AlexNet

×

PyTorch 實作

嗨，

我是一位就讀物理系

可愛的大學生甯敬宇

論文		實作	
LeNet	導入技術	PyTorch	AlexNet
AlexNet	歷史意義	參數儲存	基本應用

首先，

再談LeNet

LeNet

Input (32x32)

Convolution 1 (6x28x28)

Subsample 2 (6x14x14)

Convolution 3 (16x10x10)

Subsample 4 (16x5x5)

Convolution 5 (120*1*1)

Full Connection 6 (84)

Output (10)

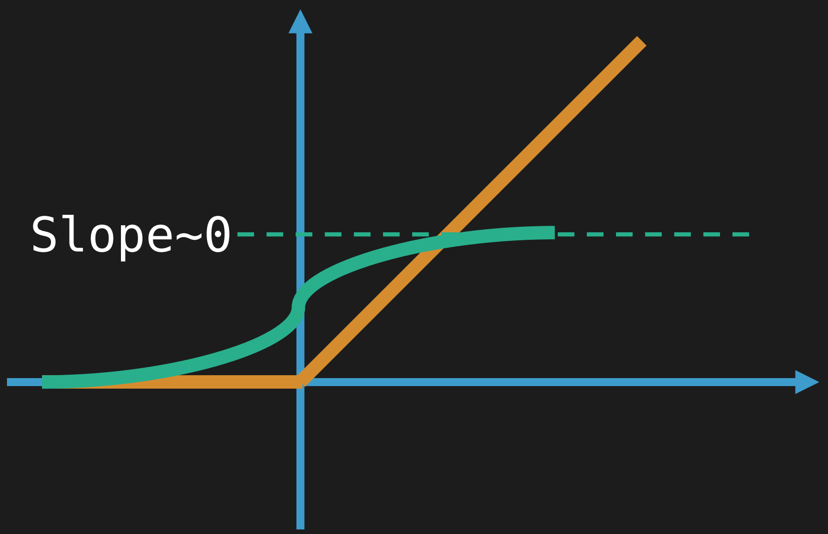
相較於LeNet，

AlexNet導入許多技術

Sigmoid在神經頓化時，

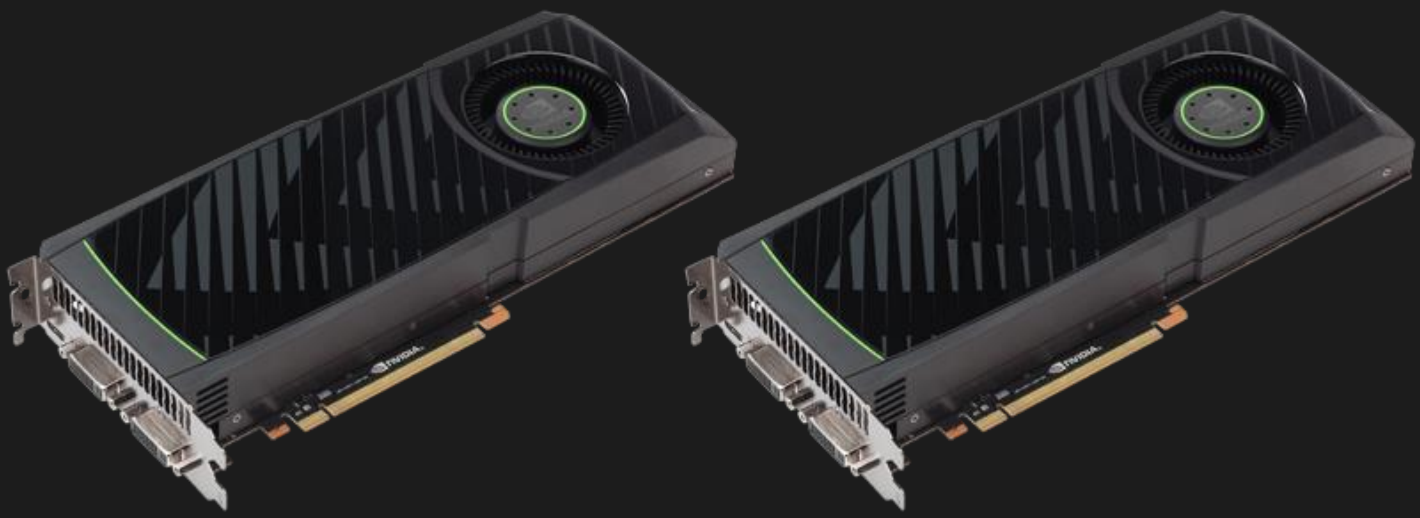
導數將趨近於零—梯度消失！

改用非線性激活函數ReLU



為了增加效率，

使用使用多顆高效GPU



採用重疊池化，

來強調特徵

在數據方面

也加入許多新觀點

使用百萬級的數據集

如ImageNet

「儘管很多人都在注意模型，但我們要關心數據，數據將重新定義我們對模型的看法」

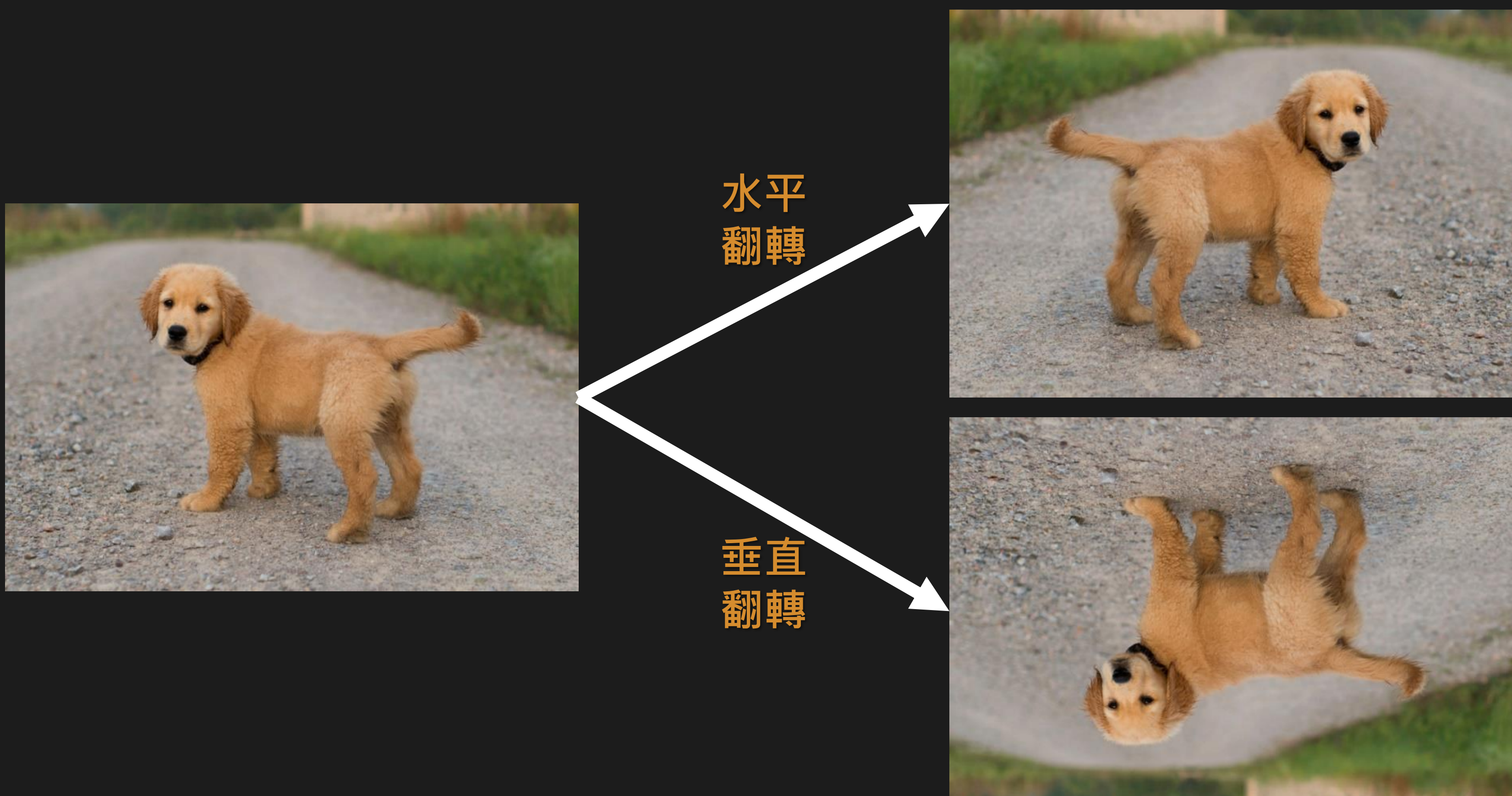
--李飛飛

導入Data augmentation

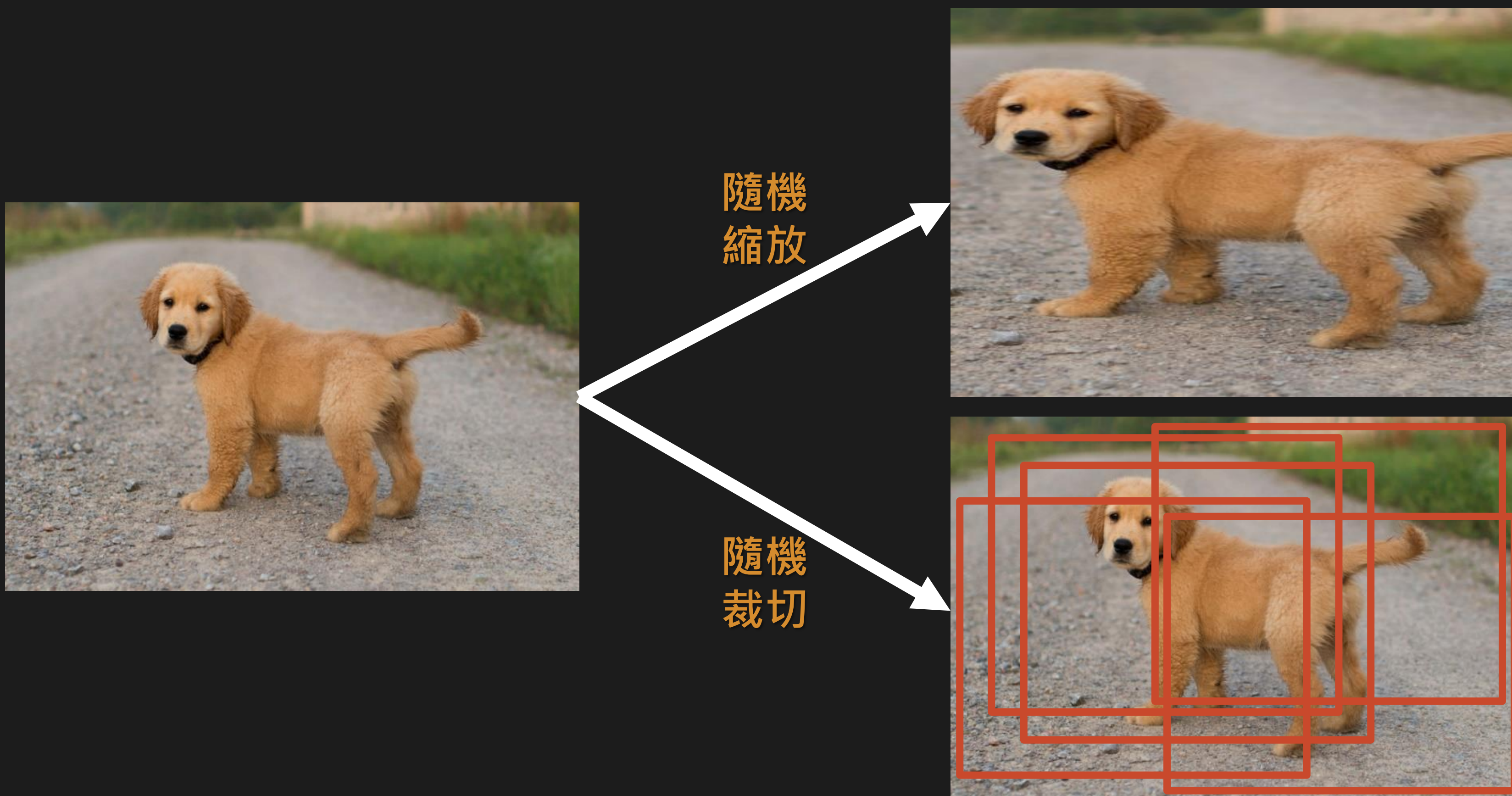
增加隨機特徵亂度

- Flip
- Random Rescale/Crop
- Color Jittering

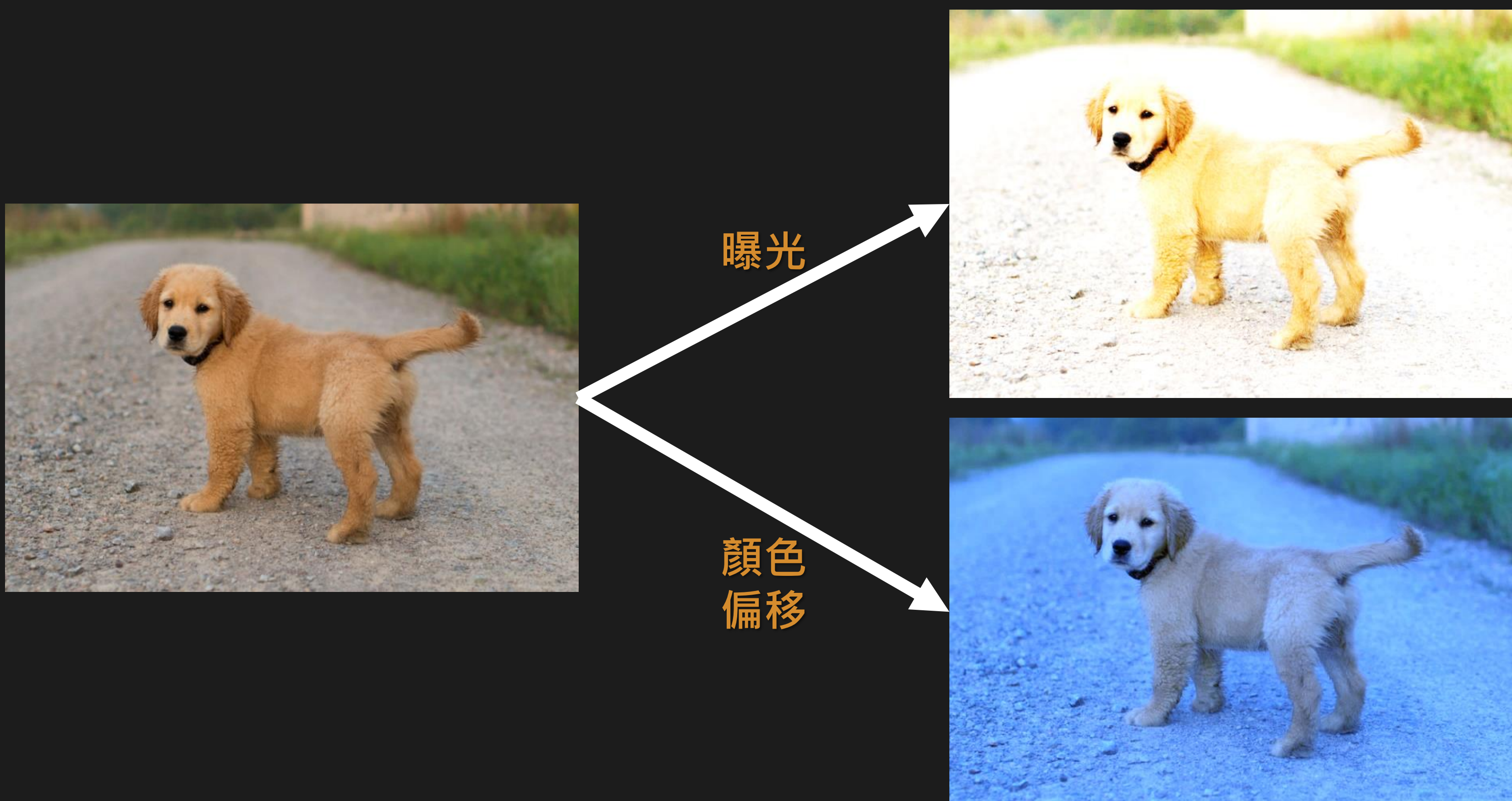
Flip



Random Rescale/Crop



Color Jittering



有條件下，

導入Dropout

- 增大數據量
- 增加Epoch

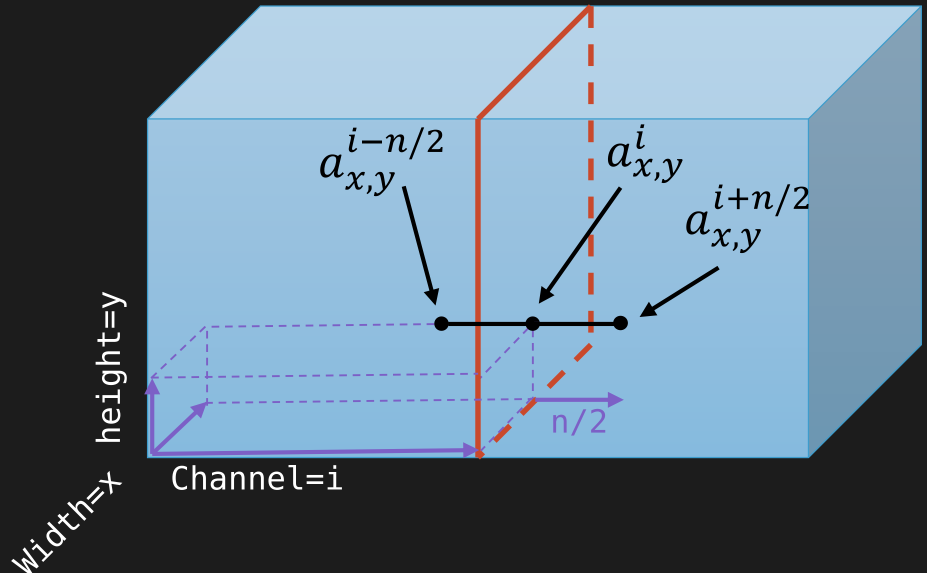
沒了~~

ㄟ… 等等，

好像少了甚麼~~

神奇的LRN

$$b_{x,y}^{i} = \frac{a_{x,y}^{i}}{\left(k + \alpha \sum_{j=max\left(0,i-\frac{n}{2}\right)}^{min\left(N-1,i+\frac{n}{2}\right)} \left(a_{x,y}^{j}\right)^{2}\right)^{\beta}}$$



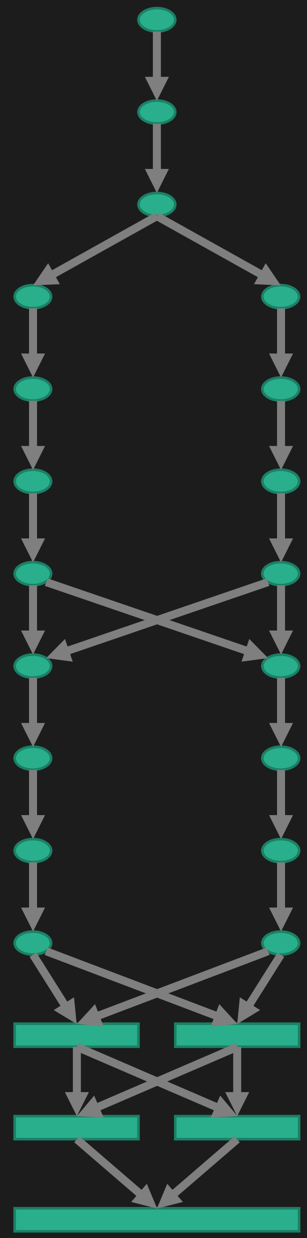
簡單的說，以$\tilde{b}_{x,y}^{i}$為準，

在α、β、k固定時

$$\tilde{b}_{x,y}^{i}=\frac{a_{x,y}^{i}}{\left(k+\alpha\sum_{j=max\left(0,i-\frac{n}{2}\right)}^{min\left(N-1,i+\frac{n}{2}\right)}\left(a_{x,y}^{i}\right)^{2}\right)^{\beta}}$$

不過，真的有用嗎？

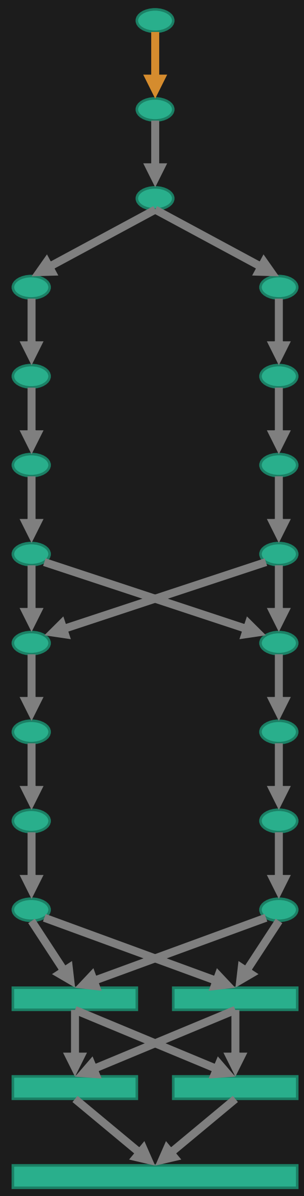
AlexNet



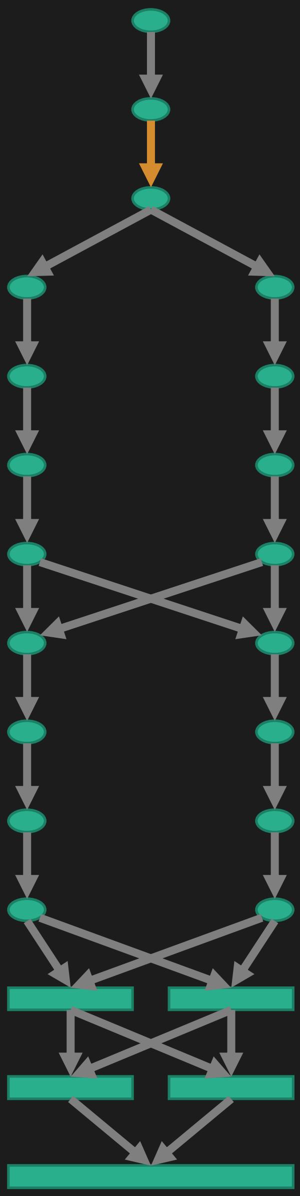
Data Augmentation
Convolutional Layer 1
拆分至2個GPU
Pooling Layer 1
Convolutional Layer 2
Pooling Layer 2
Convolutional Layer 3
Convolutional Layer 4
Convolutional Layer 5
Pooling Layer 2
Full-Connecting Layer
Full-Connecting Layer
Output

Data Augmentation

- 輸入：3X256x256
- 隨機裁切227x227大小的圖片
- 隨機將圖片水平翻轉
- 色彩微擾
- 輸出：3x227x227



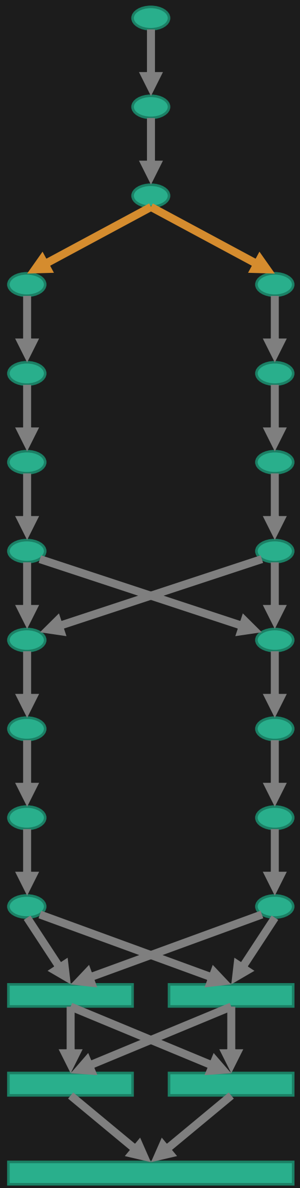
Convolutional Layer 1



- 輸入：3X227x227
- 使用96個3x11x11的filter捲積
- 捲積步長為4
- 捲積後的feature大小: $\frac{227-11}{4}+1=55$
- 輸出：96x55x55

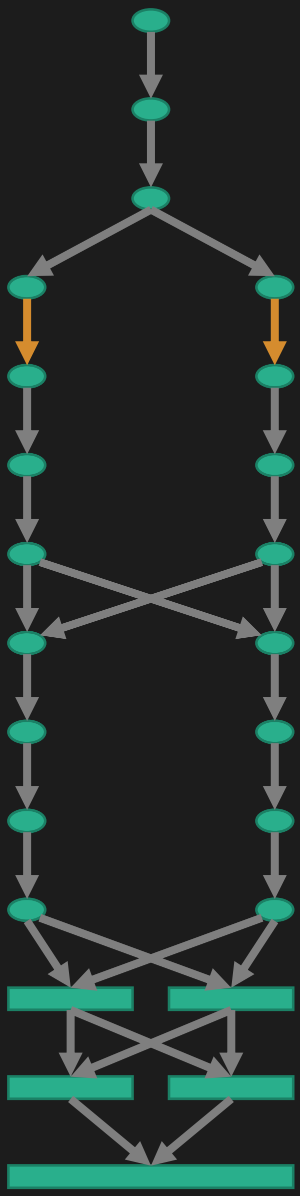
拆分至2個GPU

- NVIDIA GTX 580 3GB
- 輸入：1x96x55x55
- 輸出：2x48x55x55



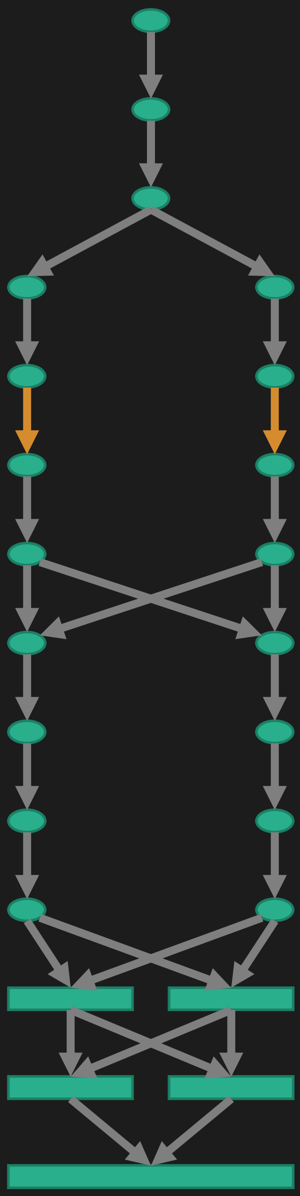
Pooling Layer 1

- 輸入：2x48x55x55
- 使用3x3的kernel做最大池化
- 池化步長為2 (重疊池化)
- 池化後的feature大小:$\frac{55-3}{2}+1=27$
- 輸出：2x48x27x27



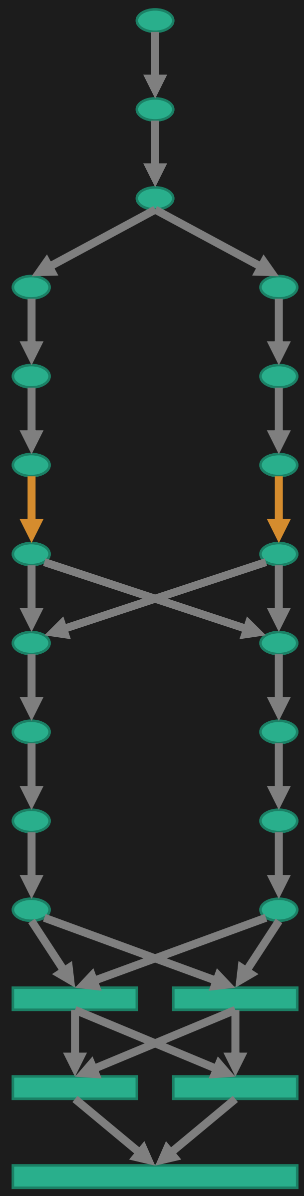
Convolutional Layer 2

- 輸入：2x48x27x27
- Padding: 27+2(2)=31
- 使用256個48x5x5的filter捲積
- 捲積步長為1
- 捲積後的feature大小:$\frac{31-5}{1}+1=27$
- 輸出：2x128x27x27



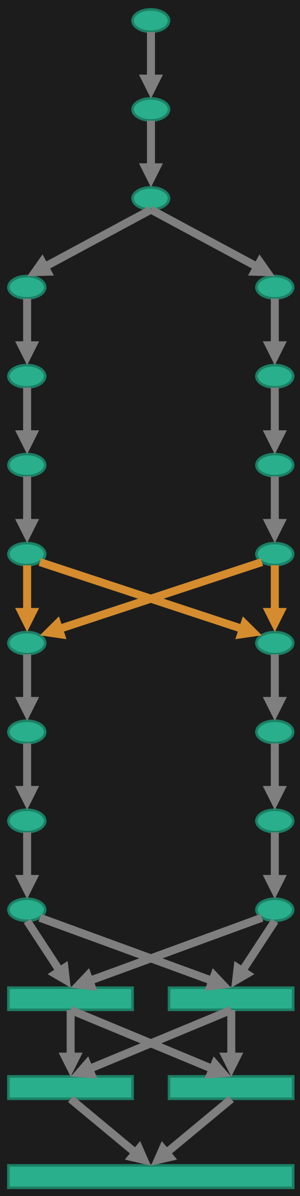
Pooling Layer 2

- 輸入： 2x128x27x27
- 使用3x3的kernel做最大池化
- 池化步長為2 (重疊池化)
- 池化後的feature大小:$\frac{27-3}{2}+1=13$
- 輸出：2x128x13x13



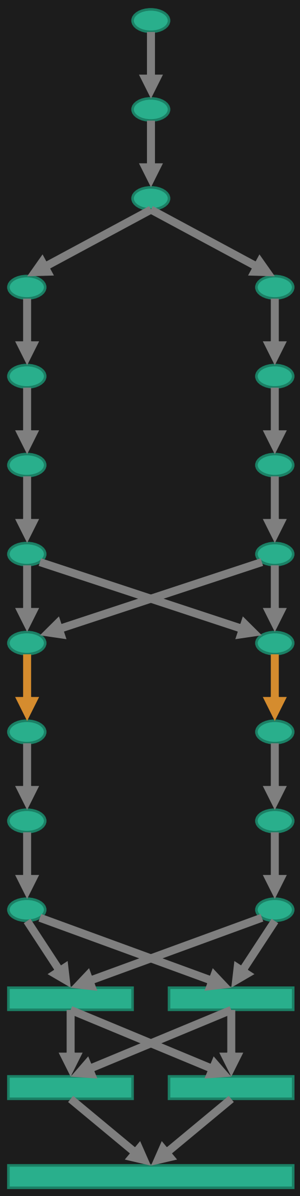
Convolutional Layer 3

- 輸入：2x128x13x13
- Padding: 13+2(1)=15
- 使用384個256x3x3的filter捲積
- 捲積步長為1
- 捲積後的feature大小:$\frac{15-3}{1}+1=13$
- 輸出：2x192x13x13



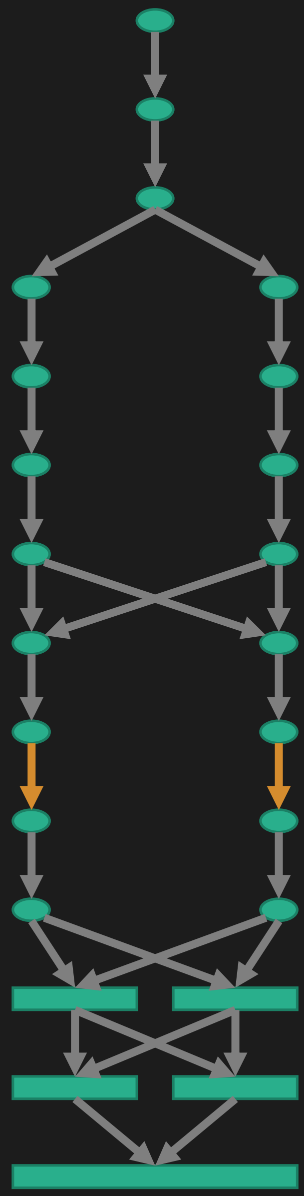
Convolutional Layer 4

- 輸入：2x192x13x13
- Padding: 13+2(1)=15
- 使用384個192x3x3的filter捲積
- 捲積步長為1
- 捲積後的feature大小:$\frac{15-3}{1}+1=13$
- 輸出：2x192x13x13



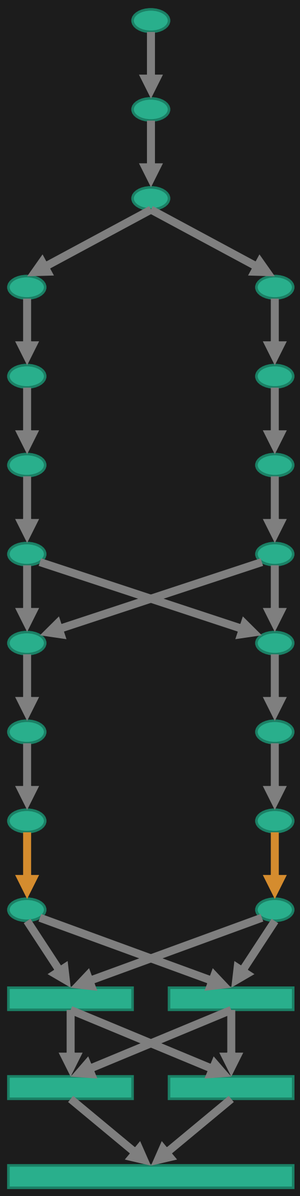
Convolutional Layer 5

- 輸入：2x192x13x13
- Padding: 13+2(1)=15
- 使用256個192x3x3的filter捲積
- 捲積步長為1
- 捲積後的feature大小:$\frac{15-3}{1}+1=13$
- 輸出：2x128x13x13



Pooling Layer 5

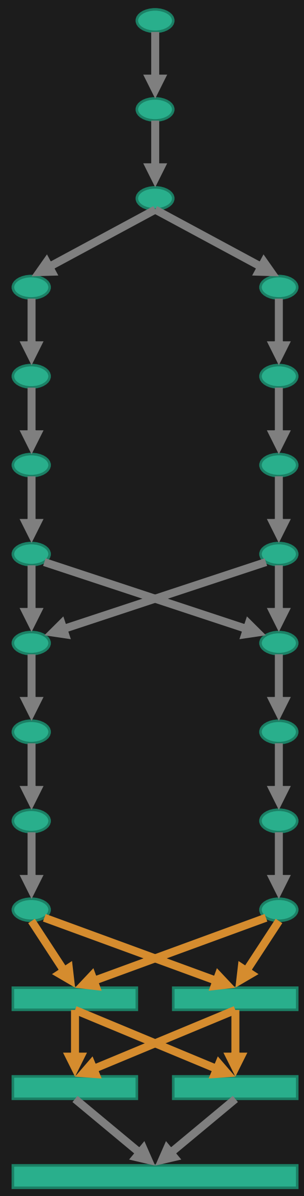
- 輸入： 2x128x13x13
- 使用3x3的kernel做最大池化
- 池化步長為2 (重疊池化)
- 池化後的feature大小:$\frac{13-3}{2}+1=6$
- 輸出：2x128x6x6



Full-connecting Layers

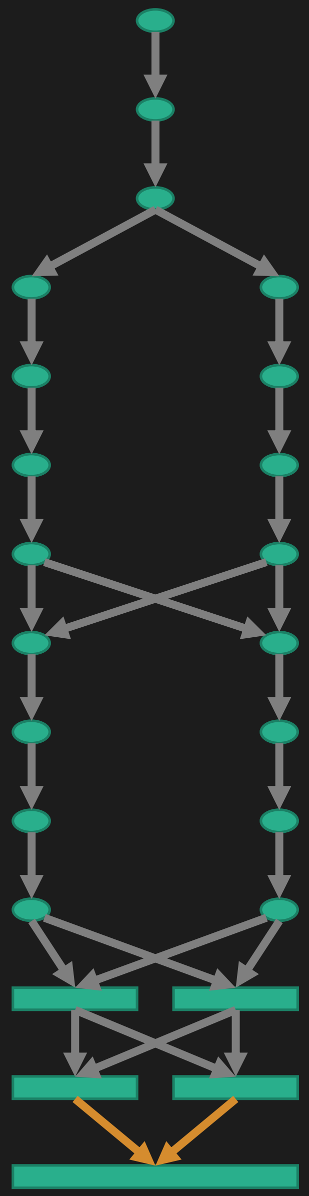
- 輸入： 2x4608
- Dropout: 50%
- 輸出：2x2048

- 輸入： 2x2048
- Dropout: 50%
- 輸出：2x2048



Output Layer

- 輸入： 2x2048
- 輸出：1000



AlexNet
的歷史意義

- 證明海量的數據對深度學習的重要
- 導入GPU
- 加深網絡有助於提升準確性
- 證明Dropout有效性
- 證明資料增強的有效性
- 證明局部響應標準化沒有助於降低誤差

簡單介紹

PyTorch

PyTorch的優勢

- 強調代碼的可讀性和簡潔的語法
- 與Numpy有高度相容性
- 易於使用GPU

一個簡單的範例

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py

堆疊Layer的兩種方式

定義 排序

定義

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

排序

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.all_conv = nn.Sequential(
            nn.Conv2d(3, 6, 5),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(6, 16, 5),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),)
        self.all_fc = nn.Sequential(
            nn.Linear(16 * 5 * 5, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10),)

    def forward(self, x):
        x = self.all_conv(x)
        x = x.view(-1, 16 * 5 * 5)
        x = self.all_fc(x)
        return x
```

AlexNet實作

https://github.com/forrestning/AlexNet/blob/master/AlexNet_monkey.ipynb