

Deep-Learning-for-Autonomous-Driving

Lab3 : Semantic Segmentation

ID : 310605007

Member : 鄭晴立

Department : 機器人學程

1 Introduction

我針對助教提供的建議，盡量使用原本提供的 code 去實現 U-Net，並在原始 UNet 加上以下改進，包含 Loss function、Skip connection 及 Dilated convolution，並由以下幾點說明其改內容。

1.1 U-Net

本次 Lab 為 U-Net 實作，以下為實作的 code。U-Net 分為 Encoder 及 Decoder。Encoder 為五次的雙層 Convolution，我將其定義在 DownConv 裡面，中間會進行 maxpooling。Decoder 部分則進行五次的 Upsampling 及雙層 Convolution，我將其定義在 UpConv。

```
class UNet(nn.Module):
    def __init__(self, n_classes=0, n_channels=3, testing=False, bilinear=False):
        super(UNet, self).__init__()

        later_dim = [64, 128, 256, 512, 1024]

        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear
        self.pool = nn.MaxPool2d(2, 2)

        self.conv0_0 = DownConv(n_channels, later_dim[0])
        self.conv1_0 = DownConv(later_dim[0], later_dim[1])
        self.conv2_0 = DownConv(later_dim[1], later_dim[2])
        self.conv3_0 = DownConv(later_dim[2], later_dim[3])
        self.conv4_0 = DownConv(later_dim[3], later_dim[4])
        factor = 2 if bilinear else 1
        self.conv4_0 = DownConv(later_dim[4], later_dim[4] // factor)

        self.conv4_1 = UpConv(later_dim[4], later_dim[3] // factor, bilinear)
        self.conv3_1 = UpConv(later_dim[3], later_dim[2] // factor, bilinear)
        self.conv2_1 = UpConv(later_dim[2], later_dim[1] // factor, bilinear)
        self.conv1_1 = UpConv(later_dim[1], later_dim[0] // factor, bilinear)
        self.conv0_1 = nn.Conv2d(later_dim[0], n_classes, kernel_size=1)

    def forward(self, x):
        x0_0 = self.conv0_0(x)
        x1_0 = self.conv1_0(self.pool(x0_0))
        x2_0 = self.conv2_0(self.pool(x1_0))
        x3_0 = self.conv3_0(self.pool(x2_0))
        x4_0 = self.conv4_0(self.pool(x3_0))

        x4_1 = self.conv4_1(x4_0, x3_0)
        x3_1 = self.conv3_1(x4_1, x2_0)
        x2_1 = self.conv2_1(x3_1, x1_0)
        x1_1 = self.conv1_1(x2_1, x0_0)
        x0_1 = self.conv0_1(x1_1)

        return x0_1
```

圖一 U-Net code

```
class UpConv(nn.Module):
    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DownConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
            self.conv = DownConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]
        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)
```

圖二 Upsampling of U-Net

```
class DownConv(nn.Module):
    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, dilation=2, padding=2, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, dilation=2, padding=2, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)
```

圖三 Convolution of U-Net

1.2 Dilated convolution

由於時間關係，沒有單獨測試 Dilated convolution，僅隨意挑選一個尺度進行實驗。

```
class DownConv(nn.Module):
    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, dilation=2, padding=2, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, dilation=2, padding=2, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)
```

圖四 Dilated convolution code

1.3 Loss Function

由於本次的所要解決的是語意分割的問題，因此有幾個較為經典的 Loss Function，包含 dice soft loss、soft iou loss 及 Lovasz softmax loss，由於時間關係，僅嘗試 dice soft loss、Lovasz softmax loss 及傳統的 Cross Entropy Loss。圖 五 為 Lovasz softmax loss 節錄。

```
def lovasz_softmax(probas, labels, classes='present', per_image=False, ignore=None):
    if per_image:
        loss = mean(lovasz_softmax_flat(*flatten_probas(prob.unsqueeze(0), lab.unsqueeze(0), ignore), classes=classes)
                     for prob, lab in zip(probas, labels))
    else:
        loss = lovasz_softmax_flat(*flatten_probas(probas, labels, ignore), classes=classes)
    return loss
```

圖 五 Lovasz softmax loss

1.4 U-Net++

1.4.1 U-Net++ Introduction

我找了幾篇 U-Net 相關文獻後，發現 U-Net++ 為基於 U-Net 做改進的代表性文章，而我將採用其方法嘗試。U-Net++ 為 2019 發表的一篇文獻，作者認為，傳統 U-Net 的 Skip connection 是有改進空間的，UNet 僅將同一層級的特徵串接在一起，因此 U-Net++ 將許多不同 scale 的 UNet 模塊串接在一起，讓不同的特徵之間能都流通。圖 六 為 UNet++ 的 code。

```
class UNetPlus(nn.Module):
    def __init__(self, num_classes=8, input_channels=3, **kwargs):
        super().__init__()

        layer_dim = [64, 128, 256, 512, 1024]

        self.pool = nn.MaxPool2d(2, 2)
        self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)

        self.conv0_0 = DownConv(input_channels, layer_dim[0])
        self.conv1_0 = DownConv(layer_dim[0], layer_dim[1])
        self.conv2_0 = DownConv(layer_dim[1], layer_dim[2])
        self.conv3_0 = DownConv(layer_dim[2], layer_dim[3])
        self.conv4_0 = DownConv(layer_dim[3], layer_dim[4])

        self.conv0_1 = DownConv(layer_dim[0]+layer_dim[1], layer_dim[0])
        self.conv1_1 = DownConv(layer_dim[1]+layer_dim[2], layer_dim[1])
        self.conv2_1 = DownConv(layer_dim[2]+layer_dim[3], layer_dim[2])
        self.conv3_1 = DownConv(layer_dim[3]+layer_dim[4], layer_dim[3])

        self.conv0_2 = DownConv(layer_dim[0]*2+layer_dim[1], layer_dim[0])
        self.conv1_2 = DownConv(layer_dim[1]*2+layer_dim[2], layer_dim[1])
        self.conv2_2 = DownConv(layer_dim[2]*2+layer_dim[3], layer_dim[2])

        self.conv0_3 = DownConv(layer_dim[0]*3+layer_dim[1], layer_dim[0])
        self.conv1_3 = DownConv(layer_dim[1]*3+layer_dim[2], layer_dim[1])

        self.conv0_4 = DownConv(layer_dim[0]*4+layer_dim[1], layer_dim[0])

        self.final1 = nn.Conv2d(layer_dim[0], num_classes, kernel_size=1)
        self.final2 = nn.Conv2d(layer_dim[0], num_classes, kernel_size=1)
        self.final3 = nn.Conv2d(layer_dim[0], num_classes, kernel_size=1)
        self.final4 = nn.Conv2d(layer_dim[0], num_classes, kernel_size=1)

    def forward(self, input):
        # UNet++ L1
        x0_0 = self.conv0_0(input)
        x1_0 = self.conv1_0(self.pool(x0_0))
        x0_1 = self.conv0_1(torch.cat([x0_0, self.up(x1_0)], 1))
        # UNet++ L2
        x2_0 = self.conv2_0(self.pool(x1_0))
        x1_1 = self.conv1_1(torch.cat([x1_0, self.up(x2_0)], 1))
        x0_2 = self.conv0_2(torch.cat([x0_0, x0_1, self.up(x1_1)], 1))
        # UNet++ L3
        x3_0 = self.conv3_0(self.pool(x2_0))
        x2_1 = self.conv2_1(torch.cat([x2_0, self.up(x3_0)], 1))
        x1_2 = self.conv1_2(torch.cat([x1_0, x1_1, self.up(x2_1)], 1))
        x0_3 = self.conv0_3(torch.cat([x0_0, x0_1, x0_2, self.up(x1_2)], 1))
        # UNet++ L4
        x4_0 = self.conv4_0(self.pool(x3_0))
        x3_1 = self.conv3_1(torch.cat([x3_0, self.up(x4_0)], 1))
        x2_2 = self.conv2_2(torch.cat([x2_0, x2_1, self.up(x3_1)], 1))
        x1_3 = self.conv1_3(torch.cat([x1_0, x1_1, x1_2, self.up(x2_2)], 1))
        x0_4 = self.conv0_4(torch.cat([x0_0, x0_1, x0_2, x0_3, self.up(x1_3)], 1))

        output_l1 = self.final1(x0_1) # L1 output
        output_l2 = self.final2(x0_2) # L2 output
        output_l3 = self.final3(x0_3) # L3 output
        output_l4 = self.final4(x0_4) # L4 output
        return [output_l1, output_l2, output_l3, output_l4]
```

圖 六 UNet++ code

1.4.2 U-Net++ Architecture

U-Net++ 的架構可分解成多 (L1~L4) 層 U-Net 組合而成，可對應到圖 六 和 圖 七 的 L1 到 L4，可視為將模型加廣，並加強各層之間特徵的連結。

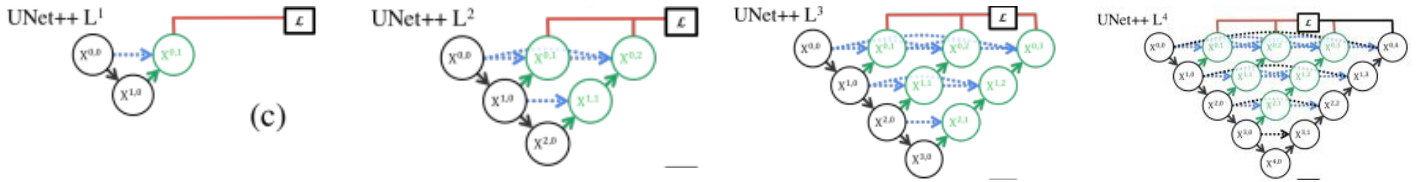


圖 七 UNet++ Architecture

1.4.3 Deep supervision

另一個重點於 U-Net++ 是 Deep supervision，因為這樣的模型會有兩個問題：

- 如果用最後一層的單一輸出，會導致模型 train 不起來。
- 由於我們將模型加廣，表現變好也是理所當然的，相對模型大小也會變大很多。

有關第一點，我已經實際嘗試過，確實會 train 不起來；而有關第二點，由於我們將模型加廣，導致表現變好也是理所當然的，因此透過 Deep supervision 的機制可以去觀察深度的影響力，藉此刪除不必要的層數，其實際操作方式為，在 validation 階段時，去觀察 L3 及 L4 的差異，若差異太小，則代表 L4 層可被刪除，因此在最後輸出 model 將最後一層刪掉，但因為其在 backward 也會有幫助，因此不會一開始就刪掉。而我在實作上，並沒有額外做刪除的動作，僅將其各層 Loss 加起來，使其 model 能夠 train 起來。

```
preds = Net.forward(data)
loss = 0
for output in preds:
    if loss_fn == 0:
        lo=torch.nn.CrossEntropyLoss(ignore_index=255)
        loss += lo(output, target)
    elif loss_fn == 1:
        # print(L.lovasz_softmax(output, target,ignore=255))
        loss += L.lovasz_softmax(output, target,ignore=255)
        # loss += criterion(output, target)
pred=preds[-1]
loss /= len(preds)
```

圖 八 Deep supervision 下的 loss 運作方式

1.5 Learning rate schedule

由於這次的圖片所要 train 的時間比較久，因此需要透過 Learning rate schedule，能夠增加 training 效率，我僅做簡單的 schedule，能夠稍微改善其效率。

```
if epoch > 80:
    lr=0.001
elif epoch > 30 :
    lr=0.1
else :
    lr=0.01
```

圖 九 Learning rate schedule

2 Result

2.1 Implement



我將前章節的修正做一個結論，mIOU 為 “ - ” 代表 train 不起來，或是前幾個 epoch 表現就非常差，因為時間不夠，所以沒時間一一挑出問題，不一定代表其改進是無效的。

我僅做出 U-Net++ 及傳統 U-Net，而且還沒有達到預期的效果，而其他的 Loss Function 及 Dilation，都因為某些問題，無法完全確定其效果如何。

表 一 個修正方式之比較

Model	Loss	Dilation	mIOU
U-Net	Cross Entropy	No	0.838307
U-Net	Lovasz softmax	No	-
U-Net	dice soft loss	No	-
U-Net	Cross Entropy	Yes	-
U-Net++	Cross Entropy	Yes	-
U-Net++	Cross Entropy	No	0.827237

表 二 預測結果

Model	Image
U-Net	
U-Net++	

2.2 Conclusion

從結果可以看出，我的各項調整皆無法使表現優於原始 U-Net，不管是 mIOU、還是肉眼，包含 loss function、及 model 的改善，大概就像我的人生一樣，庸庸碌碌、徒勞無功、超級可悲。

3 Discussion

3.1 Log file problem

整體的 code 都盡量依照助教原先所提供去修正，因此在 log file 裡面產生了蠻多看不太懂得檔案，希望未來助教能提供 code 的說明。

3.2 UNet++

一開始使用傳統 U-Net 時 validation 的 mIOU 已經可以到 0.8 左右了，後續使用其他修正皆很難往上升，或是需要花很多時間。

3.3 Testing 時會 CUDA out of memory

因為要求 testing 時要用原始大小，我用家中 12G 的 3060 顯卡仍然無法 testing，希望未來不要用原始大小的圖片去做 testing。

3.4 mIOU 的提升的影響

由於傳統 U-Net 方法已經可以達到不錯的效果，目前嘗試的幾種方法，以及文獻上所看到的方法，大多僅能提升一些效能，mIOU 在 0.7~0.85 之間，對於肉眼來說，幾乎是看不出差別的，因此可能需要捨棄傳統 U-Net 架構，使用更突破性的方法才有辦法更加提升表現性。

3.5 Upsample 的方法對於 mIOU 的影響

似乎不同的 Upsample 方式會對 mIOU 有些許的影響，嘗試了 transform 的 resize、nn 的 Usample 以及助教提供的方式，test 出來的 mIOU 會有些許不同，但沒時間實際去比較。

3.6 Code 的差異

我和同學分別在不同的時間下載助教提供的 code，裡面 code 內容似乎有些許的不同，以至於我們在 testing 的時候會遇到一些相同，但結果不同的奇怪的狀況。