國立清華大學資訊工程學系111學年度2學期專題報告

專題名稱	SensatUrban:3D semantic segmentation challenge with limited resources				
參加競賽或計畫	☐ 參加對外競賽		☐ 參與其他計畫	☑無參加對外競賽或任何計畫	
學號	108062118	108062203	108062315		
姓名	賴翰霖	江浩辰	莊子郁		

摘要

近年來，隨著人工智慧的發展，深度學習也逐漸受到世人關注，而在研究時往往要求機器擁有辨別影像資料的能力，語義分割作為電腦視覺的核心技術之一，其重要性也越來越突出。為了因應現代科技的要求，電腦視覺的場景已經擴展到城市規模，然而對於超大尺度空間，語義分割在效能及穩定性上都還沒能達到要求。為了提供一個辨別能力強、 訓練速度快、穩定性高的訓練模型，我們參考網路上的模型、演算法並進行研究。在研究過程中發現對資料的預處理及採樣方法在模型訓練上有著顯著影響，並朝此方向進行改進，試圖突破語義分割在超大型資料集訓練上的極限，希望藉由這塊領域的進步，廣泛應用在超大型計畫中，比如城市規劃、土地勘查等城市規模的企劃。

中華民國111年11月

目錄

一、研究動機與目的

身為資訊工程學系的學生，我們對於電腦視覺這種實用性高的領域非常感興趣，並且在研讀了各方面電腦視覺領域的 paper 後，我們發現在3D 語義分割（3D Semantic Segmentation）這方面，一直都沒有一個可以達到8-90% mIoU（平均交并比）的模型，普遍在不同的資料集只要5-60%的 mIoU 即是 SOTA model（State of the Arts ），指在特定研究任務或 benchmark 資料集上，目前最突出的 model），甚至這些 SOTA model 在某些占比過小的分類上也只有10-20%的 mIoU，而平均5-60%的表現與人們普遍認為的正確率高也還有一大段距離，相較於人臉辨識或異常偵測等準確率非常高的電腦視覺實務應用，3D 語義分割的表現仍無法在實務面有很好的發揮。

在研讀 Paper 的過程中，我們看到了一個有關3D 語義分割的挑戰與工作坊-- Urban 3D Challenge [1] ，Urban 3D Challenge 為歐洲計算機視覺會議（ECCV 2022）旨在建立新的城市尺度（Urban-Scale）點雲分割基準所推出的挑戰，此挑戰今年為第二年舉辦，有3D 語義分割與3D 實例分割（3D Instance Segmentation）兩個項目的競賽，而在3D 語義分割中此挑戰所使用的資料集為SensatUrban，此資料集為目前世界最大的點雲資料集，內含7.6平方公里中近30億個具有詳細分類並標註的點，其中每個點都被標記為13種語意類別之一，而此點雲最大的分類如地面、建築物、植被總共佔了整個點雲的50%以上，而自行車、鐵軌等類別只佔了總點雲的0.025%，此不平衡也表明了真實場景中不同語意類別分布極為不平均的狀況。

於是我們決定以 Urban 3D Challenge 其中的3D 語義分割研究作為我們這次的專題主題，試著在實驗室較為有限的計算資源下進行訓練，以盡量達到最好的表現。

二、現有相關研究概況及比較

在經過一個學期的研究及討論後，我們對於3D 語義分割有了初步的了解，也嘗試了該如何做加強，我們本次的主題最大的挑戰即是對大量點雲資料集做語義分割，而現有的3D 語義分割主要可以分為三種，其一為基於體素方法（Voxel-based approaches）、其二為基於2D 投影方法（2D projection-based methods），其三為基於點結構（Point-based architectures）。

基於體素方法主要為使用3D 的卷積神經網路（Convolutional Neural Network），需要非常大量的計算和儲存空間，故不易使用於過大的資料集，如我們使用的城市尺度的 SensatUrban 即不太適合。若是使用基於2D 投影的方法，是先將3D 點雲投影至2D 的圖像，之後使用2D 的卷積神經網路來學習語意，最後再投影回3D，然而此方法容易在投影過程中丟失幾何資訊，造成學習成果不佳。而使用基於點結構的網路模型主要是使用多層感知機（Multilayer Perceptron, MLP）逐點來學習語義分割，和前兩者相比，它具有較高的計算效率且能更好的保存每個點的語意特徵，然而較不容易推廣至大型的資料集。

我們針對三個方法逐一進行研究並比較他們的優缺點，分別是基於體素方法的 SparseConv[2]、基於2D 投影的 BEV-Seg-Net[3]和基於點結構的 PointNet[4]、PointNet++[5]、RandLANet[6]和 KPConv[7]。

1. SparseConv ：

   卷積神經網路對2D 圖像處理是非常有效的，然而對於3D 的點雲而言，多數3D 體素都是空的，故而直接用卷積神經網路跑過每一層效率過於低，故而 SparseConv 即為只計算稀疏數據的卷積，以此達到提高效率的目的。SparseConv 本質上是通過建立 Hash 表和 Rule Book，以保存特定位置計算結果。Hash 中的 key 為像素座標、Value 為序號，每行表示一個非空的輸入點。再用 Rule Book 以輸入序號到輸出序號的映射。

   然而，對稀疏的非空體素進行卷積計算的話，通過應用完整的卷積來擴展每層的稀疏數據，可能導致失真。

2. PointNet ：

   對於3D 空間點雲來說，具有無序性、點集合具有局部特徵結構和仿射

變換無關性三大特徵。而基於此三大特性，PointNet 的中心思想即是學習每個點的高維度特徵，之後將所以特徵聚合為一個全域特徵。PointNet 的設計主要基於三點構成：

(1) 無序輸入的對稱函數（Symmetry Function for Unordered Input）

使用一個簡單的對稱函數去收集每個點的訊息，點雲中的每個點使用 h 函數處理，再送入 g 函數中，以實現排列的不變性，而於此，如下，h 函數即為多層感知機（Multilayer Perceptron, MLP）、g 函數即為最大池化（ max pooling）。

$$f(\{x_1, \dots, x_n\}) \approx g\big(h(x_1), \dots, h(x_n)\big)$$

(2) 局部和全域資訊整合（Local and Global Information Aggregation）

通過增加每個點的局部和全域特徵的串聯，以此得到局部和全域資訊的點特徵（point-wise feature）。

(3) 對齊網路（Alignment Network）

由於點雲對仿射變換（Affine transformation）等變換具有無關性，故直接將所有點對其到一個統一個空間，以此保證網路可以學習到無關性。

雖然 PointNet 是一個簡潔而強大的網路，仍有以下缺點：

(1) 只對點的多層感知機：

僅針對各個點表徵，對局部結構的資訊整合仍稍嫌不足。

(2) 全域特徵損失：

全域特徵只由最大池化（maxpooling）獲取，且語義分割中的全域特徵是複製與局部特徵串聯，故而能形成有區別性的特徵可能性較低。

所以 PointNet 在局部處理和泛化至較為複雜或是龐大的場景上能力較有限。

3. PointNet++：

PointNet++即是基於 PointNet 的改進版。尤其針對 PointNet 的缺點—全域特徵不夠具有區別性--針對提取更有表現力的全域特徵。PointNet 在語義分割上使用編碼器-解碼器（Encoder-Decoder）的結構，先做降採樣再做上採樣，最後使用反向插值（Interpolation）和殘差連接（Skip

Connection）將對應層的局部和全域特徵進行拼接：

(1) 編碼器（Encoder）：

在原先 PointNet 的基礎上，增加了階層特徵學習框架（hierarchical feature learning framework），由採樣（Sampling）、分組（Grouping） 和 PointNet 組成一個集合抽象化 （Set Abstraction）：

a. 採樣（Sampling）：

使用最遠點採樣（FPS, farthest point sampling）進行降採樣，在點集空間中得到比較均勻的採樣結果。

b. 分組（Grouping ）：

以採樣結果為中心，找出固定規模的鄰居，組成局部鄰域（patch）。

c. PointNet：

使用 PointNet 對以上結果進行表徵。

（2） 解碼器（Decoder）：

自編碼器中得到的是全域特徵，將全域特徵和原本的局部特徵拼接，使得新的點特徵能得到鄰域資訊，通過反向插植和殘差連接（Skip connection）來得到新的的點特徵。

然而，當採樣和分組在稀疏地方操作時，一個點可能代表了很多局部特徵，因此一旦缺失，網路性能就會極大的受影響。

4. KPConv：

KPConv 基於圖像卷積（Image convolution）的想法，以點核（kernel point）取代核心像素（kernel pixel） 來定義每個核心權重（kernel weight）的應用區域。每個點上有核心權重，並且由相關函數（correlation function）定義，點核的數量並不受約束，且每個卷積位置能夠形成位移，所以更為靈活、且可以針對點雲的不同區域調整點核的形狀。

對 SensatUrban 這個資料集，工作坊有提供幾種有代表性的方法，以原始的實驗設置和參數做基準，分別是基於體素方法的 SparseConv[5]、基於2D 投影方法的 TagentConv[3]、基於點結構的 PointNet[4]、PointNet++[5]、SPGraph、KPConv[7] 、RandLA-Net [6]，而實驗數據如下：

Method	OA(%)	mAcc(%)	mIOU(%)	Ground	Vegetation	Buildings	Walls	Bridge	Parking	Rail	Traffic Roads	Street Furniture	Cars	Footpath	Bikes	Water
PointNet[4]	80.78	30.32	**23.71**	67.96	89.52	80.05	0.00	0.00	3.95	0.00	31.55	0.00	35.14	0.00	0.00	0.00
PointNet++ [5]	84.30	39.97	**32.92**	72.46	94.24	84.77	2.72	2.09	25.79	0.00	31.54	11.42	38.84	7.12	0.00	56.93
TagentConv [3]	67.97	43.71	**33.30**	71.54	91.38	75.90	35.22	0.007	45.34	0.00	26.69	19.24	67.58	0.01	0.00	0.00
SPGraph	85.27	44.39	**37.29**	69.93	94.55	88.87	32.83	12.58	15.77	15.48	30.63	22.96	56.42	0.54	0.00	44.24
SparseConv [2]	88.66	63.28	**42.66**	74.10	97.90	94.20	63.30	7.50	24.20	0.00	30.10	34.00	74.40	0.00	0.00	54.80
KPConv [7]	93.20	63.76	**57.58**	87.10	98.91	95.33	74.40	28.69	41.38	0.00	55.99	54.43	85.67	40.39	0.00	86.30
RandLA-Net [6]	89.78	69.64	**52.69**	80.11	98.07	91.58	48.88	40.75	51.62	0.00	56.67	33.23	80.14	32.63	0.00	71.31

表一、實驗數據表

我們可以觀察到 KPConv[6] 有最高的 mIoU 表現，然而幾乎所有的模型針對自行車 （Bikes） 和鐵軌（Rail）的分割效果都不甚理想。主因即為此資料集中，地表（Ground）、建築物（Building）和植被（Vegetation）三類就佔據了整個資料集中的50%以上；自行車（Bikes）和鐵軌（Rail）僅僅只佔據了0.025%左右，故而考慮到不同類別的分布不均也是我們重要的挑戰之一。

三、設計原理

我們主要以目前 Paper with Code 上面針對各模型在 SensatUrban 資料集上的分析排名作參考，並以工作坊所提供基於點的訓練模型—RandLA-Net 與基於投影（Projection）的訓練模型—BEV-Seg-Net 作為修改基礎及比較對象：

1. RandLA-Net：

由於 SensatUrban 為大規模的點雲，在考慮語義分割時首先考慮到的便是降採樣（Down Sampling）的部分，因現有設備考量，大多數的降採樣方式都是先將整個點雲集切成小點雲塊，然而許多物體便會在這切塊的過程中失去其原先的幾何結構，故而在點雲塊較大的情況下，神經網路會很難學習到物體的整體幾何結構。

而 RandLA-Net 便是基於此提出的網路結構，具有計算效率高（Computationally-efficient）和內存占用少（Memory-efficient）的兩大特性，並且在大規模點雲有很高的應用性。RandLA-Net 重要結構有二 ： 隨機採樣（Random Sampling）和局部特徵聚合模塊（Local Feature Aggregation）：

(1) 隨機採樣：

使用隨機採樣，使用 K*O（1）的時間從輸入的 N 個點中選擇 K 個點，故計算量只和我們所需採樣後的點數量相關，故而有相當高的效率。

然而，使用隨機採樣的不確定性必定會導致局部訊息的缺失，針對這個問題，作者提出局部特徵聚合模塊作為解決辦法。

(2) 局部特徵聚合模塊：

其中包含三個部分：局部空間編碼（Local Spatial Encoding）、自注意池化層（Attentive Pooling）和擴張殘差塊（Dilated Residual Block）。

a. 局部空間編碼（LocSE）：

針對點雲的座標進行編碼，使用 K 近鄰算法 （K-nearest neighbor algorithm），找出最近的 K 個點，之後將中心點座標、K 個鄰點座標、相對座標和距離串聯（concatenation）、再和鄰點對應到的點特徵串聯在一起，成為一個新的 feature。

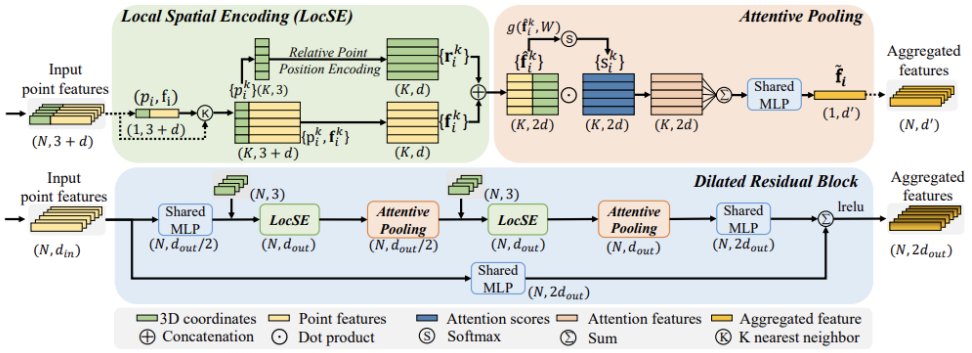
b. 自注意池化層（Attentive Pooling）

設計共享函數為每個點學習一個自己的 attention score，並將

attention score 視為一個能選擇重要特徵的 soft mask，而最終學到的特徵是這些鄰域特徵的加權總合。

c. 擴張殘差塊（Dilated Residual Block）

因為大幅度的降採樣，故擴張殘差塊的目的是為了增加每個點的感受野（Receptive field）。將多個局部空間編碼、自注意池化層和殘差連接（Skip connection）組合成一個擴張殘差塊（如下圖）。



圖一、局部特徵模塊流程圖

2. BEV-Seg-Net

BEV-Seg-Net 為基於2D 投影的方法，先將3D 點雲投影至2D 的鳥瞰圖，以2D 形式做語義分割，再重新將分類映射回3D 的點雲。雖然2D 投影方法容易丟失幾何資訊，然而基於計算能力和儲存空間的考量，2D 投影會較一般基於3D 的體素和點方法都更具效率。

BEV-Seg-Net 的主要結構有二：鳥瞰圖投影（Bird's-eye-view Mapping）、2D 多模態分割（2D multi-modal segmentation）：

（1）鳥瞰圖投影（BEV Mapping）：

為了處理 SensatUrban 的龐大資料量，將鳥瞰圖投影分為三個步驟：3D 至鳥瞰圖投影（3D-to-BEV projection），稀疏鳥瞰圖補全（sparse BEV images completion），和鳥瞰圖至3D 的重新映射（BEV-to-3D remapping）。

a. 3D 至鳥瞰圖投影（3D-to-BEV projection）

首先先設這一個滑動視窗（Sliding Window），在每一步中，對滑動視窗內的點進行映射，再將處理後的點刪除以減少資料處理量。

b. 稀疏鳥瞰圖補全（ Sparse BEV images completion ）

透過最大池化（maxpooling ）對圖片做補全。

c. 鳥瞰圖至3D 的重新映射（ BEV-to-3D remapping ）

儲存每個滑動視窗的絕對座標，並使用主題查詢為2D 語義分割提取原始點雲的位置輸出。

（2） 2D 多模態分割 （2D multi-modal segmentation ）

利用編碼器解碼器網路（Encoder-Decoder Network ） Unet 作為基準，它由4塊編碼器和5塊解碼器組成，而其中有兩塊為 ResNet-34。每一層卷積塊都有批量標準化層 （ Batch-Normalization Layer ）和整流線性單位函數層（ ReLU Layer）。

多模態網路主要依賴不同層的特徵通訊（ Feature Communication ），此處使用一個靈活的多階段融合網路，得以實現多管道數據融合，採用注意力層選擇較為重要的通道。最後，添加一個1x1卷積來降低維數，保持輸出的恆定形狀。

四、研究方法及步驟

此次專題研究一共分成以下幾個階段：

1. 準備階段

在準備階段，透過博士班學姊給我們的3D 語義分割簡介與教學讓我們對於3D 語義分割有著比較深的了解，我們再另外以 Paper with Code 上 SensatUrban 資料集的排名作為參考，將可以運行在 SensatUrban 資料集的模型與其發表的 Paper 都研讀，嘗試在各個不同的模型間找出表現好並值得參考的功能。

2. 實作階段

在實作階段，我們先著手研究的模型為 RandLA-net，並且我們將 RandLA-net 的研究與輸入切割以下三個重點部分，並且一步一步進行修改與研究：

（1） 資料集預處理

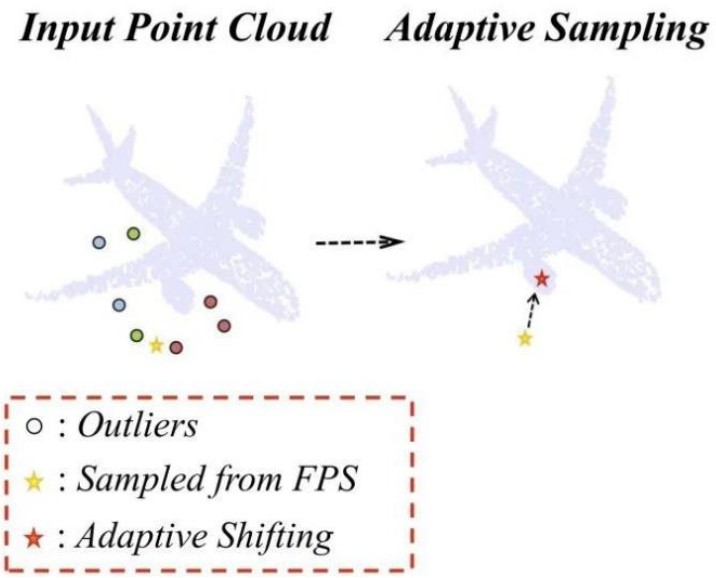
由於 SensatUrban 資料集中的物件分配得極度不平均，可能導致機器對於少數物件的訓練不夠而無法成功分辨，於是我們先對資料集進行資料增強（Data Augmentation [8]），想藉由資料的變化來增加資料的多樣性、隨機性以及總數，進而提升模型的表現。

（2） 採樣方式

由於3D Urban Challenge 工作坊所著的論文中有提及，在他們研究的過程中發現，資料的降採樣與分割方式對於訓練成果有相當大的影響。在他們的測試中，以簡單的方式結合網格採樣（Grid sampling）以及將輸入的資料分割成同等點數的區塊（block）的方式就能提升約10%的 mIoU 分數，所以我們決定在這採樣這部分進行研究及修改，我們嘗試的採樣方式如下：

a. 自適應採樣（Adaptive Sampling [9] ）

在學習點雲的特徵（feature）時，會使用 encoder 與 decoder 的網路結構。而在 encode 的過程中一般使用最遠點採樣（FPS, farthest point sampling [10] ）。然而資料中時常會帶有一些雜訊。下圖中，由於雜訊的干擾，導致最遠點採樣所採到的點為黃色星號的位置，很明顯的這個點並不是我們所要的，如果採用此點可能會造成訓練上的負面影響。而我們參考 PointASNL： Robust Point Clouds Processing using Nonlocal Neural Networks with Adaptive Sampling 這篇論文中的 Adaptive sampling 的想法，試圖降低雜訊對於訓練的影響。



圖二、FPS 與自適應採樣模擬圖

b. 隨機採樣 （Random Sampling）

為 RandLA-Net 所預設使用的降採樣方式，其方法從原始的 N 個點中均勻地選擇 K 個點。它的計算複雜度為 O（1），與輸入點的總數無關，即它是恆定時間的，因此具有固有的可擴展性。與 FPS 相比，隨機採樣具有最高的計算效率，但同時會失去資料的局部特徵，因此需要額外的技術來輔助，才能在不失特徵性下進行訓練。無論輸入點雲的規模如何。 處理10^6個點僅需0.004秒。

c. 最遠點採樣 （Farthest Point Sampling）

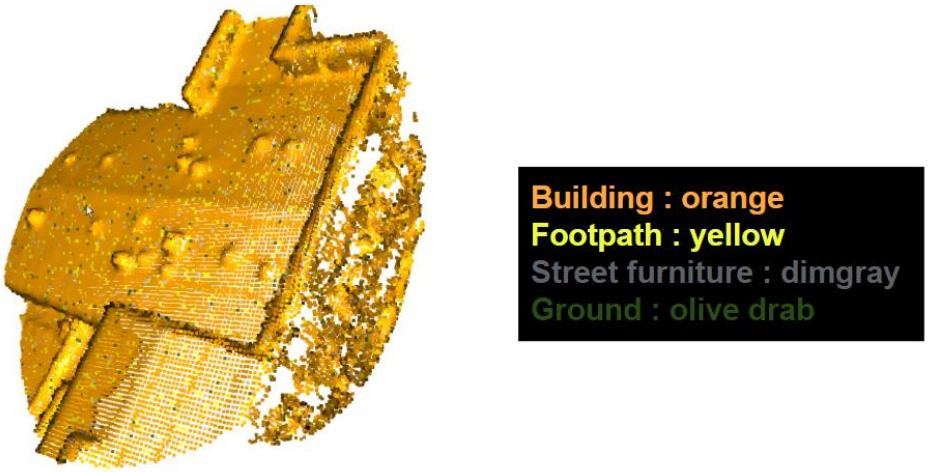
最遠點採樣是常見的採樣方式之一，因為此採樣法能夠確保所有的採樣點均勻地分布在資料中，不會有部分區域沒有被採樣到而影響訓練效果的情況，所以被廣泛的使用。

其方法是先從輸入的點雲中的 N 個點取一個點作為 P0，將其更新至採樣點集合，計算出所有點到 P0的距離，再挑選離 P0最遠的點作為 P1，將其更新至採樣點集合，並計算出所有點到 P1的距離，重複上上述動作直到有 K 個採樣點為止。因為點雲中有 N 個點，要對其採樣出 K 個點，時間複雜度為 O（KN）。雖時間複雜度遠不如隨機採樣，但其保留的點與點之間的相對特徵，有助於之後的訓練效果。

而在 SensatUrban 資料集中，由於點雲已經預設被切割成33個不同的 ply 檔，且由於硬體空間不足導致無法將所有檔案一次讀取至記憶體，讓整個 SensatUrban 資料集拼湊成完整的點雲，所以在 FPS 取樣中無法直接套用此取樣法，所以在後續系統實現部份我們有針對 FPS 再針對讀取多個檔案進行了不同的修改以符合我們的需求。

（3） 模型與點雲可視化[11][12]

點雲可視化可以讓我們簡易的查看驗證集（Validation set）與測試集（Test set）在我們預測的表現下與正解的差別，從而觀察出模型在哪些部份的與預測需要加強，有助於我們了解對模型進行改良的方向。



圖三、可視化圖（一）

圖四、可視化圖（二）

點雲可視化的結果如上面兩張圖所示。

圖三的資料中只含有一棟建築物，而模型在也成功將大部分的點標記為建築物（橘），而我們也可以明顯看出其中有極少部分的點被判斷為其他

物件，例如人行道（黃）、道路設施（灰）、地面（橄欖綠），這部分我們認為是可以忽略的誤判率。

圖四的資料中可以大概的看出左側為樹，右側為建築物，下方為地面，而模型卻幾乎將其判斷為樹（綠）、道路（橄欖綠）以及少部分的人行道（黃）、道路設施（灰）。這筆資料的預測大部分皆為錯誤的，而我們分析造成這種情況的可能原因為一筆資料的點數過少，由於設備資源有限，我們所訓練的環境並不允許我們使用官方所提供的模型參數，我們將一筆資料的點數從 65536 降低到 30000。在模型訓練時，模型能在一筆資料中只能看到30000個點，視野受到極大的限制，又因為 SensatUrban 這個資料集的分類物件大小差異巨大，最大到一整棟建築物，最小到一輛自行車，所以一筆資料通常只能訓練到一種物件，而當一筆資料中有多個不同的物件時，模型就會難以分辨。而我們想到的解決方法有二：更換 VRAM 較大的 GPU、將 Batch size 降低轉而增加一筆資料的點數。

而在 BEV-Seg-Net 的部分，我們主要針對採集的鳥瞰圖進行處理。

（1） 輸入圖像大小

由於 GPU 的硬體限制，我們難以使用論文中原始設定的尺寸（500x500）來做輸入，故而我們嘗試了三種大小，分別是100x100、200x200和300x300。而不同尺寸的輸入圖片會對每次訓練的 Batch Size 和計算時間有很大的影響。

（2） 輸入圖像僅使用 R、G、B 三個通道

因為鳥瞰圖導致原本3D 空間的物體缺少了高低差，於是我們增加了第四個通道，即為高度（Altitude）來做訓練。

五、系統實現與實驗成果

在 RandLA-net 方面，我們使用的硬體設備為：

CPU： Intel Core i7-6700

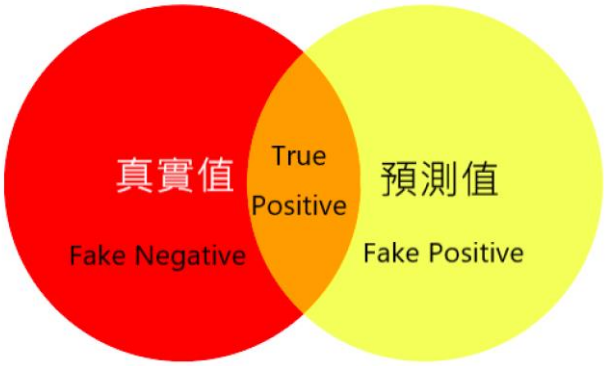
RAM： 64GB

GPU： GeForce GTX 1080（VRAM 8GB）

而我們使用的評估指標為 Mean Intersection over Union （mIoU），公式為：

$$mIOU = \frac{1}{k+1}\sum_{i=0}^{k}\left(\frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - p_{ii}}\right)$$

其中p_{ij}的i表示為真實值，j為預測值，p_{ij}即表示將 class i預測為 class j



mIoU 也可以以此圖表來表示：

圖五、mIoU 示意圖

則 mIoU 即為計算兩圓之重合比例，公式為：

$$mIOU = \frac{True\ Positive}{Fake\ Negative + Fake\ Positive + True\ Positive}$$

我們分成三個變因來介紹系統實現方式，並且一次只更改一個變因以進行比對與效能評估：

RandLA-net 初始設定為：

無進行資料增強

Batch Size = 6, Data Size = 30000

隨機取樣法

我們嘗試更改的三個變因：

1. 資料增強[13]

由於 RandLA-Net 是以點雲的形式作為輸入，所以我們的想法是使用轉換矩陣（Transform Matrix）對點雲中各點的三軸座標（Coordinate）進行調整：

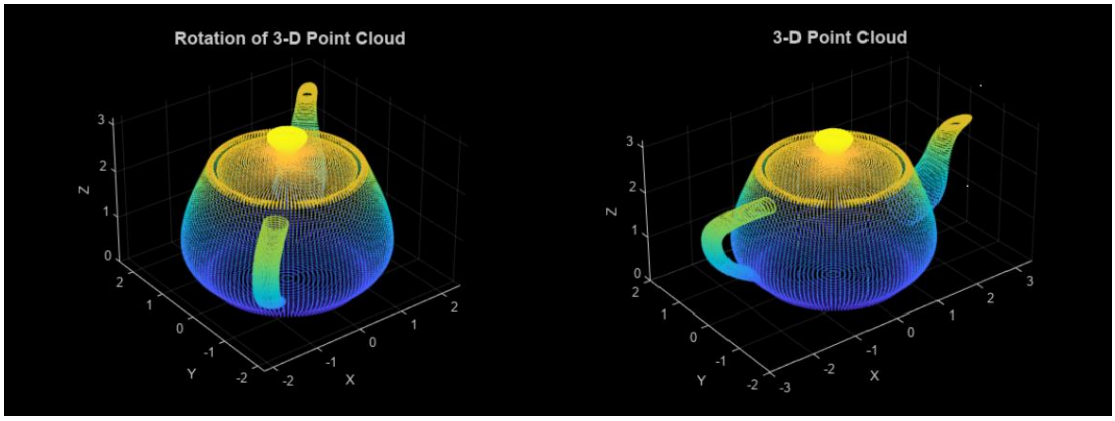
（1）隨機旋轉 （Random Rotate）

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & \\ 0 & 0 & 1 \end{bmatrix}$$

圖六



圖七

透過針對三軸的旋轉矩陣圖六與資料的每點的座標相乘，得到新的座標，相當於對整個資料進行旋轉，並以隨機的方式進行旋轉增加資料的多樣性

（2） 隨機平移（Random Translate）

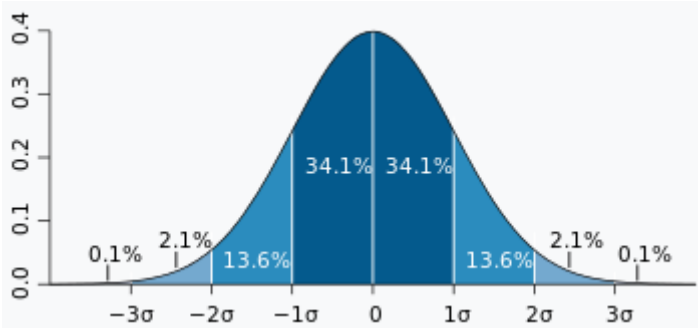
對一筆資料的所有點的座標加上一個3 X 3矩陣，讓資料整個平移一段距離，矩陣同樣以隨機的方式來增加資料的多樣性。

（3） 隨機放大（Random Scale）

將一筆資料的所有點的座標與3 X 3矩陣進行相乘，得到一個與原資料有著一定比例縮放的新資料，因為是對整體資料的縮放，所以點與點之間的距離及點雲整體密度也會受到影響，縮放的參數同樣以隨機的方式來增加資料的多樣性。

（4） 增加高斯雜訊（Gaussian Noise）

高斯雜訊所產生的雜訊值是隨機的，但將雜訊值加以統計後會呈現高斯分布（圖八），也就是常態分布。當我們觀察雜訊譜，會發現高斯雜訊與原圖較為融合（圖十），不會產生過於極端的雜訊而破壞資料的完整性，且這類雜訊還可以透過調整平均值、標準差等參數，來產生不同的雜訊。



圖八、高斯分布圖

圖九、原始圖像

圖十、增加高斯雜訊圖像

資料增強成果

Method	**mIoU**
Random Rotate	41.690
Fixed Rotate	42.758
Random Translate	42.727
Random Scale	43.351
Fixed Scale	41.237
Add Gaussian Noise	43.811
Use built-in tf_augment	42.758
Baseline	48.223

表二、資料增強數據圖

分別對於輸入資料進行隨機與定值的旋轉、平移、縮放、高斯雜訊以及使用 Tensorflow 內建的資料增強函式，且皆在同一個環境下進行測試，結果如表二所示，在沒有施加任何資料增強下的模型 mIoU 為48.223。而有資料增強的模型其表現雖然有著明顯的差異，但全部的表現皆比 Baseline 還要差，平均 mIoU 大約為 42，與預期的成果相差甚遠。由於測試出來的結果顯示出資料增強並沒有效益，所以我們後來捨棄了針對這部分的修改。

2. 批次大小（Batch Size）與資料大小（Data Size）的比例

由於在 GTX 1080的訓練嚴重受限於 VRAM 大小，所以我們並沒有在此顯示卡上面進行批次大小與資料大小的比例調整，在後續實驗室有提供給我們第二張顯示卡（GTX Titan X）的訓練資源，我們才有進行比例大小的調整，此部分會於後續篇幅提及成果。

3. 取樣方式

RandLA-net 預設的取樣方式為隨機取樣一個點後，找出此點鄰近的30000個點當成 Network 的輸入，而我們遵循他的取樣想法，但將取樣方式更改為以下兩種：

（1）. 最遠點取樣 （FPS, Farthest Point Sampling）

在 SensatUrban 資料集中，由於點雲已經預設被切割成33個不同的 ply 檔，且由於硬體空間不足導致無法將所有檔案一次讀取至記憶體，讓整個 SensatUrban 資料集拼湊成完整的點雲以進行取樣，所以針對最遠點取樣法，我們再次進行降採樣，並對讀取多個檔案並進行一層3000筆的資料取樣進行了三種不同的修改：

a. 每層的3000筆資料都來自同一個隨機選擇的點雲

優點：只需將 FPS 套用至其中一個點雲

缺點：資料多樣性差

b. 每層的3000筆資料平均的來自33個點雲

優點：資料多樣性較好，取樣比方法 a 來的平均

缺點：並非真正對於整個 SensatUrban 資料集的平均

c. 每層的3000筆資料依照33個點雲大小比例於各自點雲進行取樣

優點：資料對整個 SensatUrban 資料集來說非常平均

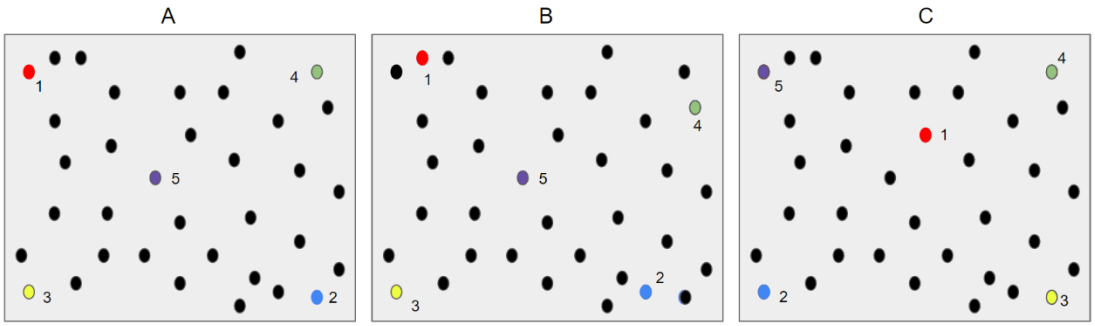
缺點：需要較多時間以進行最遠點取樣算法

Method	**mIoU in 80 epochs**
FPS Method 1	30.648
FPS Method 2	38.183
FPS Method 3	40.743
Baseline	47.267

表三、FPS 數據圖

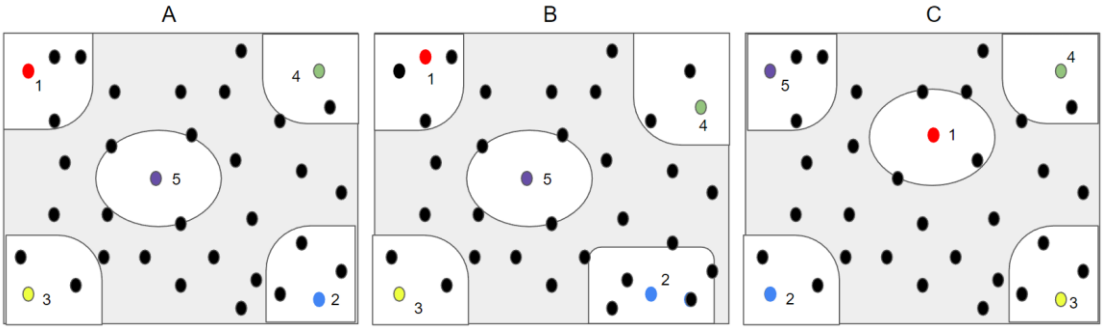
在這邊我們產生了一個疑問：最遠點取樣明明比隨機取樣更加平均，為何表現卻如此不佳？

而在此問題上，我們所得出的推測是：雖然 FPS 有著更平均的取樣結果，但是在不同 epoch 的覆蓋範圍都過於相似，以下舉一個2D 點雲為例說明：



圖十一、FPS 模擬圖（一）

上圖三個 A、B、C 為相同的點雲，取樣順序為第一個隨機取的點為紅色，接下來依照最遠點取樣的順序藍色→黃色→綠色→紫色，A 點雲與 B 點雲取樣的第一個點較相似，而 C 點雲取樣的第一個點為中心。

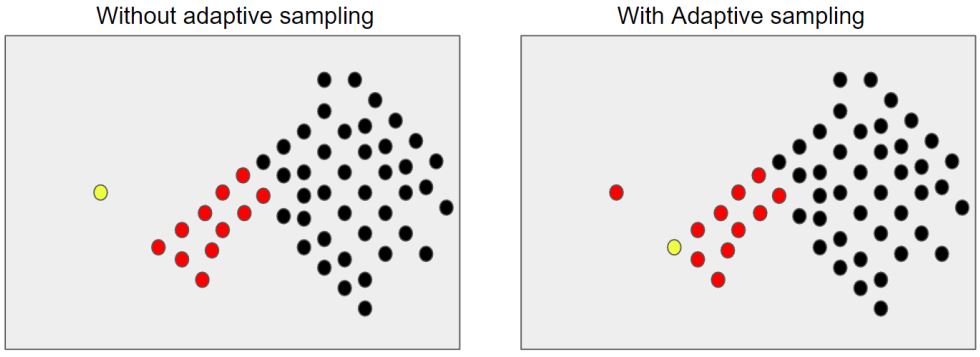


圖十二、FPS 模擬圖（二）

由上圖可見，在此點雲三個進行完全不同的三次最遠點取樣後，最後找出離取樣點最近的30000個點後，取到的資料（白色底部分）卻非常相似，皆為四個角落與中心部分。這就使得了不同 epoch 的覆蓋範圍都過於相似，而在大點雲的取樣數過少的情況下，灰色底的部分是永遠都無法取到的，也造成了有死角的產生。相較於隨機取樣，雖然平均性較差，但隨機取樣更有可能取到最遠點取樣無法取到的灰色部分，所以隨機取樣反而在資源有限的情況下可以得到更好的表現。

（2）. 自適應採樣（Adaptive Sampling）

自適應採樣在實現的過程中，也是使用其中一種採樣作為基準（原論文使用最遠點採樣），再另外處理使我們取到的離群值（Outlier）可以貼近



正常的點雲輪廓，而自適應採樣在經過評估之後並沒有實作上模型。

圖十三、有無自適應採樣之比較圖（三）

我們以實際情況舉例，假設我們一筆資料為12個點，RandLA-net 的取資料流程為：找到點後取其附近的11個點做為一筆資料。

如上圖左，當我們取到離群值（黃色點）時，離群值通常為雜訊造成，會接近點雲的輪廓，所以取到的點即為離離群值最近的11個點，一般而言會是點雲輪廓的其中一個角落，即為黃色與紅色的11個點。

而上圖右的情形顯示我們使用自適應採樣時，我們取到的中心點將會是最接近離群值的點雲輪廓點，即為上圖右的位於點雲輪廓最角落之黃色點，再找尋與其最近的11個紅色點即是此次取樣取出的資料。

以兩張圖相比，雖然我們中心點取到的點位置差異甚大，但是最後取整筆資料時卻是相同的兩份資料，可見自適應採樣對於 RandLA-net 的取資料方式並不適用，在絕大多數情況下將會花費資源運算卻得到相同資料

於 RandLA-net 方面的訓練，由於在前面使用 GTX1080的過程中發現使用 8GB VRAM 計算出的結果都差強人意，於是我們與實驗室借用了另一台電腦進行訓練，而此電腦的配備為：

CPU： Intel Core i7-6700

RAM： 64GB

GPU： GeForce GTX Titan X（VRAM 12GB）

在 VRAM 提升到12GB 的條件下，我們可以將一次輸入模型的點從180000提升至240000，於是我們將輸入的點提升為6*40000（與8G VRAM 相比提升 Data size），而也嘗試了非常接近 RandLA-Net 作者提出的初始設定4*65536的 4*60000，而使用一樣的 Anaconda 虛擬環境進行訓練，主要更改的點為批次大小（Batch Size）與資料大小（Data Size）的比例，結果如下：

Batch size / Data size	**mIoU**
6*30000 （on GTX1080）	48.223
6*40000 （on GTX Titan X）	54.087
4*60000 （close to Default Setting）	50.024
24*10000	55.598
48*5000	54.865

表四、批次與資料量之數據圖

可以看到在提升 VRAM 大小之後，整體模型的表現有著非常大的提升，可見 RandLA-Net 對於輸入的資料大小非常敏感，在進行了不同的比例測試後，我們發現24*10000的比例大概會是此模型輸入大小的甜蜜點，再提升批次大小或是資料大小都不會獲得更好的表現。

於 RandLA-Net 部分，我們也對訓練速度進行了統計，使用 RandLA-net 預設的設定跑，於 CPU 上跑雖然可以使用64GB 的記憶體，但是需要花費非常大量的時間，而若是於 GTX1080上面跑一個 Epoch 大約需要20-25分鐘，可見目前機器學習部分還是需要 GPU 的運算才較為快速，我們所測試的數據如下：

Factor	**Time per Epoch**
Default on CPU	200+min
Default on GPU	12-16min
Data Augmentation	12-16min
FPS on the whole SensatUrban	60-65min
FPS on the 1/10 downsampling SensatUrban	25-30min
Modify the batch size and data size	12-16min

表五、訓練時間數據圖

在 BEV-Seg-Net 方面，我們使用的硬體設備為：

CPU： Intel Core i7-6700

RAM： 64GB

GPU： GeForce GTX TITAN X（VRAM 12GB）和 GeForce GTX 1080（VRAM 8GB）

而我們使用的評估指標亦為 Mean Intersection over Union （mIoU）。

我們分成兩個變因來介紹系統實現方式，以進行比對與效能評估：

BEV-Seg-Net 初始設定為：

模型輸入鳥瞰圖大小為500x500

原始輸入圖僅有 R、G、B 三通道

我們嘗試更改的兩個變因：

1. 改變輸入圖片大小:

由於 GPU 的硬體限制，我們以初始的圖片大小進行操作的話，batch size 僅只能開到1，實驗的效果太差，於是我們對初始的鳥瞰圖進行重新縮放，嘗試了100x100使用的 RGB 作為我們的 baseline，mIoU 為16.32。

Method	OA(%)	mAcc(%)	mIOU(%)	Ground	Vegetation	Buildings	Walls	Bridge	Parking	Rail	Traffic Roads	Street Furniture	Cars	Footpath	Bikes	Water
BEV-rgb 100x100 epoch 300	54.87	24.40	16.32	47.26	61.39	37.72	0.00	0.00	0.00	0.00	23.22	0.00	0.00	0.00	0.00	0.00

表六、BEV 數據圖（一）

在圖片大小為100x100的情況下，在 GeForce GTX 1080（VRAM 8GB）中，batch size 可調整為32。

2. 增加高度（altitude）為第四通道:

由於原本只使用 RGB 資訊的成果不佳，故而我們想在鳥瞰圖情況下，最需要的就是每個張圖片對應點的高度資訊，所以使用高度作為其第四通道，而得到以下結果:

method	**mIoU**
BEV-rgbd 100x100 Epoch300 （on GTX 1080）	64.93
BEV-rgbd 100x100 Epoch 200 （on GTX 1080）	57.61
BEV-rgbd 200x200 Epoch 200 （on GTX TitanX）	46.93
BEV-rgbd 300x300 Epoch 80 （on GTX TitanX）	42.07

表七、BEV 數據圖（二）

圖片大小為100x100的是在 GeForce GTX 1080（VRAM 8GB）上進行訓練，且每個 Batch Size 均為32；圖片大小為200x200是在 GeForce GTX TITAN X（VRAM 12GB）上進行訓練，其 Batch Size 為16；圖片大小為300x300亦是在 GeForce GTX TITAN X（VRAM 12GB）上進行訓練，其 Batch Size 僅能開到 8。

由上述數據首先可以觀察到的即是同樣圖片尺寸和 epoch 數，卻因有無使用高度通道而產生的巨大差異，沒有使用的 mIoU 值僅有16.32；而使用的卻能達到57.61，可以看到在使用鳥瞰圖投影做語義分割時，有無加入物體高度的差異非常之大。

而在同樣使用四個通道的數據看來，可以發現 Batch Size 的大小影響非常大，然而，較大的圖片（300x300）在小類別的資料（Rail）上，表現較優秀，我們推測是因為在做圖片縮放時，可能導致小類別的物件如鐵軌（Rail）和腳踏車（Bikes）過於模糊以致無法辨別。

我們也對於每次實驗的時間進行統計，每個 Epoch 所花費的時間主要和輸入圖片尺寸相關。

Factor	**Time per Epoch**
Image Size 100x100	30-35 min
Image Size 200x200	70-80 min
Image Size 300x300	240+ min
Image Size 500x500	900+ min

表八、圖片尺寸與時間之數據圖

我們實驗結果統計如下:

Method	mIoU	Groun-d	Veg.	Buildings	Walls	Bridge	Parking	Rail	Traffic oads	Street	Cars	Footpath	Bikes	Water
RandLA-Net Default	48.20	72.23	95.23	93.41	34.08	8.48	42.91	11.16	53.26	28.33	74.43	16.72	0.00	47.50
RandLA-Net Epoch200	49.24	74.29	95.23	94.09	35.84	14.21	47.86	12.29	53.87	28.65	75.33	16.72	0.00	58.52
RandLA-net w/ Titan X	50.02	72.12	96.41	94.00	47.06	16.14	49.55	7.62	59.06	33.29	79.31	20.41	3.21	63.69
RandLA-net 6*40000	54.09	74.12	96.37	**94.44**	51.92	48.08	47.73	14.44	60.60	39.16	82.63	19.49	**9.39**	64.77
RandLA-net 24*10000	55.6	73.74	**96.66**	94.41	**52.69**	59.04	55.32	20.89	61.37	33.34	**83.01**	17.99	0.00	74.30
Hand Augmentation	40.88	67.20	94.87	90.25	34.12	4.15	32.93	0.72	54.95	23.57	72.67	17.08	0.00	38.96
tf_augmentation	41.25	66.60	95.22	90.83	32.34	4.34	38.35	0.03	54.80	23.79	70.48	15.83	0.00	43.70
FPS – method1	30.16	69.19	94.92	84.56	26.83	0.04	1.06	0.00	45.02	13.05	57.41	0.00	0.00	0.00
FPS – method2	38.70	67.73	93.53	87.77	29.77	2.34	23.97	0.00	38.72	26.22	73.99	12.11	0.00	46.95
FPS – method3	40.74	69.79	93.51	87.72	31.83	7.42	33.24	0.00	46.12	25.15	72.26	15.93	0.00	46.70
BEV-RGB 100x100 Epoch300	16.32	47.26	61.39	37.72	0.00	0.00	0.00	0.00	23.22	0.00	0.00	0.00	0.00	0.00
BEV-RGBD 100x100 Epoch300	**64.93**	**88.78**	90.44	93.96	38.25	**82.43**	**77.78**	0.00	**85.42**	**40.48**	70.88	**69.01**	0.00	**76.12**
BEV-RGBD 100x100 Epoch200	57.61	79.47	79.79	85.69	24.47	74.52	69.02	0.00	78.50	36.83	69.47	58.84	0.00	66.55
BEV-RGBD 200x200 Epoch200	46.93	68.22	88.48	90.72	33.86	7.41	40.19	0.00	58.69	28.93	68.12	20.45	0.00	65.60
BEV-RGBD 300x300 Epoch80	42.07	48.50	66.43	68.38	24.42	40.04	27.45	**22.29**	51.19	40.00	58.31	28.01	0.00	31.80

而原本實驗的 Baseline 結果如下:

Method	mIoU	Groun-d	Veg.	Buildings	Walls	Bridge	Parking	Rail	Traffic oads	Street	Cars	Footpath	Bikes	Water
PointNet	23.71	67.96	89.52	80.05	0.00	0.00	3.95	0.00	31.55	0.00	35.14	0.00	0.00	0.00
Point-Net++	32.92	72.46	94.24	84.77	2.72	2.09	25.79	0.00	31.54	11.42	38.84	7.12	0.00	56.93
TagentConv	33.30	71.54	91.38	75.90	35.22	0.007	45.34	0.00	26.69	19.24	67.58	0.01	0.00	0.00
SPGraph	37.29	69.93	94.55	88.87	32.83	12.58	15.77	**15.48**	30.63	22.96	56.42	0.54	0.00	44.24
SparseConv	42.66	74.10	97.90	94.20	63.30	7.50	24.20	0.00	30.10	34.00	74.40	0.00	0.00	54.80
KPConv	**57.58**	**87.10**	**98.91**	**95.33**	**74.40**	28.69	41.38	0.00	55.99	**54.43**	**85.67**	**40.39**	0.00	**86.30**
RandLA-Net	52.69	80.11	98.07	91.58	48.88	**40.75**	**51.62**	0.00	**56.67**	33.23	80.14	32.63	0.00	71.31

六、未來目標

目前我們於 SensatUrban 資料集已經達到60%以上的 mIoU，但以 Urban 3D challenge 的排行榜來看，目前於此挑戰能夠達成的最高表現約為74.5%的 mIoU，我們將以此為目標，希望於有限的計算資源上可以達到高水準的表現，並且於一些佔比過小的 class，如 Bike, Rail 等分類可以在模型上進行一些加強與調整，或是調整訓練集各個分類之比例，使這些分類不再只有1-20%的表現 ，而是可以有著更高的 mIoU。

七、團隊合作方式

於前期文獻閱讀部分，我們組內每週都會各自找尋不同有關3D 語義分割的論文，並且以讀書會的方式提出我們找到的不同模型值得嘗試的優點並記錄。

中期實作部分，我們每週都會與學長開會，進行進度回報與問題討論，並且我們三位同時於更改不同的檔案，進行同步工作以加速嘗試的進度。

八、結論

人工智慧發展日新月異，人們對於電腦視覺技術的要求也越來越高，語義分割作為電腦視覺的核心技術之一，其發展對於機器識別能力尤其重要。為了因應現代科技的要求，語義分割的識別範圍已經擴展到城市規模，然而現今沒有一個模型在這類超大型資料集上有穩定且良好的表現。為此，我們參考網路上的模型、演算法並進行研究，試圖在有限的資源下設計出能在超大型資料集上表現優異的分割模型。在經過無數次的研究以及訓練後，我們發現降採樣方式對於模型訓練影響極大，不管是訓練時長或訓練成果都與其密不可分，若能夠設計出時間複雜度低且能夠不失區域特徵的降採樣方式，必定能在這個領域上有所突破。儘管我們的模型成果並沒有超越現有的模型，但我們所分析出來的數據以及想法，能為其他研究作為參考，作為未來這方面領域的鋪成。

九、參考文獻

[1] https://urban3dchallenge.github.io/

[2] Ben Graham,（2015）. *Sparse 3D convolutional neural networks.*

[3] https://github.com/zouzhenhong98/SensatUrban-BEV-Seg3D

[4] Charles R.Qi & Hao Su & Kaichun Mo & Leonidas J. Guibas,（2016）. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation.*

[5] Charles R. Qi & Li Yi & Hao Su & Leonidas J. Guibas,（2017）. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space.*

[6] https://github.com/QingyongHu/RandLA-Net

[7] Hugues Thomas & Charles R. Qi & Jean-Emmanuel Deschaud & Beatriz Marcotegui &François Goulette &Leonidas J. Guibas,（2019）. *KPConv: Flexible and Deformable Convolution for Point Clouds.*

[8] https：//paperswithcode.com/task/3d-point-cloud-data-augmentation

[9] https：//blog.csdn.net/Yong_Qi2015/article/details/107625326

[10] https：//blog.csdn.net/dsoftware/article/details/107184116

[11] https：//towardsdatascience.com/guide-to-real-time-visualisation-of-massive-3d-point-clouds-in-python-ea6f00241ee0

[12] https://goteleport.com/blog/x11-forwarding/

[13]https://www.mathworks.com/help/vision/ref/pctransform.html