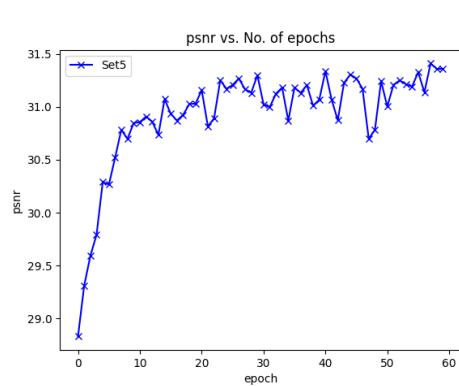
雖然作者強調他沒有隨便爆疊一波到400層，但這個算量仍然很驚人，於是開始運用之前在IMDN學習的技巧一步步縮減模型

1st嘗試



DRLM內本來有3個residual block，由於每次residual block的input會和上次residual block的結果concat，所以channel會從 $1 \rightarrow 2 \rightarrow 4 \rightarrow 8$ 以倍速成長，residual block又有輸入和輸出的大小必須完全相同的規定(因為需要相加)，所以第一次嘗試直接把第三個residual block拔掉

並將總channel數從64下降到12，並只保留四層DRLM，算量仍然非常驚人



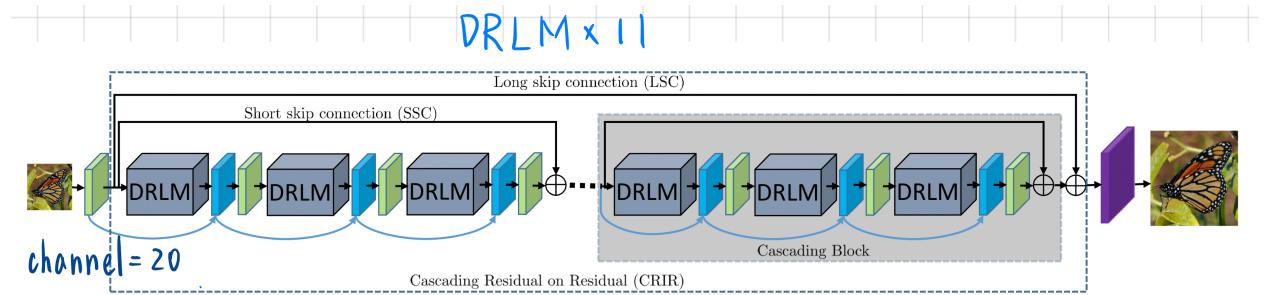
train了60個epoch，覺得好像還行，就是learning rate不小心調太大($lr = 0.02$)，以至於PSNR數值非常跳

```
S-E:\git\...py\noname.py
Number of LR imgs (patches): 5      , Number of HR imgs (patches): 5
=====
=====EVALUATION=====
=====
Tesing: baby.png          , PSNR: 33.72398376464844
Tesing: bird.png           , PSNR: 34.00670623779297
Tesing: butterfly.png     , PSNR: 27.75505828857422
Tesing: head.png           , PSNR: 30.651264190673828
Tesing: woman.png          , PSNR: 30.908193588256836

Average PSNR: 31.409 dB

=====
FLOPs : 10.767045792 G
PARAMS : 0.082599 M
=====
Your FLOPs is larger than 11.1 G.
You will not get this 10 points.
=====
Overall Ranking Score: 101347.406
=====
Your score on Kaggle should be 271.194214
```

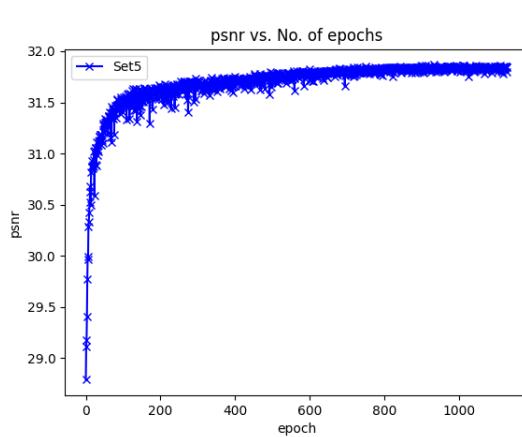
2nd嘗試



鬼轉depthwise seperable convolution壓算量與加深深度和channel數

將DRLN的concat basic block(將cannel數從兩、三倍轉回一倍的block)以depthwise seperable convolution撰寫，算量就下降非常多

最後設定channel數為20，深度有11個DRLM



爆train一波的結果停在31.86，覺得應該是channel數不夠，於是開始第三次改良

```

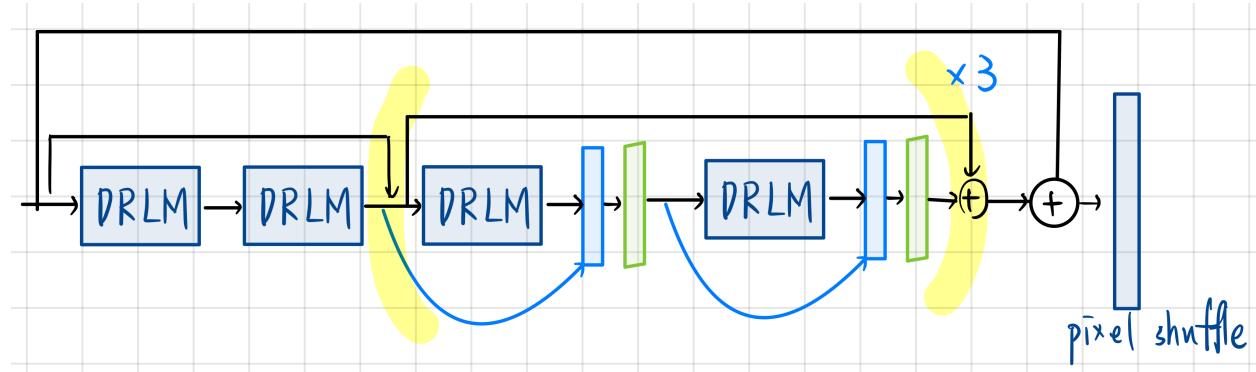
PS E:\drln> python demo.py
Number of LR imgs (patches): 5      , Number of HR imgs (patches): 5
=====
=====EVALUATION=====
=====
Tesing: baby.png      , PSNR: 34.016319274902344
Tesing: bird.png      , PSNR: 34.78202819824219
Tesing: butterfly.png , PSNR: 28.531753540039062
Tesing: head.png      , PSNR: 30.789440155029297
Tesing: woman.png     , PSNR: 31.21934700012207

Average PSNR: 31.868 dB

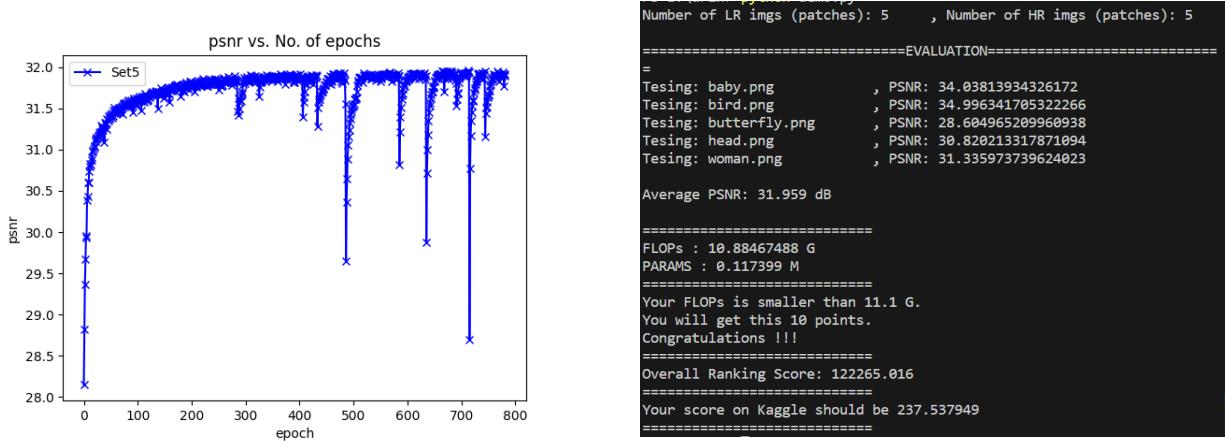
=====
FLOPs : 10.58146676 G
PARAMS : 0.09062 M
=====
Your FLOPs is smaller than 11.1 G.
You will get this 10 points.
Congratulations !!!
=====
Overall Ranking Score: 148709.922
=====
Your score on Kaggle should be 241.916397
  
```

3rd嘗試

覺得前兩次深度跟複雜度都不太夠，於是再次改良模型



1. 加大channel數到36: 讓模型具有一定複雜度
2. 將DRLM block改成IMDModule使用蒸餾的方式:
由於residual link的特性是輸入大小要和輸出大小一樣，所以在convolution上算量就會較大，改成IMDModule使用蒸餾的方式，convolution的input為remaining channel < original channel，最後將蒸餾結果concat在一起，通過沒什麼算量的Laplacian channel attention，得到輸出結果
3. 下降concat的數量: 調整DRLM block的連接，一開始兩個DRLM不concat，只拉residual link就好，接下來每兩個DRLM為一組(所以這樣channel得層數就從 $2 \rightarrow 3 \rightarrow 4$ 轉變成 $2 \rightarrow 3$)，這樣算量會下降很多，而且還可以具有8層的DRLM block
4. pixel shuffle作調整: 原先DRLN是將channel數增加到channel*9，再pixelshuffle(3)，後面接上convolution，但經過pixel shuffle會使圖形放大，所以後方的convolution從channel = 36->3時，算量會遽增，而前方的channel = 36 → 36*9，也非常地佔算量，所以更改成channel = 36 → 3*(scale**2)，直接少了大概4G FLOPs數
5. 將ReLU改成LeakyReLU，具有更多的複雜度

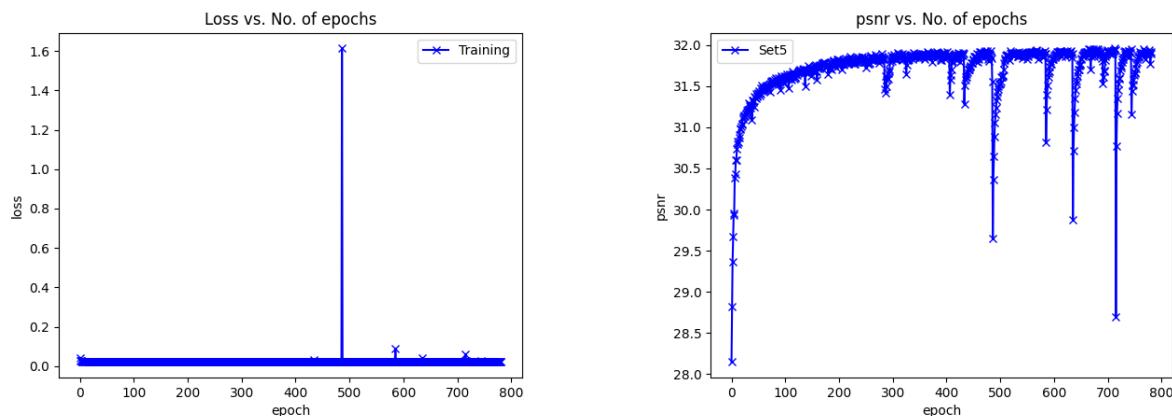
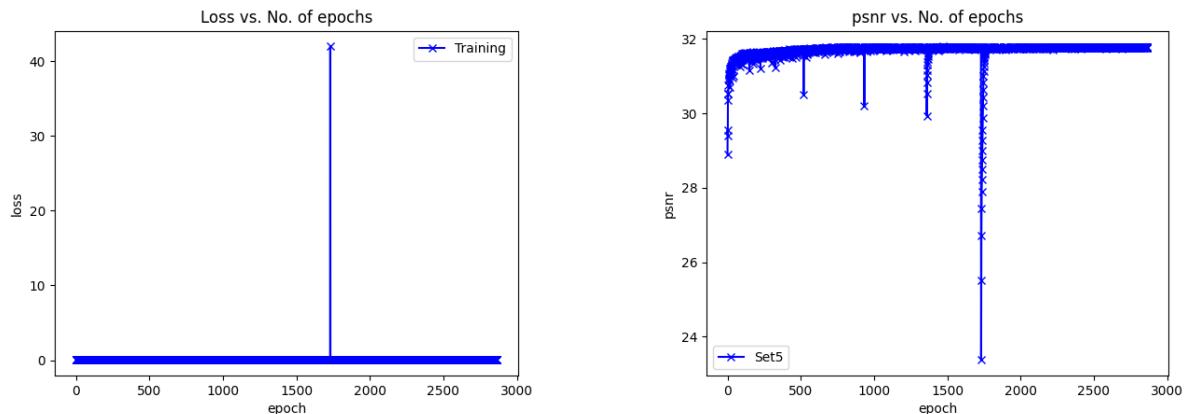


由於這個模型前200個是在工作站上train，但由於工作站倒數一周的時候較滿，所以後600 epochs就以自己的電腦train，沒想到放一個晚上就變成這樣了，但具有還不錯的結果(?如果沒有奇怪的掉下去的話應該有機會可以突破32，可惜工作站好滿嗚嗚嗚QQ

用自己的電腦train出32了，達標了，感動!!!!

```
Train Loss: 0.021638404928211473
100%|██████████| 6/6 [00:01<00:00,  3.84it/s]
Val PSNR: 28.34320640563965
Test PSNR: 31.990581512451172
Best PSNR: 31.998088836669922
Epoch: 1047 lr: 0.00025
100%|██████████| 585/585 [01:02<00:00,  9.32it/s]
Train Loss: 0.021659377643949967
100%|██████████| 6/6 [00:01<00:00,  3.72it/s]
Val PSNR: 27.847368240356445
Test PSNR: 32.00132369995117
Model saved!
Best PSNR: 32.00132369995117
```

Loss suddenly jumps up



後來上網查詢發現多人也都有差不多的問題，train loss突然飆的超高，並非over fitting，而是learning rate調太高了，模型在訓練的時候可能錯過minimum的數值

所以在最後繳交上去的model中之所以會突然train loss飆高的原因是預估錯他的收斂效果了，大概300個epoch時就已經31.88，但在前幾次訓練都是使用1000個epoch前，使用cosineannealinglr從0.001下降至0.0003，而300個epoch時通常= 0.0008左右還非常的高，後來手動調整，但又忘了存紀錄(那又是另外的狀況了XD)

我推測還有可能是訓練資料已經訓練到很低的點了，沒有甚麼進步空間，所以Loss就開始亂走動，將batch開大、加入Flickr2K即可解決這個問題。

在此次final project中學到了甚麼

在此次的final project中真的收穫很大，從一開始尋找適合的model，將model一個一個load進工作站當中，到讀論文讀程式碼，看懂內部的架構，真的花了很多時間，但也因此對於後續改變模型有很大的幫助、並讚嘆模型架構的發明者真的好厲害。

在實作的過程中，學習到要將dataset.py和train.py寫好，一個負責讀資料另一個負責搞定訓練，在期末專題的後兩個星期，幾乎都沒有變動這兩個檔案，只需要專注在模型的建構上就好了，尤其是在dataset.py中，讀資料的方法(讀取.npy，直接是numpy array)與排序(`np.ascontiguousarray`)對同個模型可能具有不同的資料輸入速度，很高興我在嘗試過別人的model後有看到極高的training速度，再次撰寫dataset.py時有努力做到差不多的效果。

在讀程式碼中，一開始真的對於全部包好的模型好不習慣，很多檔案都分別放置，常常需要展開非常多個分頁查看程式碼，但仍可能看不懂他在做些甚麼QQ，後來學習到這才是寫一個模型的好習慣，將檔案分開，讓每個檔案有專注做的事情，在更改、查找上都會具有較快的速度，在後期就有比較習慣這種模式，再查看DRLN和EDSR可以比較快找尋到想要的檔案，像：模型與他使用到的block位置，還有README真的幫助很大，他會顯示這個模型的大概架構與最終成績和如何train、test檔案等等。

在模型架構上因為想要達到bonus的標準同時可以拿到PSNR = 32，所以一開始就從輕量級的模型入手，砍channel數量砍module數量，發現結果太慘又把他們給加回來，非常非常苦惱要怎麼在有限的算力下達到一定的複雜度，看了很多cnn code，大部分都是使用縮圖，但此次專題是以每個像素點為評分標準，看到Mobilnet與efficientnet疊了非常多層但算量很低，除了縮圖以外他們還使用到了depthwise seperable convolution，於是轉戰depthwise convolution，整體的算量真的下降很多，為了具有一定複雜度增加module數量、增加channel數量，最後才可以達標。

在此次期末專題中learning rate的調整也非常重要本來使用固定的learning rate，後來發現loss一直降不下去，開始變動learning rate的大小，由於需要訓練很多個epoch，所以設定在前1000~500個epoch中以CosineAnnealingLR的方式從0.001緩慢下降至0.0003，但也需要看模型的狀況調整。

嗯雖然NVIDIA執行長黃仁勳曾經說過：“the more you buy, the more you save.”，GPU的出現是用來省電省錢的，但自從入手了一台具有RTX的筆電後，好像時不時就想著不然爆train一波好了，所以這次期末專題大概前前後後有train超過50個模型，前幾個甚至都train到4000個epoch，總時長大概具有快一星期吧(對不起筆電，我真的不是故意讓你時不時就燒到90度的)，所以可能這句話到我這邊會變成the more I buy, the more I train，好像沒有特別省到電力XD。

完成final project真的很開心，在這次的期末專題真的學到很多、也進步很多，還好有比較早開始做這樣才有很多時間找model、學習看整份model code，比較可惜的是最後沒有嘗試到transformer架構，但真的在過程中收穫很大。

對AI training課的感想

首先真的很感謝每一位上課的助教，我真的從來沒有想過可以在一個暑假中學會train model、有能力可以寫出兩份projects和多份作業等等，助教們辛苦了。

覺得整個訓練內容很完整，以期中專題為分界，前半部分是CNN-based後面開始介紹Transformer相關內容，讓我們了解AI發展、應用與最新的技術。

會希望在CNN-based部分可以多增加整份程式的撰寫，實話說我到期中專題的時候才比較知道一份程式會需要些甚麼，不然真的常常漏東漏西，會希望助教和我們分享建構模型的技巧，convolution可以怎麼疊、通常一個模型怎樣的標準才算有料等等。

在後半部分，具有非常多概念的東西，對於很多模型很多技術都會覺得很厲害，會希望助教們可以帶我們看一下RNN、LSTM、Transformer的code等等，這樣除了了解架構的重要概念與理論以外，還可以知道他是如何被實作出來的、如何打出來的。

謝謝助教們和老師精心安排的課程，也謝謝互相在performance上battle的同學們，真的收穫很多！

建構dataset和train architecture

程式碼在notion輸出的部分會較醜，可以請助教點選下面連結

- [train architecture](#)

download training data

1. DIV2K

DIV2K Dataset

Radu Timofte,
Eirikur Agustsson,
Shuhang Gu,

<https://data.vision.ee.ethz.ch/cvl/DIV2K/>

看好放大倍率x3下載LR檔案，高解析度的HR檔都是相同的

在windows系統中只要點擊連結就可以下載

在linux系統中可以，右鍵點選連結、複製連結，在linux terminal中打上

```
wget 網址
```

解壓縮

```
unzip filename.zip
```

2. Flickr2k

下載網址:<http://cv.snu.ac.kr/research/EDSR/Flickr2K.tar>

非常的胖(20多G)大概要下載1hr

```
wget http://cv.snu.ac.kr/research/EDSR/Flickr2K.tar --no-check-certificate
```

解壓縮

```
tar xvf Flickr2K.tar
```

Dataset

dataset中具有三個非常重要的元素分別是，`init`、`getitem`、`len`

__init__

```
class DIV2K(Dataset):
    def __init__(self, root, patch_size = 192, mode = "train", repeat = 1):
        super(DIV2K, self).__init__()
        self.root = root
        self.mode = mode
        self.repeat = repeat
        self.images_hr = natsorted(os.listdir(os.path.join(self.root, 'DIV2K_{}_{}_HR'.format(self.mode))))
        self.images_lr = natsorted(os.listdir(os.path.join(self.root,
                                                        'DIV2K_{}_{}_LR_bicubic'.format(self.mode), 'X3')))
        self.hr_list = []
        self.lr_list = []
        for i in self.images_hr:
            self.hr_list.append(os.path.join(self.root, 'DIV2K_{}_{}_HR'.format(self.mode), i))
        for i in self.images_lr:
            self.lr_list.append(os.path.join(self.root, 'DIV2K_{}_{}_LR_bicubic'.format(self.mode), 'X3', i))
        self.patch_size = patch_size
```

`init`部分會將高解析、低解析資料的路徑都處理好，先使用 `os.path.join` 找到檔案資料夾，`os.listdir` 列出資料夾下的所有檔案，`natsorted` 將他們依檔案名稱做排序，接著創建成兩個list分別是 `self.hr_list` 和 `self.lr_list`，存放所有data的完整路徑

__getitem__

```
def __getitem__(self, idx):
    img_hr, img_lr = self.load_file(idx)
    img_hr, img_lr = self.get_patches(img_hr, img_lr)
    img_hr, img_lr = self.toTensor(img_hr, img_lr)
    return img_hr, img_lr
```

`getitem`部分會將model input和target做配對，

首先在 `self.load_file` 中會拿取 `self.hr_list` 和 `self.lr_list` 中在 `idx % len(image_hr)` 的資料

get_patches

```
def get_patches(self, img_hr, img_lr):
    patch_size = self.patch_size
    scale = 3

    # t for target, i for input
    tp = patch_size
    ip = tp // scale
    ih, iw, _ = img_lr.shape

    ix = random.randrange(0, iw - ip + 1)
    iy = random.randrange(0, ih - ip + 1)
    tx, ty = scale * ix, scale * iy

    img_lr = img_lr[iy:iy + ip, ix:ix + ip, :]
    img_hr = img_hr[ty:ty + tp, tx:tx + tp, :]
    return img_hr, img_lr
```

再來在get_patches中，因為dataloader預設不處理在同一個batch中不同大小的input，但download下來的data size都不盡相同，所以可以做將image切成同樣大小的patch，就可解決這個問題

toTensor

```
def toTensor(self, img_hr, img_lr):
    img_hr = np.ascontiguousarray(img_hr.transpose(2, 0, 1))
    img_hr = torch.from_numpy(img_hr).float() / 255.

    img_lr = np.ascontiguousarray(img_lr.transpose(2, 0, 1))
    img_lr = torch.from_numpy(img_lr).float() / 255.
    return img_hr, img_lr
```

因為在讀取檔案的部分，避免每一次訓練都需要使用cv2進行讀檔，所以在某一Github的model，提供png2npy.py，將.png直接轉成numpy array並儲存

故在toTensor的程式中，和用cv2讀檔較不一樣的地方是，不需要將channel的順序從BGR轉成RGB，但還是需要將HWC的結構，轉成CHW方便之後訓練

在訓練中遇到很大的問題是為甚麼感覺別人的code和我的code差不多，但他人的運行速度就是比較快，除了先將資料進行預處理以外，這邊使用到了 `np.ascontiguousarray` 的技巧，`np.ascontiguousarray` 是將原先不連續的儲存方法，變成連續的儲存，在資料的拿取上就會較為快速，當時處理速度可從35it/s到70it/s，相差極大

train architecture

```
def train(start_epoch, n_epochs):
    global psnr_best
    model.to(device)
    for epoch in range(start_epoch, n_epochs + 1):
```

```

        print('Epoch: {}'.format(epoch), 'lr: {}'.format(optimizer.param_groups[0]['lr']))
        train_loss, lrs= train_epoch()
        print('Train Loss: {}'.format(train_loss))

        with torch.no_grad():
            val_psnrs = val_epoch()
            print('Val PSNR: {}'.format(val_psnrs.mean()))
            test_psnrs = test_epoch()
            print('Test PSNR: {}'.format(test_psnrs.mean()))
        # ...
        save_checkpoint(epoch, model, optimizer, psnr_best, history)
    return history

```

首先train內需要有每個epoch做了些甚麼事情，包含計算train_loss、val_psnr、test_psnr，故寫了三個函式train_epoch()、val_epoch()、test_epoch()分別計算

```

def train_epoch():
    model.train()
    train_loss = 0
    lrs = []
    for idx, (img_hr, img_lr) in enumerate(tqdm(train_dataloader)):
        img_hr = img_hr.to(device)
        img_lr = img_lr.to(device)
        output = model(img_lr)
        loss = criterion(output, img_hr)
        train_loss += loss.item()
        lrs.append(optimizer.param_groups[0]['lr'])
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    train_loss /= len(train_dataloader)
    return train_loss, lrs

def val_epoch():
    model.eval()
    psnrs = []
    for idx, (img_hr, img_lr) in enumerate(tqdm(val_dataloader)):
        img_hr = img_hr.to(device)
        img_lr = img_lr.to(device)
        with torch.no_grad():
            output = model(img_lr)
            psnr = psnr_tensor(output * 255, img_hr * 255)
            psnrs.append(psnr)
    return torch.tensor(psnrs)

```

在train_epoch()中需要命令model.train()、在val_epoch()中需要命令model.eval()

```

def test_epoch():
    model.eval()
    psnrs = []
    for idx, data in enumerate(test_dl):

```

```

    img_name = data[0][0]
    img_lq = data[1].to(device)
    img_gt = data[2].to(device)
    with torch.no_grad():
        _, _, h_old, w_old = img_lq.size()
        h_pad = (2 ** math.ceil(math.log2(h_old))) - h_old
        w_pad = (2 ** math.ceil(math.log2(w_old))) - w_old
        img_lq = torch.cat([img_lq, torch.flip(img_lq, [2])], 2)[ :, :, :h_old + h_pad, :]
        img_lq = torch.cat([img_lq, torch.flip(img_lq, [3])], 3)[ :, :, :, :w_old + w_pad]
        output = demo_UHD_fast(img_lq, model)
        preds = (output[:, :, :h_old*3, :w_old*3].clamp(0, 1) * 255).round()

    img_gt = (img_gt[:, :, :h_old*3, :w_old*3] * 255.).round()

    psnr = psnr_tensor(preds, img_gt)
    psnrs.append(psnr)
    # print(f'Testing: {img_name:20s}, PSNR: {psnr}')
return torch.tensor(psnrs)

def save_checkpoint(epoch, model, optimizer, psnr_best, history, dir='checkpoint'):
    state = {'epoch': epoch,
             'model': model.state_dict(),
             'optimizer': optimizer.state_dict(),
             'psnr_best': psnr_best,
             'history': history}
    torch.save(state, os.path.join(dir, 'checkpoint.pth'))

result = {'train_loss': float, 'val_psnr': float, 'test_psnr': float, 'lrs': []}
history = []
resume = './checkpoint/checkpoint.pth' #./checkpoint/checkpoint.pth

```

test_epoch()部分使用助教提供的範例code，計算Set5的PSNR-RGB