





# Machine Learning with *Kay*

Lasith Niroshan  and James D. Carswell 

Technological University Dublin, Ireland

Correspondence: Lasith Niroshan ([d19126805@mytudublin.ie](mailto:d19126805@mytudublin.ie))

**Abstract.** Computational power is very important when training Deep Learning (DL) models with large amounts of data (Wooldridge, 2021). Hence, High-Performance Computing (HPC) can be leveraged to reduce computational cost, and the Irish Centre for High-End Computing (ICHEC) provides significant infrastructure and services for research and development to both academia and industry. A portion of ICHEC's HPC system has been allocated for institutional access, and this paper presents a case study of how to use *Kay* (Ireland's national supercomputer) in the remote sensing domain. Specifically, this study uses clusters of *Kay* Graphics Processing Units (GPUs) for training DL models to extract buildings from satellite imagery using a large number of input data samples.

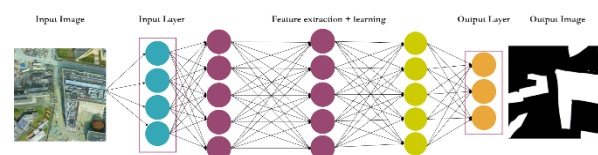
**Keywords.** *Kay* Supercomputer, Machine Learning, Deep Learning, Image Processing, GIScience

## 1 Introduction

*Machine Learning* (ML) is a subset of *Artificial Intelligence* (AI) – in effect, a computational approach to train an algorithm (mathematical model) to "learn" how to perform specific functions/tasks from large amounts of data. It learns using *Artificial Neural Networks* (ANN) which have proven successful at addressing complex problems in many domains, such as weather forecasting (Wang et al., 2019), drug discovery (Vamathevan et al. 2019), business intelligence (Chaturvedi et al., 2017), and computer vision (Siau et al., 2018), etc. *Deep Learning* (DL) is a subfield of ML consisting of multiple interconnected *hidden* layers of "neurons" that takes a group of weighted inputs, applies an activation function, and returns an output (Figure 1).

In ML, real-world processes are computationally represented by a model trained using *features* in the data. Features are the individual/independent image

characteristics (e.g. colour, tone, texture, pattern) of an object (e.g. tree, building, car, etc.) used to train the ML models (Bishop et al., 2006). Generally, learning algorithms require large amounts of data to train, learn and ultimately predict accurate outcomes. The key components of ML can be listed as follows; Input Data, Features, Training, and subsequent Model.



**Figure 1:** Typical DL network architecture to extract buildings from a given satellite image. By adjusting the weights on the connections between nodes, "learning" improves. Modern DL networks can contain dozens of hidden layers.

Raw computational power is another important ML component as a lack of sufficient computing power is a well-known constraint in AI-related research (Vedovello, 1998). Much scientific research is often limited by costly computing resources, while having access to High-Performance Computing (HPC) infrastructure can significantly impact research scope and results. Unlike typical academic research computers (e.g. desktop, laptop), HPCs take advantage of parallel processing to perform calculations much faster (Dennis, 1980). For AI research especially, access to an HPC resource is practically imperative to reduce the computational time/cost of complex model training using large datasets.

Accordingly, the Irish Centre for High-End Computing (ICHEC) provides HPC infrastructures and services for both the academic and industrial research sectors in Ireland. A number of institutions (including TU Dublin) are already using *Kay* (Ireland's national supercomputer) infrastructure in their research experiments.

This paper presents a practical GIScience example showing how to make actual use of Ireland's *Kay* supercomputing infrastructure in a research project. It

describes the complete end-to-end workflow from login to downloading ANN results using real-world spatial data input (raster satellite images and vector maps). Additionally, a number of run-time/processing issues encountered and solutions adopted are also discussed.

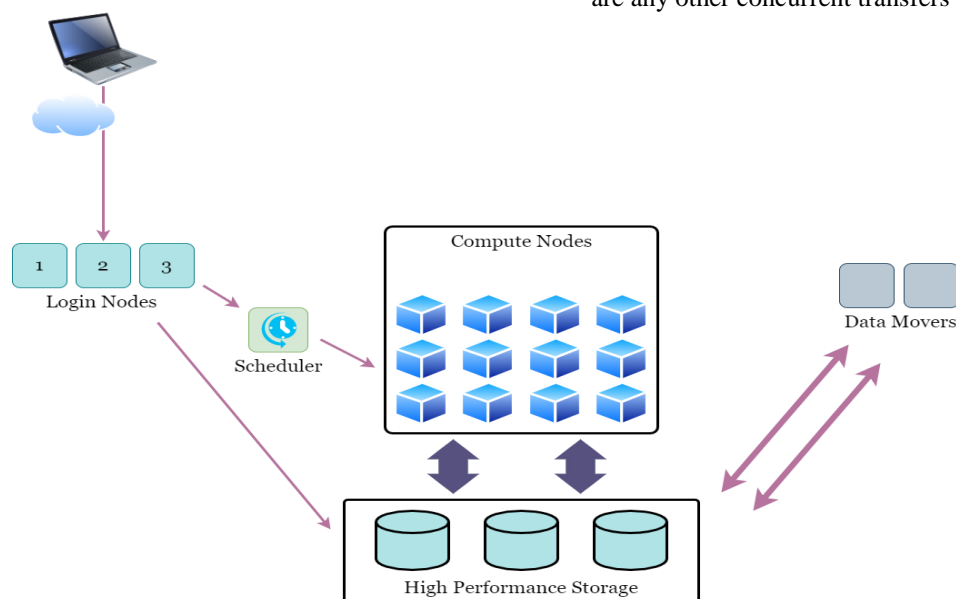
## 2 State-of-the-Art Supercomputing

General-purpose computers (e.g. laptop, desktop) take input data, stores it, and generates output using a serial processing approach. Whereas supercomputers consisting of larger storage volumes and multiple processors can leverage parallel processing to complete the same time-intensive computing tasks much faster (Dennis, 1980; Murray, 1997). Instead of processing one task at a time, it processes many tasks simultaneously.

The most common supercomputer operating system (OS) is Linux based; an open-source, Unix-like OS. Since supercomputers generally work on scientific problems, custom programs have often been written in traditional scientific programming languages such as Fortran and now in more modern languages such as C, C++, and Python.

A representation of an abstract supercomputer allows for understanding its data flow and integrated processes. Figure 2 presents a generalised schematic of an abstract supercomputer, followed by a description of its key components.

- **Login Nodes:** *Login Nodes* provide remote access to the supercomputer and allow users to manage their workflows, source codes, and datasets. Login Nodes are used to submit *jobs* (a unit of work that a job scheduler gives to the operating system) to the *Compute Nodes* via the *Job Scheduler*.
- **Scheduler:** When users submit jobs to the supercomputer, the job *Scheduler* feeds the jobs into the Compute Nodes. In effect, the Scheduler runs jobs on the Compute Nodes on behalf of the user. The Scheduler is responsible for maintaining optimal resources for the supercomputer using a job queue.
- **Compute Nodes:** Programs run on *Compute Nodes*, and the Scheduler provides access to these processors. To execute tasks efficiently (faster and optimally), Compute Nodes consist of fast interconnections between the nodes. Significant performance improvements are achieved by exploiting a parallel processing approach.
- **High-Performance Storage:** *High-Performance Storage* denotes a fast storage component residing inside the supercomputer. This storage is generally private and not shared with other users.
- **Data Mover Nodes:** *Data Mover Nodes* are externally connected servers responsible for transferring data to and from High-Performance Storage. Performance of the Data Movers depends on distance to and capabilities of the other end, plus encryption algorithms and if there are any other concurrent transfers taking place.



**Figure 2:** Schematic of an abstract supercomputer  
(source: <https://icer.msu.edu/sites/default/files/Introductory%20Supercomputing.pdf>)

## 2.1 Kay - Ireland's National Supercomputer

The ICHEC provides the diverse HPC infrastructure required for compute intensive research needs in Ireland, including the National HPC service<sup>1</sup>, Condominium service<sup>2</sup>, PRACE (Partnership for Advanced Computing in Europe) access<sup>3</sup>, EuroHPC Competence Centre<sup>4</sup>, and Academic Training<sup>5</sup>. *Condominium* access allows for academic researchers to use the computing resources of the national HPC system. Many third-level institutions, including TU Dublin, are already registered with the Condominium Service, and research students can contact their local Access Contact Point to gain access to the HPC system.

*Kay* is the name given to the primary high-performance supercomputer provided by ICHEC. It can execute high computation/memory-intensive processes in fields such as biomedical research, drug discovery, nanotechnology, genomics and, in particular to this case study, GIScience. In order to support such varied research domains, *Kay* has a wide range of scientific APIs, compilers, and development libraries - known as *modules*. Table 1 shows the list of modules used in the experiments presented in this paper.

**Table 1:** List of modules used in GIScience experiments.

Package Name	Description
CUDA Toolkit	The CUDA toolkit provides a development environment for implementing, optimising and debugging GPU-accelerated applications with NVIDIA GPUs.
GCC	The GNU Compiler Collection (GCC) includes compilers (e.g. C, C++) and supporting libraries (e.g. libstdc++) provided by GNU.
Intel Compiler	The Intel compiler icc/icpc/fortran is the flagship C/C++/Fortran compiler from Intel.
Python/Conda	<i>Kay</i> provides Python programming interpreters to use in experiments. Conda is an open-source package and environment management system.

Briefly, *Kay* is comprised of five major components having different process-specific capabilities. Table 2 summarises the details of each component.

**Table 2:** List of major *Kay* components.

Name of component	Description
Cluster	A cluster of 336 nodes where each node has 2x20-core 2.4 GHz Intel Xeon Gold 6148 (Skylake) processors, 192 GiB of RAM, a 400 GiB local SSD for scratch space and a 100Gbit OmniPath network adaptor.

<sup>1</sup> <https://www.ichec.ie/academic/national-hpc>

<sup>2</sup> <https://www.ichec.ie/academic/condominium-service>

<sup>3</sup> <https://www.ichec.ie/academic/prace-access>

GPU	A partition of 16 nodes with the same specification as above, plus 2x NVIDIA Tesla V100 16GB PCIe (Volta architecture) GPUs on each node. Each GPU has 5,120 CUDA cores and 640 Tensor Cores.
Phi	A partition of 16 nodes, each containing a self-hosted Intel Xeon Phi Processor 7210 (Knights Landing or KNL architecture) with 64 cores @ 1.3 GHz, 192 GiB RAM and a 400 GiB local SSD for scratch space.
High Memory	A set of 6 nodes each containing 1.5 TiB of RAM, 2x20-core 2.4 GHz Intel Xeon Gold 6148 (Skylake) processors and 1 TiB of dedicated local SSD for scratch storage.
Service & Storage	A set of service and administrative nodes to provide user login, batch scheduling, management, networking, etc.

A large, fast storage facility is essential because deep learning projects depend on large data volumes. Briefly, *Kay* provides *Home* storage to store personal files and source code and *Work* storage to store, in our case, the datasets used for DL model training. Table 3 summarises the properties of each type of storage.

**Table 3:** Data storage areas of *Kay*

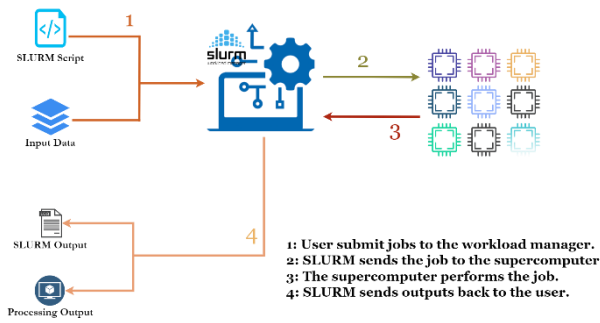
Property	Home	Work
Purpose	Store source code	Store datasets
Path	/ichec/home/users/<username>	/ichec/work/<condominium_name>
Capacity	25 GB	Larger Limit
Access Permission	Only to user	All users in the condominium

### 2.1.1 *Kay*'s Processing Workflow

The processing workflow of *Kay* can be described in four primary phases, and progressing through each of these is essential to complete any task successfully. If any one of these phases fail, the user will not get the desired output. First, the user must submit their job via the *SLURM* (Simple Linux Utility for Resource Management) script file and the input data to the *SLURM* workload manager. The *SLURM* workload manager is an open-source job scheduler for supercomputers and is responsible for managing the processing workflows in *Kay*. The submitted job then passes to *Kay*'s central processing unit, also known as the compute nodes. Users don't have direct access to this part. Once the job is completed, the output files are sent to the workload manager where *SLURM* sends the output files to the specific user. The cost of the compute task is calculated and added to the user's account appropriately. An overview of the processing workflow is illustrated in Figure 3.

<sup>4</sup> <https://www.ichec.ie/eurocc-eurohpc-competence-centre-initiative>

<sup>5</sup> <https://www.ichec.ie/academic/training-education>



**Figure 3:** Overview of Kay's processing workflow.

## 2.2 Mask R-CNN for Building Extraction

Mask R-CNN is a well-known Deep Learning technique applied to instance segmentation problems (e.g. building extraction) in the image processing domain (He et al., 2017). Instance segmentation has also been extended to various other applications such as object detection in videos, reconstructing 3D buildings from aerial LiDAR, and even directing surgery robots. Specific to our spatial change detection problem (i.e. detecting changes in raster images when compared to the current state of OpenStreetMap vector data), it is feasible to use Mask R-CNN instance segmentation results. Therefore, Mask R-CNN models were trained to extract buildings from a given satellite image, and the predictions were then processed to identify any changes in the OSM maps.

Among the many implementations of Mask R-CNN, this study is based on the Mask R-CNN tool implemented by Matterport Inc. (Abdulla, 2017). In addition, Python3, TensorFlow, and Keras APIs were used to integrate the Mask R-CNN tool. The Mask R-CNN implementation is built on the Feature Pyramid Network (FPN) (Lin et al., 2017) and a ResNet101<sup>6</sup> backbone. Three layers (i.e. *heads*, *4+*, and *all*) should be trained to obtain a complete model. Training *heads* involves training a network head layer, *4+* training denotes finetuning layers from ResNet Stage 4 and above, and finally, training *all* layers indicates finetuning all layers of the model with MS COCO datasets (Lin et al., 2014).

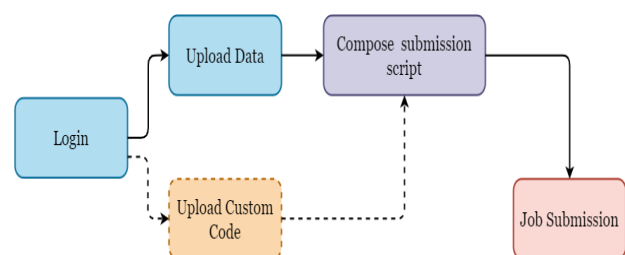
## 2.3 Generative Adversarial Networks for Spatial Change Detection (OSM-GAN)

Many different image processing techniques are used for temporal change detection in images, such as Markov random fields (Gong et al., 2013), principal component analysis (Yousif and Ban, 2013), and CNN based difference image approach (Jong et al., 2019). However, Each of the techniques mentioned above have their own limitations, for example, low performance, low segmentation accuracy, higher time complexity, etc. With

relevant to Generative Adversarial Networks, previous studies applied various image processing techniques for seasonal change detection (Lebedev et al., 2018), heat-map generation (Albrecht et al., 2020), and image classification (de Jong et al., 2019). Inspired by this work, our OSM-GAN approach applies a spatial change detection methodology that uses spatial imagery (satellite images) and OSM vector data (Niroshan and Carswell, 2022a) to train its models. It uses a Generative Adversarial Network (GAN) (Isola et al., 2017) to generate binary feature-maps (i.e. a black&white image that represents particular map features (e.g. buildings) as white blobs) for a given satellite image. Then the prediction is post-processed to extract any changes (new or modified buildings). Results show that OSM-GAN deep learning models can perform a satellite image to feature-map translation with a high confidence level (Niroshan and Carswell, 2022a).

## 3 Machine Learning with Kay

Researchers need a user account on the ICHEC system before submitting a job to Kay. Once the account is approved, users then join an existing project, apply for their own National Service project, or request access through an institution *Condominium* (HPC, 2020). All the use-cases in this paper were tested using the TU Dublin condominium access to Kay. Figure 4 illustrates the next steps to follow after successfully logging into the Kay Supercomputer, and the steps are further described using a real-world image processing case study.



**Figure 4:** The main steps to submitting a successful batch job after logging into Kay. The dotted line path indicates optional steps.

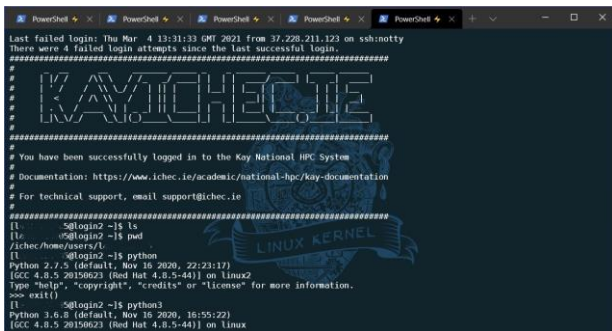
### 3.1 Log in

Users must configure an SSH security key on the local device to remotely access Kay. First, the SSH keys (private and public) need to be generated on the computer used to connect to Kay. Second, the public key needs to be sent to [support@ichec.ie](mailto:support@ichec.ie) from the registered email, and the ICHEC team will copy it to a suitable place; most importantly, the private key should not be sent or shared.

<sup>6</sup>[https://www.tensorflow.org/api\\_docs/python/tf/keras/applications/resnet/ResNet101](https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet/ResNet101)

Once users set up the SSH key, users can log into Kay using an SSH<sup>7</sup> remote connection. The SSH key, ICHEC username and password are then used to initiate an SSH session into Kay's login node. Users can use any preferred command-line interface (CLI) to execute commands (e.g. windows PowerShell, command prompt, Linux Bash). A screenshot after successful login is illustrated in Figure 5.

```
ssh <username>@kay.ichec.ie -i <path for the private key>
```



**Figure 5:** The terminal view of Kay's login node.

### 3.2 Upload Required Data and Codes

The SCP<sup>8</sup> CLI application was used to upload source code and other relevant data in our experiments; however, Kay supports both SCP and SFTP file transfer applications. The required files were compressed before uploading to Kay using the following command. This compression step helps to avoid missing any files, especially when uploading source code. The following commands were used for compression and uploading operations.

- Compression of the source code directory  
`zip -q -r <filename>.zip <directory name>`
- Upload the compressed file  
`scp -i <private_key> <local file path> <username>@kay.ichec.ie:<remote file path>`

### 3.3 Modules

Modules provide extra functionality and services to assist users to build flexible applications. They allow access to specific software, compilers, and libraries used for scientific experiments. Kay consists of different versions of modules, which allows users to prevent dependency errors. For example, different versions of the Anaconda package manager are available to use with different versions of Python programs. Modules should be loaded within the SLURM script, which will appear in the compute nodes. The most useful commands related to modules are listed in Table 4.

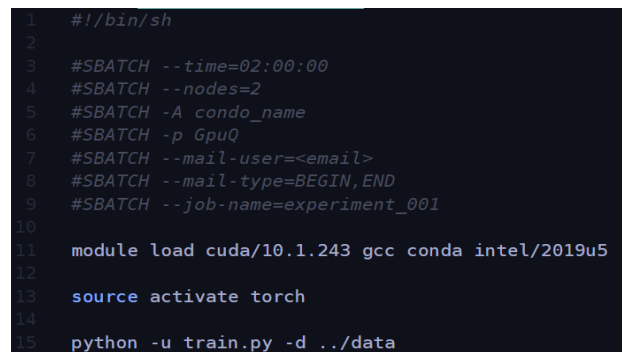
**Table 4:** List of commands to manage modules in Kay

Command	Description	Example
module av	shows modules available in the system.	
module load <name>	loads specified modules	module load conda
module unload <name>	unloads specified modules	module unload conda
module list	lists all modules currently loaded	
module purge	unload all modules	

### 3.4 The Submission Script/SLURM script

A submission script is a shell script that includes all the configurations (SLURM directives) and steps of the job. Since Kay manages compute-intensive workloads using the SLURM workload manager, users must submit batch jobs through submission scripts. The SLURM workload manager optimally allocates "compute nodes" for a user's jobs. Therefore, submission scripts for our experiments were prepared using the following steps. A sample screenshot of the submission script is shown in Figure 6.

- Create a submission script.  
`touch <job_name>.sh`
- Edit the submission script to add instructions of the job.  
`nano <job_name>.sh`  
`or`  
`vim <job_name>.sh`



**Figure 6:** A sample of the submission script. The script accepts Linux commands and SLURM directives. SLURM directives must be prefixed by #SBATCH. This script requires 2 hours of processing time on two compute nodes in the GPU queue. Also, it asks to send an email at the beginning and end of the job.

Although the submission script accepts Linux commands, the SLURM directives must be included to define the essential parameters and configurations. A list of important SLURM directives is listed in Table 5.

<sup>7</sup> <https://linux.die.net/man/1/ssh>

<sup>8</sup> <https://www.unix.com/man-page/linux/1/scp/>

**Table 5:** List of SLURM directives used in submission scripts.

Purpose	#SBATCH option	Example
Specify queue to run the job	-p queue_name	#SBATCH -p GpuQ
Specify account of the user	-A account_name	#SBATCH -A sample_user
Define number of nodes to allocate for job	-N number or - nodes=number	#SBATCH --nodes=2
Indicate email address	--mail-user=email_address	#SBATCH --mail-user=user@email.ie
Send email when job starts	--mail-type=BEGIN	#SBATCH --mail-type=BEGIN
Send email when job ends	--mail-type=END	#SBATCH --mail-type=END
Specify job name	--job-name=jobname	#SBATCH --job-name=experiment_001

### 3.5 Job Submission

We used separate submission scripts for each experiment as an error reduction technique. Once the submission script is completed, it is submitted to the workload manager using the "sbatch" command. This command accepts a single command-line argument - the submission script path. The shell command can be presented as follow.

```
sbatch <submission_script_name>.sh
```

This command queues the submitted job into the job queue, and the workload scheduler assigns the requested resources for the task. The SLURM workload manager consists of a useful set of commands that supports additional controls and for obtaining more information about the submitted jobs (Table 6). A complete set of commands are listed on the ICHEC website<sup>9</sup>.

**Table 6:** Commonly used SLURM commands in the experiments.

Command	Description	Example
sinfo	list details about queues and partitions	sinfo
squeue	return a list of queued jobs. show information of user's jobs	squeue -u \$USER
mybalance	show summary of core hour balance of the account	mybalance

## 4 Geospatial Use Cases

This section describes how to use the Kay supercomputer to train DL models for spatial image change detection applications. First, a Mask R-CNN based building extraction model, implemented using TensorFlow, was

<sup>9</sup> <https://www.ichec.ie/academic/national-hpc/documentation/slurm-commands>

trained. Second, a PyTorch based GAN model (i.e. OSM-GAN) was trained to detect any spatial changes. These scenarios included a number of experiments to build several models using different datasets.

### 4.1 Training Mask R-CNN models on Kay

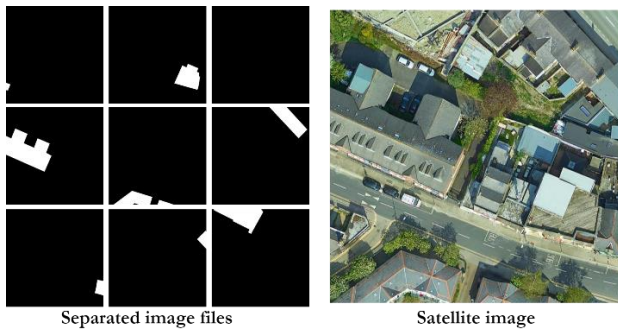
A north inner-city area around TU Dublin was selected to make training data for the Mask R-CNN experiments. Google Earth satellite images and corresponding OpenStreetMap (OSM) building footprint data in this area were mined using customised data crawlers to create the datasets required for DL model training. Since OSM vector data is in *GeoJSON* format (a standard format for encoding map features), the data was converted into binary *.png* image format to distinguish the building edges easily. A training dataset directory consists of a satellite image (*.jpg* format) and all affiliated building footprints in individually separated binary images.

Several datasets (based on varying resolution/zoom levels) were composed using the previously mentioned crawlers. The images (satellite images, associated building footprints) were 768x768 pixels in size. Each dataset is split into two parts: 75% for training and 25% for validation. Table 7 summarises the statistics of each dataset, and Figure 7 illustrates an instance of a dataset.

Two pyramid-based training approaches (low-to-high resolution and high-to-low resolution) were evaluated to find the best strategy for identifying buildings in satellite images. Transfer learning is the core idea behind this training approach – this means several datasets and training sessions were used to build a single model and the final model comprises the knowledge of all intermediate models.

**Table 7:** Datasets used in Mask R-CNN Kay experiments.

	Dataset				
	1	2	3	4	5
Raster Resource	Google Earth				
Vector Resource	OSM				
Resolution	2m	1m	50cm	30cm	15cm
Zoom Level	16	17	18	19	20
Number of satellite images	234	315	1080	4416	17178
Average of building footprints	546	324	93	21	9



**Figure 7:** A data sample used in Mask R-CNN experiments. Each separated feature image (left) and the satellite image (right) are 256x256 pixels in size. All images are stored separately.

Both models were trained using five datasets. On a typical "gamer spec" GPU-enabled laptop (NVIDIA® GeForce RTX™ 2060), a total training time of 2084 minutes (approximately 35 hours) was needed to train a Mask R-CNN model with 3389 data samples. The total training time can be split up into 266 minutes, 595 minutes, and 1223 minutes for *heads*, *4+*, and *all* layers training times respectively.

A **Low-to-High pyramid model** implies training a model starting from a dataset with the lowest resolution, and step by step, increments the training dataset with subsequent higher resolution images. For instance, Dataset 1 (2m resolution) was used to train the first model; then, this model was used as the source model for the next training stage with Dataset 2. Similarly, subsequent datasets were used to train the other intermediate models. Finally, the final model was obtained after training the 4th intermediate model using a dataset having the highest resolution (Dataset 5). A **High-to-Low pyramid model** follows the reverse training sequence of the Low-to-High pyramid model (i.e. the training process starts with a dataset having the highest resolution).

Each dataset and source code were compressed and uploaded to Kay using `zip` and `scp` commands.

- Compression command  

```
zip -r dataset_16.zip data_16
```
- Uploading command  

```
scp -i ~/id_ed25519 ./dataset_16.zip  

abc@kay.ichec.ie:/ichec/work/xyz/abc/data
```

Where: abc=username and xyz=condominium name

The following submission script was used to submit a batch job to Kay after uploading the datasets and the source code.

```
1:#!/bin/sh
2:#SBATCH --time=05:00:00
3:#SBATCH --nodes=2
4:#SBATCH -A abcd
5:#SBATCH -p GpuQ
6:#SBATCH --mail-user= user@email.ie
7:#SBATCH --mail-type=BEGIN,END
8:#SBATCH --job-name=H2L_16
9:module load cuda/10.1.243 gcc conda
intel/2019u5
```

```
10:export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/ichec/packages/conda/2/pkgs/cudnn-7.6.0-cuda10.1_0/lib
11:export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/ichec/packages/conda/2/envs/cpu_tf1.14_torch1.1_hvd0.16/lib/
12:conda activate mrcnn_tf2
13:WORKING_DIR=/ichec/home/users/user_name/DeepMapper/DNN
14:cd $WORKING_DIR
15:python -u train.py -d ./data/16
```

The following section describes the functionality of each statement above:

**Line 1.** When a script starts with a hashbang (`#!/`), the program loader parses the rest of the program/script to a specified program in Unix-like Operating Systems. For instance, in this submission script, the program loader is instructed to use the `/bin/sh` program instead of any other.

**Line 2.** SLURM directive `'#SBATCH --time'` indicates the time required for the job. This configuration will stop the task once the time is up, even though the task is not completed.

**Line 3.** This SLURM directive is used to request the number of compute nodes.

**Line 4.** `'#SBATCH -A'` indicates the account name that needs to be charged for the task.

**Line 5.** This statement indicates the job queue required to execute the task. For instance, `GpuQ` was applied since we chose the GPU queue to run the training process.

**Line 6.** `'--mail-user'` SLURM directive defines the email address where task updates should be sent. Note that, after '=' sign, users have to insert their preferred email address.

**Line 7.** We used both email options to get notification emails when the task starts and ends.

**Line 8.** This assigns a name for the job, which can be used to search for job details later on. For example, the `'sinfo -j H2L_16'` command is used to extract information about the above task.

**Line 9.** This statement is used to load modules required for the task, such as,

- `cuda/10.1.243`
- `gcc`
- `conda` and
- `intel/2019u5`

**Line 10.** This statement links compatible cuDNN and CUDA libraries with the task.

**Line 11.** Appropriate TensorFlow libraries were linked using this bash statement.

**Line 12.** `'conda activate mrcnn-tf2'` command activates the suitable Python environment for the task. For example, `mrcnn-tf2` is the suitable environment for the code.

**Line 13.** This command initiates an environment variable for the working directory containing the source code of the task.

**Line 14.** 'cd' command is used to change the current working directory in Unix-like Operating Systems.

**Line 15.** This command executes the training process stored in train.py.

The submission script to the SLURM workload manager using the sbatch command is shown below.

```
sbatch train_H2L_16.sh
```

An email was received when the task was started, then the *sinfo* and *squeue* commands were used to monitor the job. Also, a notification was received when the training process was completed. Even when a job fails, errors are notified by email and the job's standard output (stdout) is stored in the home directory to resolve any issues. Linux commands "cat" and "tail" can be used for this purpose.

```
cat ~/slurm-823528.out
tail -n 10 ~/ slurm-823528.out
```

Finally, the SCP command is used to download the trained models and other files (e.g., log files, temporary models, and testing outputs).

```
scp -i ~/.ssh/id_ed25519
abc@kay.ichec.ie:/ichec/work/xyz/abc/mod
els/H2L_16.zip ~/.Desktop\
```

The steps mentioned above are repeated until both pyramid models are trained using all five datasets to obtain the final models. The time taken for each training stage of each dataset is listed below (Table 8). The third intermediate model from the High-to-Low feature pyramid approach with 30cm/pixel images achieved the best qualitative prediction results. However, since Mask R-CNN modelling was not deemed accurate enough for predicting spatial changes, an improved method of object detection and segmentation based on Generative Adversarial Networks (OSM-GAN) was proposed next.

**Table 8:** Comparison of training times for Low-to-High Mask R-CNN model.

Dataset	Resolution	Training Times (h:m:s)	Time Quota for layers (seconds)		
			heads	4+	all
1	2 m	05:00:19	3314.90	6138.98	8565.12
2	1 m	04:16:07	2328.43	4725.16	8313.41
3	50 cm	03:01:03	1249.74	2968.89	6644.37
4	30 cm	01:47:18	866.89	1928.70	3642.40
5	15 cm	01:27:12	599.41	1690.27	2942.33

## 4.2 Training OSM-GAN models using Kay

A vector data crawler is first used to mine for applicable OSM vectors (saved in GeoJSON format). Additionally, OSi building footprint vectors are also processed and segmented into GeoJSON files. These two vector data

sources are used for OSM-GAN model training and change detection processes. The separated GeoJSON objects are then translated to binary images (white pixels represent buildings, black pixels represent the background) and stored in separate directories based on their ground coordinates. The binary images are merged into a single 256x256 pixel-sized image using a merging process. Two different bit depths (24-bit colour and 8-bit greyscale) and various pixel resolutions were used to train the models. As a result, the following experiments evaluated sixteen OSM-GAN models in total. Figure 8 presents a single training sample of the OSM-GAN dataset created using a Google Earth satellite image and OSM building footprints.



**Figure 8:** A data sample from the OSM-GAN training dataset. The left side is the input Google satellite image – the right side is a binary image of the current OSM building footprint (feature-map) of the area.

As in the previous experiments, datasets were produced on a laptop and then compressed and uploaded to Kay using the following commands.

```
zip -r ./dataset_1.zip ./google_osm_data
scp -i ~/.ssh/id_ed25519 ./dataset_1.zip
abc@kay.ichec.ie:/ichec/work/xyz/abc/osm
_gan/data
```

Next, a submission script for the training task is created and submitted to the SLURM workload manager. Below is the submission script used in this experiment.

```
1:#!/bin/sh
2:#SBATCH --time=02:00:00
3:#SBATCH --nodes=2
4:#SBATCH -A condominium_name
5:#SBATCH -p GpuQ
6:#SBATCH --mail-user=xxxxx@yyyyy.ie
7:#SBATCH --mail-type=BEGIN,END
8:#SBATCH --job-name=OSM_GAN_T1
9:module load cuda/10.1.243 gcc conda
intel/2019u5
10:export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/ichec/packag
es/conda/2/pkg/cudnn-7.6.0-cuda10.1_0/lib
11:conda activate torch
12:WORKING_DIR=/ichec/work/condominium_name/user
name/osm-gan
13:cd $WORKING_DIR
12:python -u train.py --dataroot
../data/Google_OSi_24bit_z19/ --name
Google_OSi_24bit_z19 --model pix2pix --
direction AtoB --save_epoch_freq 50
```



The script is then submitted to SLURM Workload Manager to begin the DL model training process. Once processing is complete, the final outputs are downloaded to the local computer for prediction purposes.

Notably, the OSM-GAN model trained with 24-bit, 30cm Google Earth imagery and OSM building footprints scored the highest accuracy (88.4%). A post-analysis of all OSM-GAN model prediction results can be found in (Niroshan and Carswell, 2022b).

## 5 Common Problems and Solutions

This section presents some issues encountered during Kay experiments and how ICHEC advised to resolve these issues.

P1. *Home* storage was exceeded during DL experiments. Temporary files created during experiments can exceed the default storage size of 25GB. Especially when Anaconda<sup>10</sup> is used to manage packages for the Python-based code, the *conda*<sup>11</sup> environment creates large files in the home storage.

S1. Removing unnecessary conda environments. The "du -sh ~/.conda/envs" command provides a breakdown of the storage cost for each conda environment. Then it is straightforward to determine which environment/packages need to be removed. The following commands can be used to manage conda environments.

- Remove a conda environment  
`conda remove --name <env name> -all`
- Viewing a list of installed packages  
`conda list -n <env name>`
- Remove a package from a conda environment  
`conda remove -n <env name> <package name>`

P2. *LD\_LIBRARY\_PATH* is the predefined environment variable in Linux/Unix which sets the path for the *linker* when linking dynamic libraries or shared libraries. If *LD\_LIBRARY\_PATH* is not defined, the program will not execute successfully.

S2. Add *LD\_LIBRARY\_PATH* environment variable to the submission script. The best way to use *LD\_LIBRARY\_PATH* is to export the path after modules are loaded. The following statement can be used to set paths for cuDNN and CUDA:

```
Export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/ichec/packages/conda/2/pkg/cudnn-7.6.0-cuda10.1_0/lib
```

P3. Conflicts between CUDA and cuDNN versions.

The CUDA Deep Neural Network (cuDNN) library is a GPU-accelerated library of primitives for deep neural networks (DNN). It provides implementations for standard DL functions such as forward and backward convolution, pooling, normalisation, and activation layers. CUDA is a parallel computing platform that uses the GPU for general-purpose computing tasks. When a mismatched CUDA and cuDNN are imported, the program will not execute correctly. For example, cuDNN v7.6.0 cannot be used with CUDA 11.2, but it can be used with CUDA 10.1.

S3. Managing CUDA and cuDNN versions carefully. cuDNN support matrix<sup>12</sup> and cuDNN archive<sup>13</sup> provide comprehensive details on reducing CUDA and cuDNN conflicts. Download setup files into the local computer after choosing the suitable CUDA and cuDNN versions (downloading process requires NVIDIA membership). Then use SCP command to upload the setup file into Kay.

P4. Create a conda environment on Kay.

S4. Follow the steps below to create a conda environment.

```
module load conda
conda create -n envAI python=3.7
conda info --envs
source activate envAI
conda install tensorflow-gpu
```

P5. The working storage was overloaded during the training process. The intermediate outcomes of the Deep Learning process exceeded the memory limit, and the process stopped. These files cannot be deleted manually as they are created in a short period of time.

S5. A directory watching program was developed and executed in the login node to remove intermediate output files if necessary. After submitting the job to the SLURM workload manager, this program was executed on the login node.

## 6 Conclusions

Supercomputers offer a powerful computing platform for scientific discovery in many fields. The advances in AI and DL algorithms in particular remind us that the use of common desktop/laptop research computers is no longer enough to accomplish (efficiently) many modern compute-intensive problems.

This paper presented a practical use of the Kay supercomputer in GIScience research. The demonstrated

<sup>10</sup> <https://www.anaconda.com/open-source>

<sup>11</sup> <https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>

<sup>12</sup> <https://docs.nvidia.com/deeplearning/cudnn/support-matrix/index.html>

<sup>13</sup> <https://developer.nvidia.com/rdp/cudnn-archive>

image processing use-cases required CUDA support to train its models – a process that took days to complete on a traditional laptop with a single GPU. The use of Kay significantly reduced training times from days to hours (Table 8). In comparison, training a Mask R-CNN model on a single NVIDIA GeForce RTX 2060 GPU (typical "gamer spec" laptop) took 35.5 hours (~1.5 days). As such, Kay helped enormously when conducting multiple experiments using different image datasets/resolutions and DL model parameter variations.

Also discussed were some practical run-time issues that could arise when running jobs on Kay. The ICHEC provides an excellent user support mechanism, well-documented user guides and tutorials, and additional online events for further consultations with users.

## Data and Software

Google Earth satellite images and OSi orthophotos were used with OSM vector data to train and validate Deep Learning models in these experiments. Sections 4.1 and 4.2 explicitly describe the relevant data resources and how they were prepared. Google (raster images) and OSM (vector footprints) data crawlers can be accessed in the GitHub repository:

<https://github.com/Lasith-Niro/DeepMapper-Backend>.

From a software perspective, Python-based Mask-RCNN and Generative Adversarial Network implementations were used in the experiments. SLURM scripts used by Kay are published in the following GitHub repository: [https://github.com/Lasith-Niro/kay\\_scripts](https://github.com/Lasith-Niro/kay_scripts)

## ACKNOWLEDGMENTS

The authors wish to thank all contributors to the OpenStreetMap project and gratefully acknowledge Ordnance Survey Ireland for providing both raster and vector data for the experiments. This research is funded by Technological University Dublin College of Arts and Tourism, SEED FUNDING INITIATIVE 2019-2020. The authors also wish to acknowledge the Irish Centre for High-End Computing (ICHEC) for their provision of computational facilities and support.

## References

Abdulla W., Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow, Github, 2017.

Albrecht C. M., R. Zhang, X. Cui, M. Freitag, H. F. Hamann, L. J. Klein, U. Finkler, F. Marianno, J.

Schmude, N. Bobroff and others, "Change detection from remote sensing to guide openstreetmap labeling," ISPRS International Journal of Geo-Information, vol. 9, p. 427, 2020.

Bishop CM. Pattern recognition. Machine Learning. 2006 Feb;128(9).

Chaturvedi S, Mishra V, Mishra N. Sentiment analysis using machine learning for business intelligence. In2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI) 2017 Sep 21 (pp. 2162-2166). IEEE.

De Jong K. L. and Bosman A. S., "Unsupervised change detection in satellite images using convolutional neural networks," in 2019 International Joint Conference on Neural Networks (IJCNN), 2019.

de Jong KL, Bosman AS. Unsupervised change detection in satellite images using convolutional neural networks. In2019 International Joint Conference on Neural Networks (IJCNN) 2019 Jul 14 (pp. 1-8). IEEE.

Dennis J.B. Data flow supercomputers. Computer. 1980 Nov 11;13(11):48-56.

Gong M, Su L, Jia M, Chen W. Fuzzy clustering with a modified MRF energy function for change detection in synthetic aperture radar images. IEEE Transactions on Fuzzy Systems. 2013 Feb 26;22(1):98-109.

He K, Gkioxari G, Dollár P, Girshick R. Mask r-cnn. InProceedings of the IEEE International Conference on Computer Vision 2017 (pp. 2961-2969).

Isola, P.; Zhu, J.Y.; Zhou, T.; Efros, A.A. Image-to-Image Translation with Conditional Adversarial Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 5967–5976.

Lebedev M. A., Y. V. Vizilter, O. V. Vygolov, V. A. Knyaz and A. Y. Rubis, "Change Detection in Remote Sensing Images Using Conditional Adversarial Networks", International Archives of Photogrammetry, Remote Sensing & Spatial Information Sciences, vol. 42, 2018.

Lin T. Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, Microsoft coco: Common objects in context. In European Conference on Computer Vision 2014 Sep 6 (pp. 740-755), Springer, Cham.

Lin T.Y., Dollár P, Girshick R, He K, Hariharan B, Belongie S. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2017 (pp. 2117-2125).

- Murray C.J. *The supermen: The story of Seymour Cray and the technical wizards behind the supercomputer.* John Wiley & Sons, Inc.; 1997 Jan 1.
- Niroshan L. and Carswell J.D. OSM-GAN: Using Generative Adversarial Networks for Detecting Change in High-Resolution Spatial Images. 5th International Conference on Geoinformatics and Data Analysis (ICGDA 2022), Paris, France, January 2022, Springer Lecture Notes on Data Engineering and Communications Technologies.
- Niroshan L. and Carswell J.D. Post-Analysis of OSM-GAN Spatial Change Detection. 19th International Symposium on Web and Wireless Geographical Information Systems (W2GIS 2022), Konstanz, Germany, April 2022, Springer Lecture Notes in Computer Science.
- Siau K, Wang W. Building trust in artificial intelligence, machine learning, and robotics. *Cutter Business Technology Journal.* 2018 Mar 26;31(2):47-53.
- Vamathevan J, Clark D, Czodrowski P, Dunham I, Ferran E, Lee G, Li B, Madabhushi A, Shah P, Spitzer M, Zhao S. Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery.* 2019 Jun;18(6):463-77.
- Vedovello C. Firms' R&D activity and intensity and the university–enterprise partnerships. *Technological Forecasting and Social Change.* 1998 Jul 1;58(3):215-26.
- Wang B, Lu J, Yan Z, Luo H, Li T, Zheng Y, Zhang G. Deep uncertainty quantification: A machine learning approach for weather forecasting. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining 2019 Jul 25 (pp. 2087-2095).*
- Wooldridge, Michael. "A brief history of artificial intelligence; what is it, where we are and where we are going." (2021): 272. Flatiron Books.
- Yousif O, Ban Y. Improving urban change detection from multitemporal SAR images using PCA-NLM. *IEEE Transactions on Geoscience and Remote Sensing.* 2013 Mar 14;51(4):2032-41.