Classification of land cover types in NZ satellite imagery

Xavier Miles

February 11, 2021

Abstract

Image learning is an area of statistical learning which aims to construct models based on training images, that can then provide predictions for testing or future images. This summer project aimed to use artificial neural networks on satellite imagery to identify and classify different rural regions within New Zealand. The remote sensing field is concerned with using cameras and sensors to determine some information about distant objects, so this project falls within this field. Despite rural sensing being a well-established discipline, there is limited academic work about applying new machine learning techniques to these types of problems, especially with the specific types of networks used in this project. This paper reviews various types of (convolutional) networks that have been developed for image learning tasks, and two of these were chosen to be tested in this project. Satellite imagery of all rural New Zealand from the Sentinel-2 satellites was collected using a Python API, and this data collection process could easily be adapted to obtain more satellite images in the future. The two chosen networks were trained to identify regions of water and vegetation (two foreground classes) in the NZ satellite images, using manually generated labels/annotations. The Mask R-CNN model was able to achieve mIOU and mF1 scores of 0.73 and 0.82, respectively, which measure the accuracy of the predictions. The Deeplabv3+ model was trained on this dataset, but was just predicting a single class for every image, so further work is required to get this model working properly. Later, the land cover database maintained by Manaaki Whenua was used to obtain an different set of groundtruth labels, which have seven foreground classes. Mask R-CNN was tested on the satellite images with the second set of labels, but further work is required to reduce the computational expense of this more-complex dataset, as this prevented this network from be able to train and make predictions. The final section will discuss the key findings and challenges in this project, and suggest key improvements or modifications for any subsequent related projects between Stats NZ and the UC SAIL group.

1 Introduction

Humans are constantly identifying and classifying objects within whatever scene they are viewing. Digital image processing aims to translate this process into a set of instructions so that computers can perform the various object detection tasks that is subconsciously performed by people. Training a model to automatically detect and identify objects can be extremely useful, as it will allow the automation and improvement of processes that were done previously done by people, often through long hours of tedious work. Some current applications of image processing are using car-mounted cameras to assess the quality of roads to prevent potholes and other deformations, medical imaging (locating tumours, diagnosis etc.), facial recognition software used in passport gates, and the car-mounted sensors that enable self-driving cars. In this work, the aim is to use artificial neural networks to create an automated system which can classify the different types of land covers in a satellite image of rural New Zealand. This project was commissioned by Stats NZ, who interested in exploring the possibility of utilising such an automated system to reduce the amount of surveying they would have to perform, as this could significantly reduce their costs. For example, Stats NZ could use the system on satellite images of a farm to detect total area of farmland, area of planted crop, and size of herd, which would enable them to reduce the size of the survey farmers are expected to fill out yearly (currently about 40 pages).

There are four main types of image learning problems; classification, semantic segmentation, object detection, and instance segmentation. These are illustrated in Figure 1. Classification aims to categorise an image and produces a label (or set of potential labels) for the entire image. Semantic segmentation aims to detect objects in an image and provide accurate object boundaries via per-pixel classifications. Object detection aims to extract and categorise multiple objects within an image using rectangular bounding boxes. Instance segmentation extends object detection by identifying and classifying object instances using a more detailed boundary, which means that the predicted object *masks* can be arbitrary shapes (not just rectangular). Both object detection and instance segmentation models can differentiate between different instances of the same type of object, and can allow predicted instances to overlap one another. Note that, in Figure 1, the colours in the object detection and semantic segmentation images correspond to the object types (person, sheep, dog), while there is a distinct colours in the instance segmentation image for each instance since it differentiates between instances of same-class objects.

This project aims to identify land cover types from satellite images with accurate boundaries, so a semantic or instance segmentation model will be necessary. The regions of land covers should not overlap since the images are birdseye view of a rural landscape, and it is not important to distinguish between different instances of the same land cover types, which means that the tested models should perform semantic segmentation (per-pixel classification). However, instance segmentation models can be used, but only if the output is converted to per-pixel classifications so that the predictions can be evaluated equivalently to the predictions from semantic

segmentation models.

The rest of the paper is organised as follows. Section 2 includes some background research in the remote sensing field and discusses some options for how this could be used to adapt and improve the modelling done in this project. Some deep network models for each of the four image learning problems, from least to most complex, will be discussed in Sections 3. The hardware and software requirements, data collection and preparation, and data ethics are included in Section 4. Then, Section 5 outlines any important considerations for model setup and tuning, and then presents the results (and non-results) from the experimental studies. Finally, Section 6 discusses any significant limitations of this paper and provides potential directions of future related work.

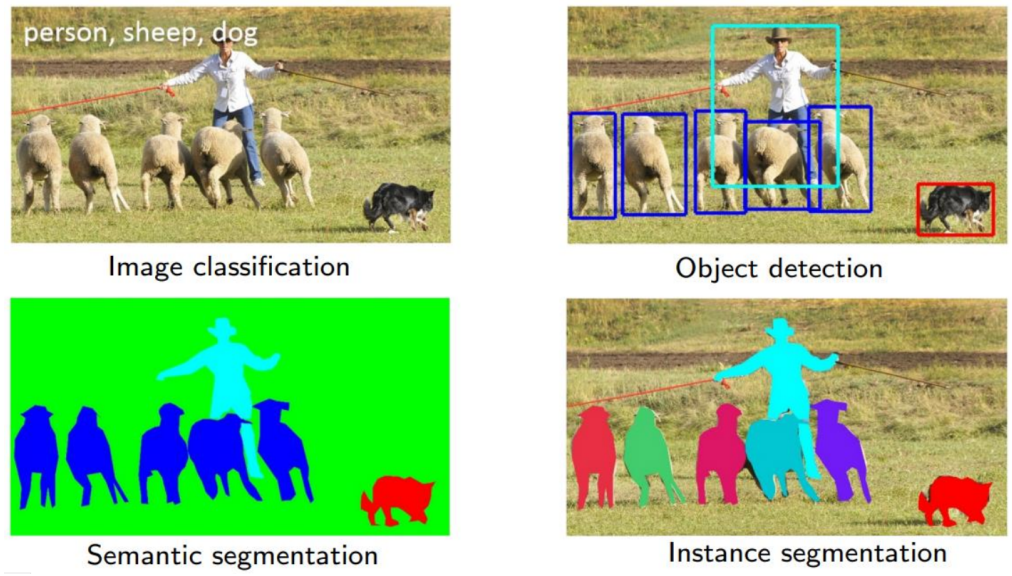


Figure 1: Types of Image Learning Problems (Source Lin et al. 2014)

2 Background Research - Remote Sensing

This section will provide some context for this project, which aims to classify land cover types using satellite imagery, and thus is a type of *remote sensing* problem. Due to this project's short timeframe, and emphasis being more on applying and understanding the machine learning models than thoroughly researching the field, the amount that this section's content informed the project was limited. However, this section provides a resource for future work on this project, as better land cover predictions are likely to be obtained by better integrating the machine learning models with established techniques and best practices in remote sensing.

The field of remote sensing is concerned with the gathering of information about an object at a distance from the object. Often, remote sensing problems will be about gathering information about the Earth using planes or satellites, and this typically will be achieved by detecting the speed and direction of reflections of electromagnetic waves. In this project, the gathered satellite images of New Zealand could include parts of the electromagnetic spectrum other than the visible colour portion, since the satellites collect this information (refer to Section 4 for more information about satellites used). The aim of this project is to perform *passive* remote sensing, since the reflection of the sunlight is detected by the sensor, as opposed to *active* remote sensing, where the reflection of an emitted signal is detected by the sensor. Examples of passive remote sensing are meteorological satellites or film photography, and examples of activate remote sensing are radar and lidar.

Using remotely sensed imagery for land cover classification is a well-established problem, as this can have further use cases, such as disease mapping or studying climate change (Aplin 2004). The use cases of classified land cover types can usually be grouped into environmental management, which is concerned with the monitoring and control of natural resource-exploitation, and environmental understanding, which is concerned with the scientific analysis of natural and artificial processes related to determining land cover types. The most likely use case of a land cover classification system evolved from this project would be to inform Stats NZ of the activity and resources of farms, which comes under environmental management. However, this project could still benefit from aspects of environmental understanding, as this could help equip the predictive models with more robustness and versatility (through better interpretation of different land cover types).

Many use cases of land cover classifications involve comparisons across different points in time. Given that classifications are for images, which are snapshots in time, this requires collecting multiple images of the same areas over different time points. Since the satellite images should be standardised across time, with respect to atmospheric conditions/corrections, it usually simplest to use the same source as this means that the data processing steps should be consistent. Many multitemporal studies will just perform post-classification comparisons, but the comparisons could be made using *Normalised Difference Vegetation Index* (NDVI) (Kastens and Legates 2002) or other novel metrics derived using texture and context analysis

(Herold, Scepan, and Clarke 2002). This project just focused on the type of comparisons that could be made post-classification, since it a fundamental part of it is testing the capabilities of the segment-then-classify algorithms.

Scott et al. 2017 used various neural network models to achieve above-95% accuracy of land cover classification using high-resolution imagery. They are performing image classification without any object localisation methods, which is a simpler problem than the one this project aimed to solve. However, Scott et al. 2017 do show that transfer learning and image data augmentation both lead to significantly higher accuracy, and these techniques were used in this project (refer to Section 5). Kussul et al. 2017 present a novel neural network architecture to perform semantic segmentation, especially designed to segment and classify land cover types in satellite imagery. Their model outperforms alternative machine learning methods, random forest and multilayer perceptron ensembles, when tested on satellite imagery of rural Ukraine. This is a model that could be used in future work with NZ satellite imagery, and would provide a comparison to the models used in this project.

3 Model Review - Networks for Images

3.1 Neural Network

Artificial Neural Networks (NN) are a type of statistical model which aims to simulate the incredibly-complex process occurring in a human brain by setting up a sequence of *neurons*. Each neuron within a NN has an associated set of weights (coefficients) and bias (constant term) which are used to get a weighted sum of the input information. The output from this linear combination is then fed into a non-linear *activation function*. The data is input into the first *layer* (group) of neurons, which then outputs numbers using the associated weights, biases, and activation function, and this output serves as input for the next layer of neurons. This *feed-forward* process is continued until the output layer is reached and a prediction is given. Since NN are usually very complex, they provide almost no clear statistical inferences and are mostly used for predictions.

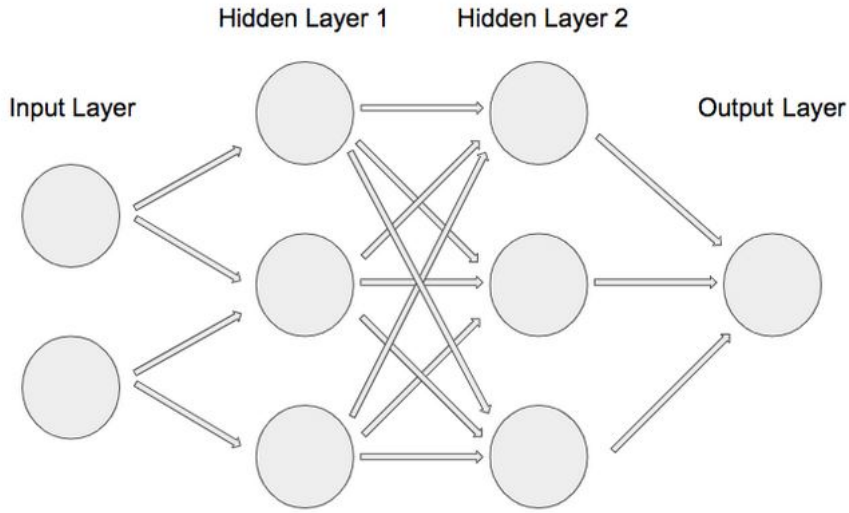


Figure 2: Simple Neural Network (Source Berner 2020)

Neural networks have an input layer, one or more hidden layers, and an output layer. Neural networks are typically referred to as *deep networks* once they have more than a few hidden layers, since it becomes infeasible for a human to predict what the numerical output of the model will be given an input. Figure 2 illustrates a simple network with two hidden layers. Note that an output layer can contain multiple nodes that can then be translated into a prediction for classification or regression. In this network, the first hidden layer is *fully-connected* to the second hidden layer, since all of the nodes in the first hidden layer are connected to all of the nodes in the second hidden layer. The input layer and output layers are not fully-connected to their respective hidden layers, but it is common for them to be.

If every layer in a network is fully connected to its adjacent layers, then it is referred to as a *full-connected network*.

There is various other components that can be added to a neural network and these attempt to allow the model to understand/capture more complex relationship.

3.2 Convolutional Neural Network

Convolutional neural networks (CNN) utilise the hierarchical structure of data by extracting information from a small regions and then combining this to form more complex patterns. CNNs employ a mathematical operation called convolution, and this step in a CNN is referred to as a convolution layer. A convolutional layer requires a $(n \times n)$ kernel, which is then convoluted with each $(n \times n)$ subsection of the 2-D grid, such as illustrated in Figure 3. These models are extremely well-suited to image datasets since images contain very hierarchical information due to adjacent pixels tending to be similar colours (or at least having some relationship). Most neural networks which are used for image processing will contain a convolution layer, but typically introduce other layers or relationships, which means that (vanilla) CNNs can provide a baseline model to compare against when training more complex/deeper networks.

It has been shown that CNN can outperform other predictive techniques when performing image classification tasks, such as recognising handwritten digits (numbers and letters) (LeCun et al. 1998). However, CNNs have a relatively simple network structure, which means that more network components and relationships are required to detect and classify multiple objects within an image (object detection).

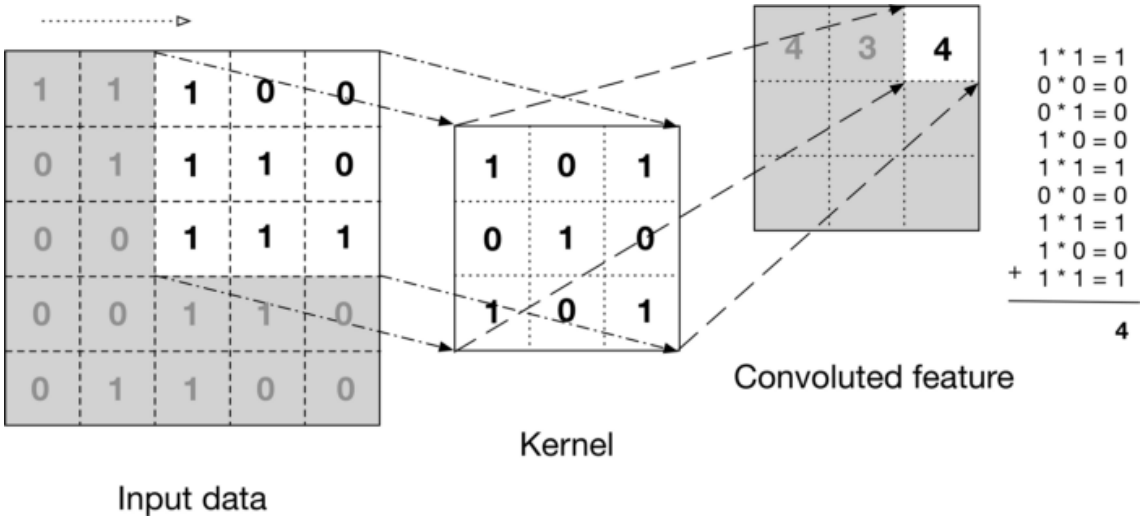


Figure 3: Example convolution layer with (5×5) input grid and (3×3) kernel (Source Yen 2018)

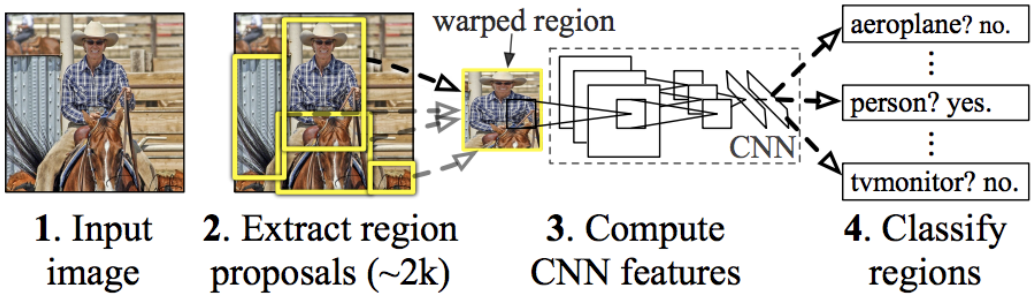


Figure 4: R-CNN Structure (Source Girshick et al. 2015)

3.3 Object Detection: Region-based Convolutional Network

The Region-based Convolutional Network (R-CNN) adapts the traditional CNN structure to be able to perform object detection in an image by constructing a number of local regions where it has detected an object and determining the type of object (Girshick et al. 2015). R-CNN uses a search algorithm to generate 2000 candidate regions (which are subsets of the entire image), warps these candidate regions in square images, and uses these as input to a CNN. This process is visualised for an example image in Figure 4.

Since an object can be at any location and scale within an image, an exhaustive search for candidate regions would best ensure that all objects are detected by the model, but this is computationally infeasible due to the large search space. Instead, R-CNN employs a selective search algorithm that is designed to capture all scales, use diverse techniques so it can handle different image quality and conditions, and be fast to compute. This algorithm initially generates a large number of candidate regions and then uses a recursive/greedy process to merge the most similar candidate regions (using a similarity measure) until there is only 2000 candidate regions remaining, which serve as the input to a CNN structure.

As the R-CNN applies a CNN model to square images which are warped rectangular sub-regions within the image, any objects it detects will be contained within a rectangular *bounding box*. This is useful about knowing the presence of an object and it's general location, but gives limited information about the boundary of the object, which could be used to better isolate the object from the rest of the image.

3.4 Improving Object Detection: Fast/Faster R-CNN

Fast R-CNN is an objection detection algorithm aimed at remedying the main weaknesses of R-CNN to provide significantly faster training and testing speeds with improved detection accuracy (Girshick 2015). Fast R-CNN improves R-CNN by

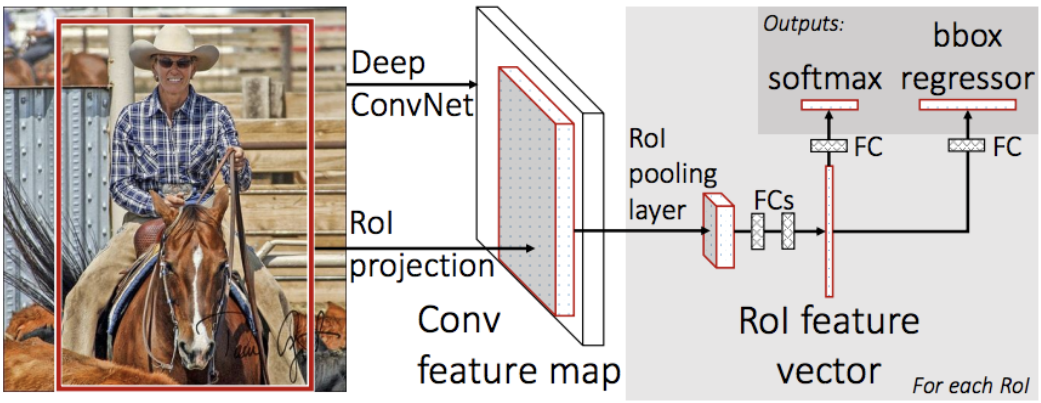


Figure 5: Fast R-CNN Structure (Source Girshick 2015)

feeding each image into a CNN to generate a *convolutional feature map*, from which the regions of interest (RoI) can be extracted, where the RoIs are equivalent to the candidate regions individually fed through a convolution layer. This process means there is only only one convolution operation performed per image, rather than 2000 convolution operations per image like R-CNN. Also, the selective algorithm used in R-CNN is a fixed process which does not attempt to improve itself, which may lead to bad candidate regions being generated and computational resources being unnecessarily wasted. Fast R-CNN still uses this algorithm to initially generate the candidate regions, but outputs information on how to improve these so that the efficiency and performance can improve.

Once the RoIs are extracted using the convolutional feature map, they are then warped into a square, reshaped to a fixed size using a RoI pooling layer, and then fed through fully connected layers to get RoI feature vectors. The RoI pooling layer uses max pooling to shrink the RoIs (of arbitrary size) into a small feature map of a fixed, preset size (layer hyperparameter), and the RoIs must be warped into squares so that fully connected layers can be used. Given a RoI feature vector, a softmax layer is used to predict the class of the proposed region/object and a different (regression) layer is used to output bounding-box regression offsets which indicate how the RoI could be adjusted (in all four directions) to better accommodate the object.

The process to obtain output information from an image using Fast R-CNN is visualised in Figure 5.

Faster R-CNN further iterates on the Fast R-CNN model by addressing the largest computational bottleneck; the identification of candidate regions (Ren et al. 2016). The selective search algorithm used in R-CNN and Fast R-CNN is a relatively slow process which inefficiently uses the data/model structure to improve the exposition of candidate regions. Faster R-CNN abandons selective search in favour of a novel network component called *Region Proposal Network* (RPN), which allows the model to more effectively learn local regions to classify (or not) as various types of objects. RPN is a fully-connected network, and takes a convolutional feature map as an input

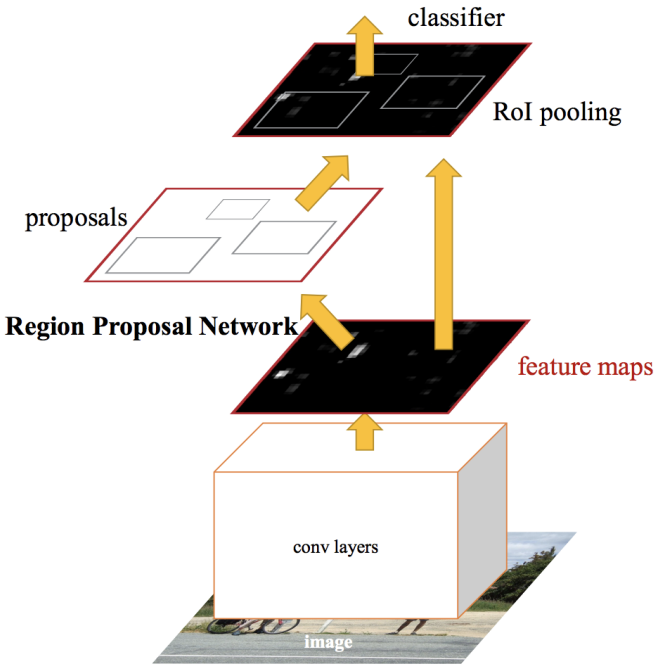


Figure 6: Faster R-CNN Structure (Source Ren et al. 2016)

and outputs a set of rectangular candidate regions with a set of associated scores of how likely each of the regions are to be objects.

The region proposals and the convolutional feature map are then input into a Fast R-CNN object detection network to determine the presence and type of objects within the image. This entire model structure is visualised in Figure 6. Note that the convolutional feature map is derived from the original image via a sequence of convolutional layers. The RPN and Fast R-CNN components of the model can be trained independently but this would lead to different trained parameters within the convolutional layers. Therefore, Ren et al. 2016 consider three different techniques for training this network, but decide to test using *4-step alternate training* which works by alternating between training the RPN and the Fast R-CNN components while keeping the other part fixed. However, the authors state that *approximate joint training* was found to provide a trained Faster R-CNN model which performed marginally worse than 4-step alternate training (mAP of 69.9% vs 70.0%), but significantly reduces training time (decrease of 25-50%). It is possible there are others changes to the Faster R-CNN training process which could further reduce training time, but any changes seem unlikely to significantly improve performance metrics (eg mAP).

3.5 Semantic Segmentation: Deeplab

When performing semantic segmentation on an image, the goal is to identify regions which contain objects along with the class of the objects, which is equivalent to

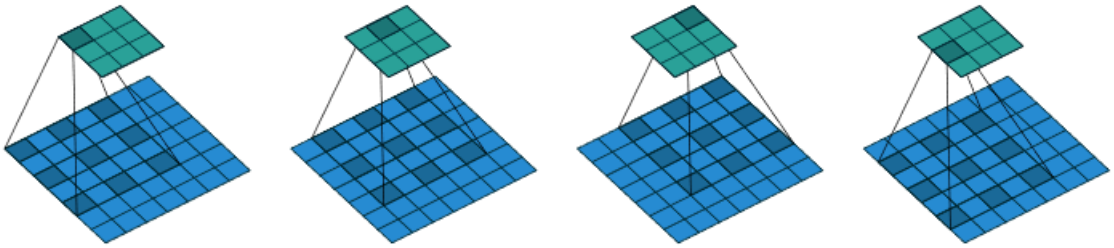


Figure 7: Atrous convolution with a 3×3 kernel, a 7×7 input, and dilation rate 2 (Source Dumoulin and Visin 2016)

performing pixel-wise classification. By convention, a pixel is classified as *background* if it is not classified as one of the target objects. Fully Convolutional Networks (FCN) extends the traditional CNN structure to efficiently learn to make dense predictions for per-pixel tasks (like semantic segmentation), and is an example of a Deep Convolutional Network (DCNN). The FCN model achieves the per-pixel classification through alternating convolution and pooling layers and the upsampling the resulting information to get a feature/class map, which allows it to handle images of an arbitrary resolution to be input. However, the greatest disadvantage to the FCN structure is the combination of convolution and pooling layers means that the output class map will often be in low resolution and fuzzy object boundaries.

Deeplabv1 aimed to achieve less fuzzy object boundaries and higher resolution class maps than previous DCNNs (Chen, Papandreou, et al. 2017). There tends to be a large reduction in spatial resolution due to the large number of pooling and down-sampling/stride in DCNNs, so Deeplabv1 removes the down-sampling from the last few max-pooling layers and adds upsampling to the subsequent convolution layers, calling this new method of convolution with upsampling *atrous convolution*. This leads to Deeplabv1 producing feature/class maps which are output at a higher resolution than alternative models, which can lead to higher mIOU scores representing better predicted object masks. Figure 7 visualises atrous (or dilated) convolution, where the top layer is computed from the bottom layer, with a dilation rate of 2 which creates a gap of 1 (in either direction) between the source gridpoints/pixels.

To address fuzzy object boundaries, Deeplabv1 applies bilinear interpolation and fully connected Conditional Random Fields (CRF) to the output score map. Bilinear interpolation is the equivalent of linear interpolation for interpolating functions of two variables (in a two-dimensional space), and is achieved by performing linear interpolation in one of the variables (directions on a grid) and then repeating for the other variable. With respect to a score map, bilinear interpolation corresponds to inserting pixels into the existing grid and determining their value/colour from the adjacent, pre-existing pixels. A CRF is a graphical structure that connects pixels (nodes) to each other (edges) and enforces that their predictions are similar. This had often been done with locally-connected CRF models to ensure that nearby pixels have similar predictions, but Deeplabv1 employed a fully connected CRF

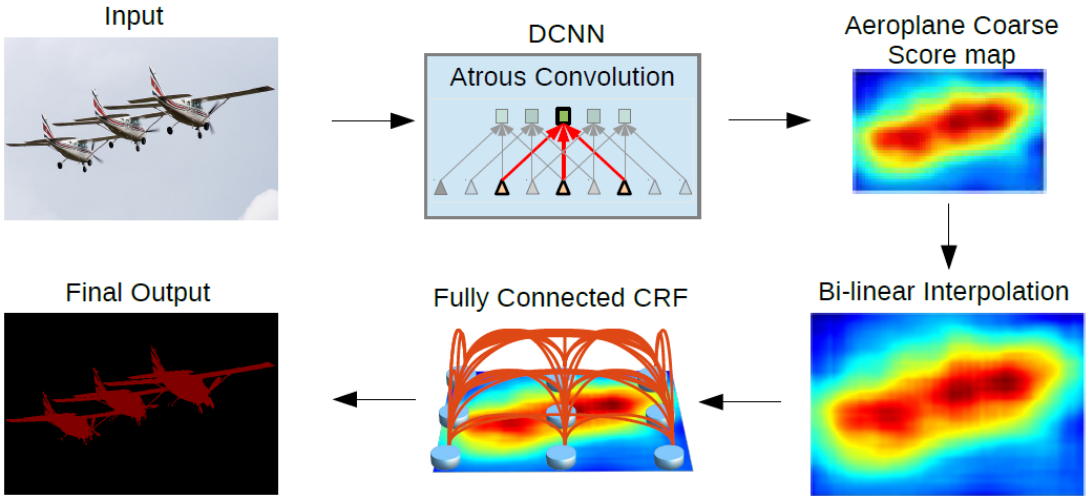


Figure 8: Deeplabv1 Structure (Source Chen, Papandreou, et al. 2017)

model (with a Gaussian CRF potential), which allows their model to capture any long-range dependencies between pixels that the previous approaches ignored. The entire process of obtaining a semantic segmentation prediction from the Deeplabv1 model is shown in Figure 8.

To improve the performance beyond the Deeplabv1 model, Deeplabv2 targets effectively recognising and classifying objects at multiple scales (Chen, Papandreou, et al. 2017). The typical approach adopted by DCNNs to this challenge is to train using re-scaled versions of the same image and then aggregate the feature or score maps obtained for each image. However, He, Zhang, et al. 2015 showed that regions of any scale can be efficiently and effectively classified by resampling convolutional features obtained from a single scale, as demonstrated in their *Spatial Pyramid Pooling* R-CNN method called SPP-net. Inspired by this SPP structure, Deeplabv2 introduces *Atrous Spatial Pyramid Pooling* (ASPP) which uses parallel atrous convolution layers with different sampling rates to generate features that are further processed (separately) before being fused to generate the final feature map. An ASPP layer is visualised in Figure 9, and this shows one of the sets of convolutions sizes (rate=$\{6,12,18,24\}$) that was tested by Chen, Papandreou, et al. 2017.

Deeplabv3 redesigned the ASPP component to include a separate global image pooling to capture global features (large-scale relationships), and then input this information into the ASPP structure along with the existing sets of convolutions. This modification is beneficial to predictive performance since atrous convolution tends to complicate the integration of local and global relationships, and the ASPP component now explicitly integrates these in a single step. Also, the fully connected CRF is removed from Deeplabv3, since it no longer offers any improvement to the final segmentation output, which is likely because the complexity of Deeplabv3 is enough to naturally reduce the type of uncertainty in predictions that CRF aims to mitigate. This new structure is illustrated in Figure 10.

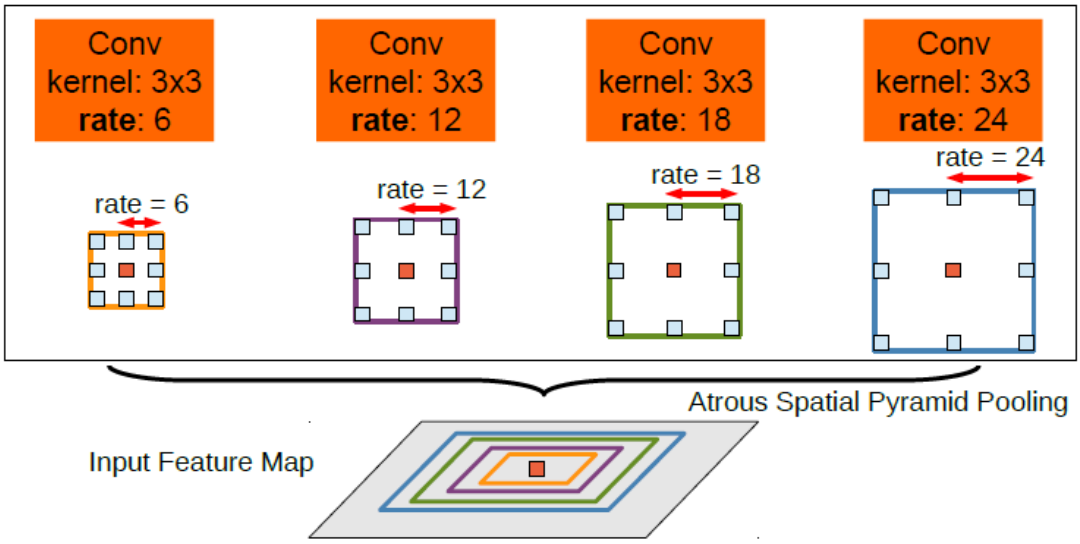


Figure 9: Atrous Spatial Pyramid Pooling (ASPP) (Source Chen, Papandreou, et al. 2017)

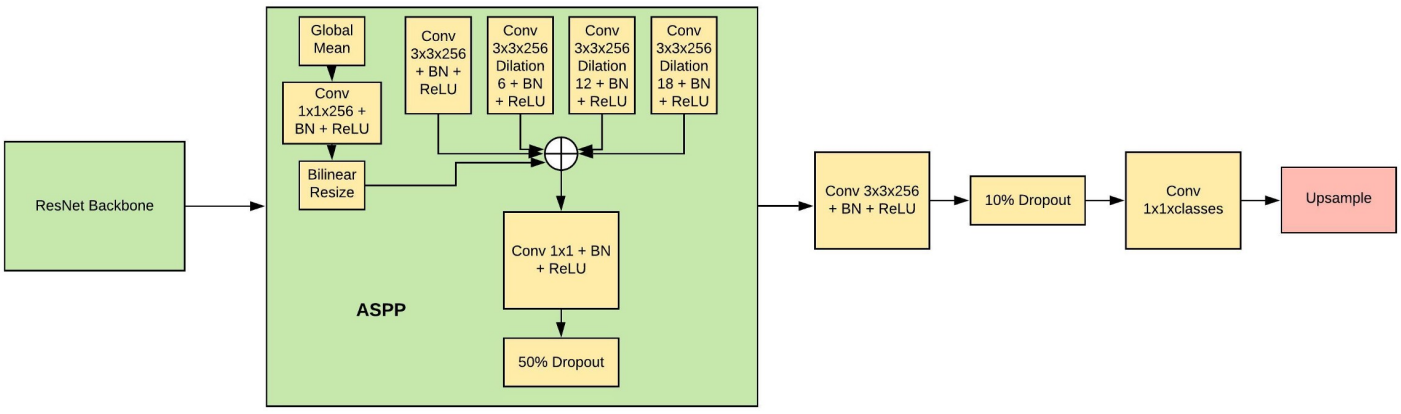


Figure 10: Deeplabv3 Structure (Source Li 2020)

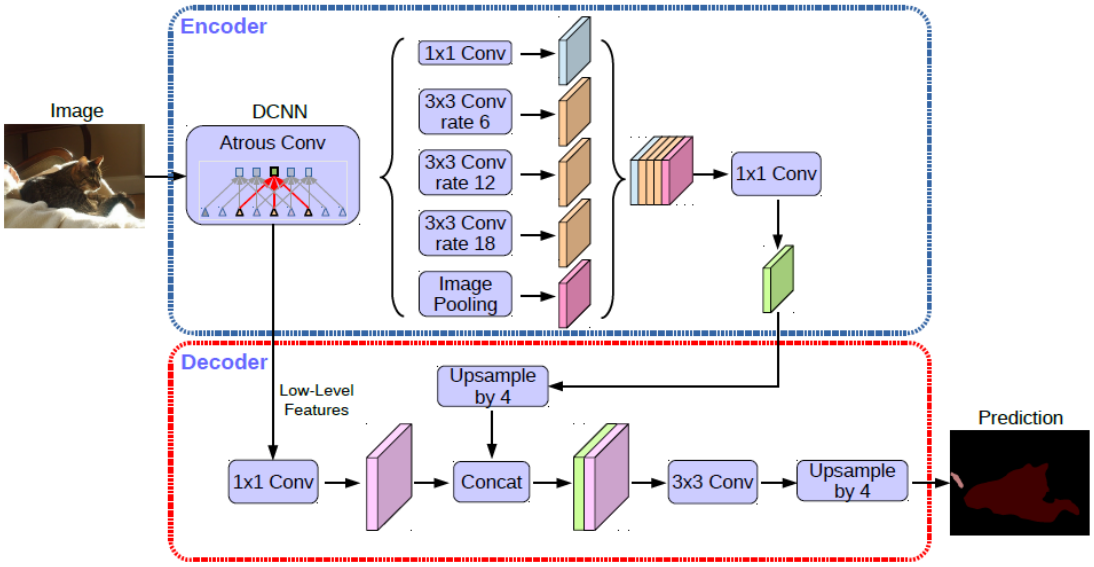


Figure 11: Deeplabv3+ Structure (Source Chen, Zhu, et al. 2018)

Deeplabv3+ aims to improve the ability of the model to recover spatial information to better capture object boundaries, which it achieves by introducing an encoder-decoder structure (Chen, Zhu, et al. 2018), as illustrated in Figure 11. The encoder module reduces the dimension of the feature map to capture higher semantic information which may rely on recognising general shapes and long-distance pixel relationships. The decoder module recovers the spatial resolution so that the local relationships (ie object boundaries) are better interpreted and predicted.

Deeplabv3+ also uses the Xception backbone network, which offers better performance and lower computational cost than ResNet, which was the backbone used in the previous versions of Deeplab. The advantages of Xception are enabled by *depthwise separable convolution*. Depthwise convolution is when convolution is applied to each of the channels of an image (ie Red, Green, Blue) using a separate kernel for each channel. Pointwise convolution is when convolution is applied to the channels of an image together to give an output image with the same width and height as the original image but only one channel. Depthwise separable convolution is the combination of applying depthwise convolution then pointwise convolution, which means that spatial information is collected for each channel independently (depthwise) and the information is combined from all channels (pointwise).

Figure 12 illustrates the convolutions required to perform depthwise separable convolution for an image/grid with three channels. In this illustration, (a) then (b) would produce depthwise separable convolution; whereas (c) then (b) would produce atrous depthwise separable convolution.

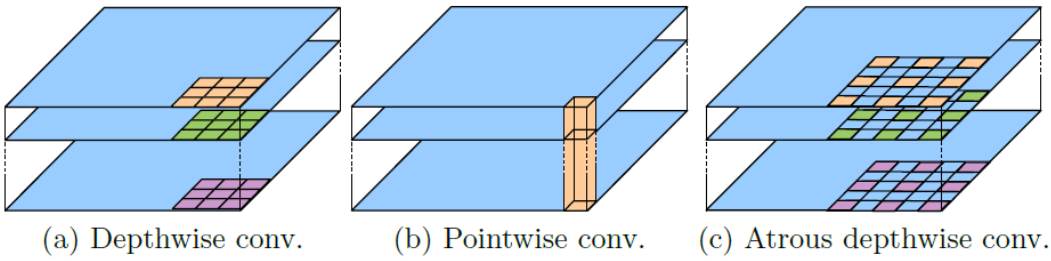


Figure 12: Components of Depthwise Separable Convolution (Source Chen, Zhu, et al. 2018)

3.6 Instance Segmentation: Mask R-CNN

The Mask R-CNN model extends the Faster R-CNN structure to be able predict object masks, thus performing instance segmentation rather than just object detection (He, Gkioxari, et al. 2017). Like Faster R-CNN, Mask R-CNN first uses a RPN to obtain candidate object bounding boxes. Unlike Faster R-CNN, Mask R-CNN then outputs a binary mask for each RoI, in parallel to the network that predicts the class and box offset. This means that for each RoI, the mask and class prediction are performed separately, which is inspired by the requirement of the mask output to extract much finer spatial layout of an object. He, Gkioxari, et al. 2017 find that the parallel structure of Mask R-CNN allows the model to outperform other models where an object's classification depends on it's mask predication.

Additionally, the developers of Mask R-CNN demonstrate the flexibility of the Mask R-CNN model through estimation of human poses. He, Gkioxari, et al. 2017 use one-hot encoded masks (ie dummy variables) and train Mask R-CNN to predict K masks, one for each of K key body parts (eg right shoulder, left knee). Through a simple adaption, the authors demonstrate Mask R-CNN can reliably predict human poses, and show that this can be combined with the vanilla Mask R-CNN (which predicts person segmentation masks) to gain more information from an image of people. Furthermore, Mask R-CNN is shown to outperform some alternative deep networks which were developed specifically for keypoint detection.

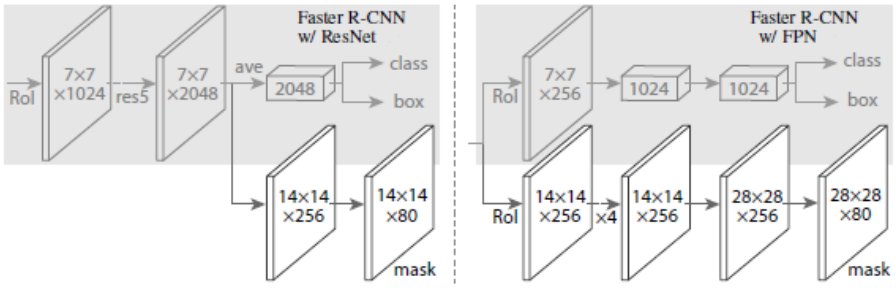


Figure 13: Mask R-CNN Structures with ResNet C4 (left) and Feature Pyramid Network (right) Backbones (Source He, Gkioxari, et al. 2017)

4 Data Collection and Processing

4.1 Environment Requirements - Hardware and Software

The training and testing of the models was performed on a dedicated computer (referred to as *GPU machine*) which uses Ubuntu 18.04 (Linux) and contains a NVIDIA GeForce RTX 2080 Graphics Processing Unit (GPU). It is important to use a GPU for training and testing, instead of a Central Processing Unit (CPU), as these computer chips are specifically designed for handling image processing operations. MobaXterm was used to SSH (Secure Shell session) into the GPU machine, which means that the GPU machine could be accessed from any computer using the University of Canterbury (UC) network. By using the Remote Desktop Connection application, a computer anywhere can access a virtual computer on the UC network, and therefore access the GPU machine via MobaXterm. This meant that the GPU machine could be accessed from a personal computer at home.

This project used the Python programming language, through a mixture of scripts and Jupyter notebooks. Jupyter notebooks are a collection of code blocks, which helps with debugging and checking that a model is running correctly. Sometimes, Google Colab was used to work on the Jupyter notebooks when the testing GPU machine was not available. Google Colab offers a online virtual computer on which to run Jupyter notebooks, but this is less powerful than the GPU machine and requires uploading the entire dataset to Google Drive, which takes time and could lead to unintended inconsistencies in testing conditions (eg. wrong files or files in wrong folders).

The models used relied on the Python packages Tensorflow and Keras (with Keras being an extension of Tensorflow). The Python code was run within Conda environments, which allows the installation of Python packages into dedicated environments. For example, the Mask R-CNN model required Tensorflow 1 so it was run in a Conda environment which included Tensorflow 1 (and other packages), whereas the Deeplab model required Tensorflow 2 so it was run in a different Conda environment which had Tensorflow 2 installed. This meant that the different versions of the same software (Tensorflow) could be installed onto the same computer, and easily accessed, without causing issues.

4.2 Collecting Satellite Images

This project, which was commissioned by Stats NZ (Statistics New Zealand), aimed to provide proof of concept that deep neural networks can be used to automatically detect land cover types within satellite images of rural New Zealand. Stats NZ had no preference about which neural network was used, but were interested in whether a reasonable level of prediction accuracy could be achieved with freely available satellite images. Stats NZ and the Spatial And Image Learning (SAIL) group had

agreed that commercially-purchased satellite images could be considered later either when it was found that the freely available images were not of sufficient quality or the project was ready to be tested more rigorously for deployment, but freely available satellite images were found to be sufficient for this (preliminary) project. The two main sites which regularly capture satellite imagery of the entire world and offer free downloads are Sentinel and Landsat, which both include a collection of satellites. Sentinel was chosen for this project since this gave higher spatial resolution images (10 vs 30 metres), where the spatial resolution is defined in terms of the physical width and height corresponding to one pixel.

The sentinelhub Python package was used to collect satellite images from Sentinel-2 satellites. This package also offers the ability to download images from the Sentinel satellites, the Landsat satellites, and some other satellite systems. This process requires creating a login on the sentinelhub website and adding appropriate API keys to the Python script. This API/website offers a month-long free trial which includes a limited number of processing capacity, and this was (barely) sufficient to collect the images for this project (ie. there was no processing units left over). If this system was used to collect more data in the future then it would be necessary to either contact sentinelhub for a free plan available for research and pre-commercial use or pay for a commercial plan.

A *SentinelHubRequest* object is used to create a request to the sentinelhub API/website, which requires specification of:

- an *evalscript* string which includes information about which spectral bands to retrieve and how to construct a pixel representation from these.
- which satellite to request information from.
- time interval over which to search for recorded images and method to pick which of these to return.
- bounding box (rectangle) which specifies the area over which to retrieve satellite images in a specified coordinate system.
- (width, height) of the image in pixels. Typically, the *bbox_to_dimensions* function is used to construct these dimensions to the desired spatial resolution for a given bounding box.
- *config* object which contains the API keys (specific to the user).

The steps in the Python script to retrieve the satellite images were:

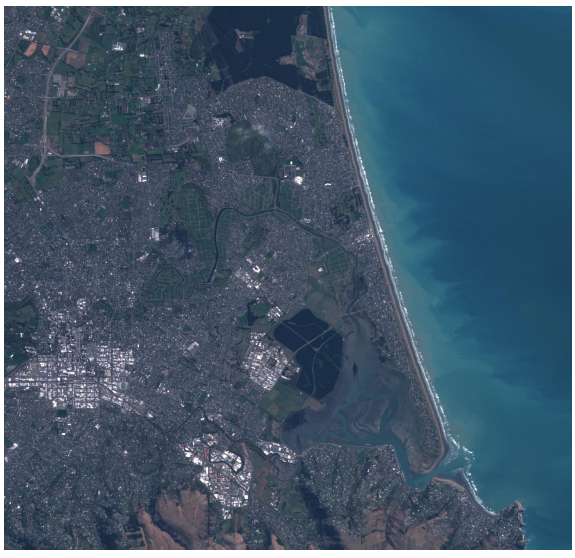
1. Import the outline of New Zealand (in WGS84 coordinates) from CSV file and convert to a MultiPolygon object using shapely.geometry.shape(...).
2. Split the New Zealand shape into a list of bounding boxes using OsmSplitter(...) and OSM zoom level 11.

(a) Cambridge (1520 x 1563)

(b) Lakes Rotoma and Rotoehu (1535 x 1543)

(c) Wellington (1435 x 1503)

(d) Amberley (1459 x 1395)



(e) Christchurch (1450 x 1387)

(f) Rakaia (1432 x 1393)

Figure 14: Examples of Retrieved Satellite Images (width x height in pixels)

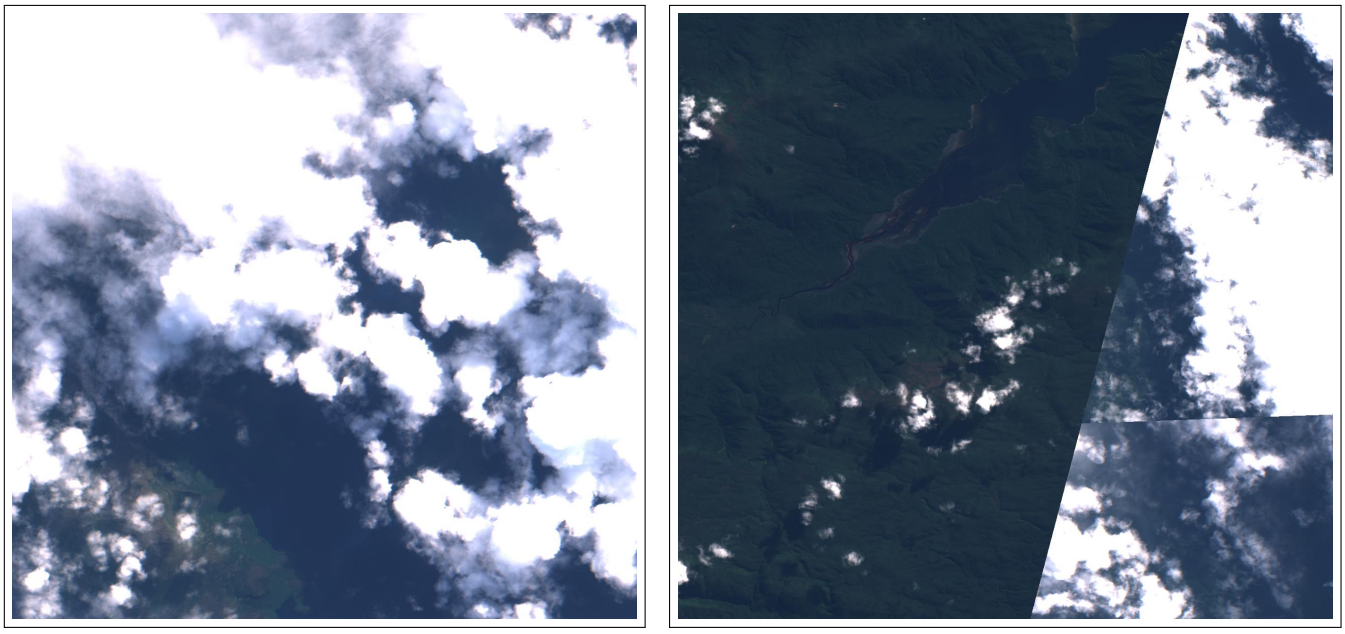
3. For each of these bounding boxes, retrieve satellite images using a SentinelHubRequest object and save these in JPEG format (with the coordinates included in the filename).

Since a lot of the data preparation required visually inspecting and verifying the images, the satellite imagery data was collected in the standard Red-Green-Blue (RGB) colour bands, but the Sentinel-2 satellite collects twelve spectral bands and some other spatial data. These other spectral bands could potentially be used to construct training data with more information to train and test the neural network models, which could lead to more accurate predictions. However, only four of the spectral bands offer 10 metre resolution (Blue, Green, Red, NIR) and the rest offer a variety of lower resolutions, which means care should be taken when choosing which bands to download and how to integrate these together. Since the images seemed natively relatively dim, the evalscript was modified to increase the value of the RGB integers by a factor of three, which corresponds to increasing the image brightness by a factor of three.

The request to sentinelhub searched for satellite images during March-April 2020 and specified that the sentinelhub website should mosaic (stitch together) the images in this time interval to provide final images with the "least cloud-coverage". It was important to minimise the amount of clouds in the images since clouds obscure what is on the ground, thus degrading the ability of the model to detect the land cover type during training and testing. This automatic mosaicing did a reasonable job at producing images without clouds, but did not completely prevent clouds and there were some sharp edges of clouds in images, as shown in Figure 15. Autumn was chosen as the time to search for images to avoid brightness issues that might occur in Summer, glare issues that would result from snow on the mountains in Winter, and issues related to the large number of clouds likely present during Winter. Further testing could determine whether searching over a different or longer time interval results in better quality images with respect to glare/brightness, cloudiness, and other seasonal factors. Alternatively, the s2cloudless Python package provides access to the sentinelhub cloud detector and could be used to customise the cloud-prevention process, but this may significantly increase the complexity of the data collection process.

4.3 Annotating the Images

After collecting the 1572 satellite images using sentinelhub, these images were split into three folders and manually annotated by the three team members using the VGG Image Annotator (VIA) website. This involved mapping out polygon shapes and adding a label to each of these polygons. The two area types labelled were water and vegetation, and these were sourced by visually inspecting the images. This annotating involves a large amount of variability, as the regions drawn may differ from person-to-person and may even differ from day-to-day with the same person. This means that this is imprecise (and subjective) data that was being



(a) Majority-Cloud Image (b) Sharp Cloud Edges from leastCC Mosaicing

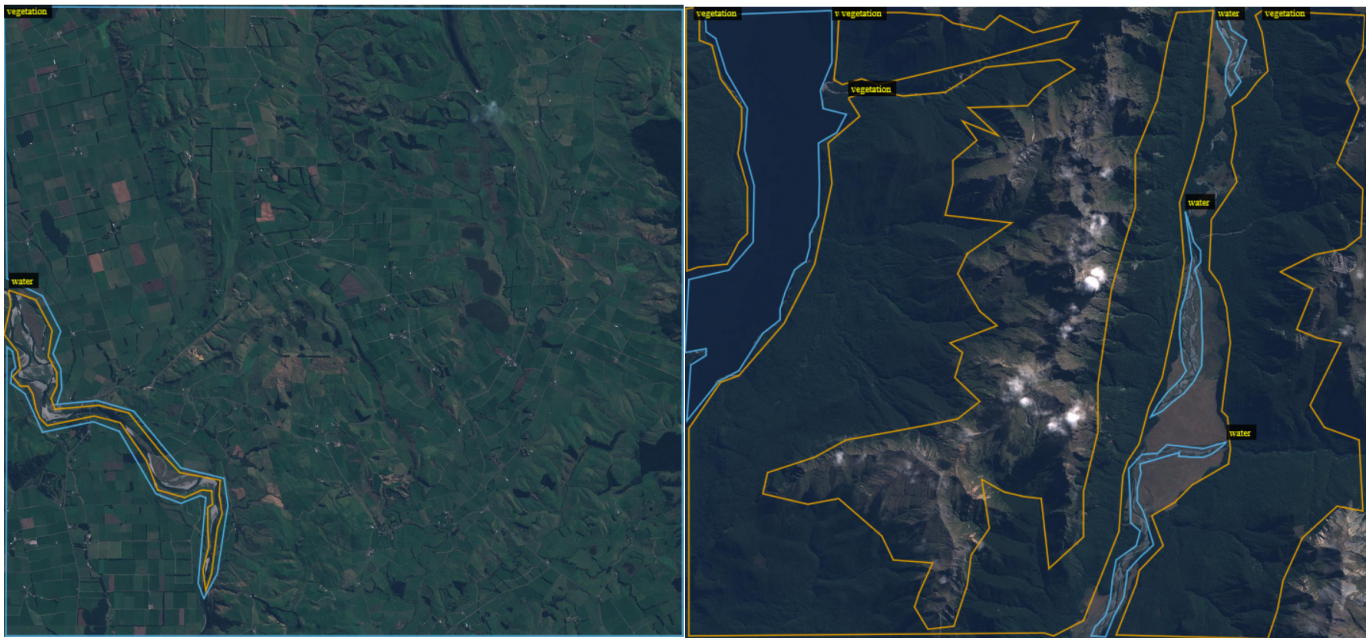
Figure 15: Cloudy Image Examples

used to inform the models, but it served as a starting point that can be used to setup the networks and test their general capabilities. Also, hopefully some of the inconsistency was averaged out by including a large number of images.

Because the annotating of these images was menial and extremely time-consuming, not all of the images ended up being labelled. There was 1,001 annotated images with 3,855 instances (1,491 water and 2,364 vegetation). Two screenshots of labelled satellite images on the VIA website image are included in Figure 16. It can be seen in (b) that the boundary between vegetation and barren/un-vegetated land is not immediately clear, and this may lead to inconsistency between labels on different images. Furthermore, even the (a) introduces some subjectivity-related complications; should the riverbed be labelled as water (as water might be there sometimes) and should empty crop fields be labelled as vegetation (as these are likely to contain crops sometimes).

4.4 Annotating the Images V2.0

After annotating the images, the imprecision in the annotations (vegetation vs rock, where water bodies end and shore begins etc.) was discussed with Stats NZ, who suggested looking into the Land Cover Database (Mainland New Zealand) which is maintained by Manaaki Whenua. This offers a set of land cover labels which are more specific (35 low-level classes or 7 high-level classes) and of higher quality (polygons with more vertices) since they are produced by experts in determining land environment. This dataset was downloaded as a CSV file which included the



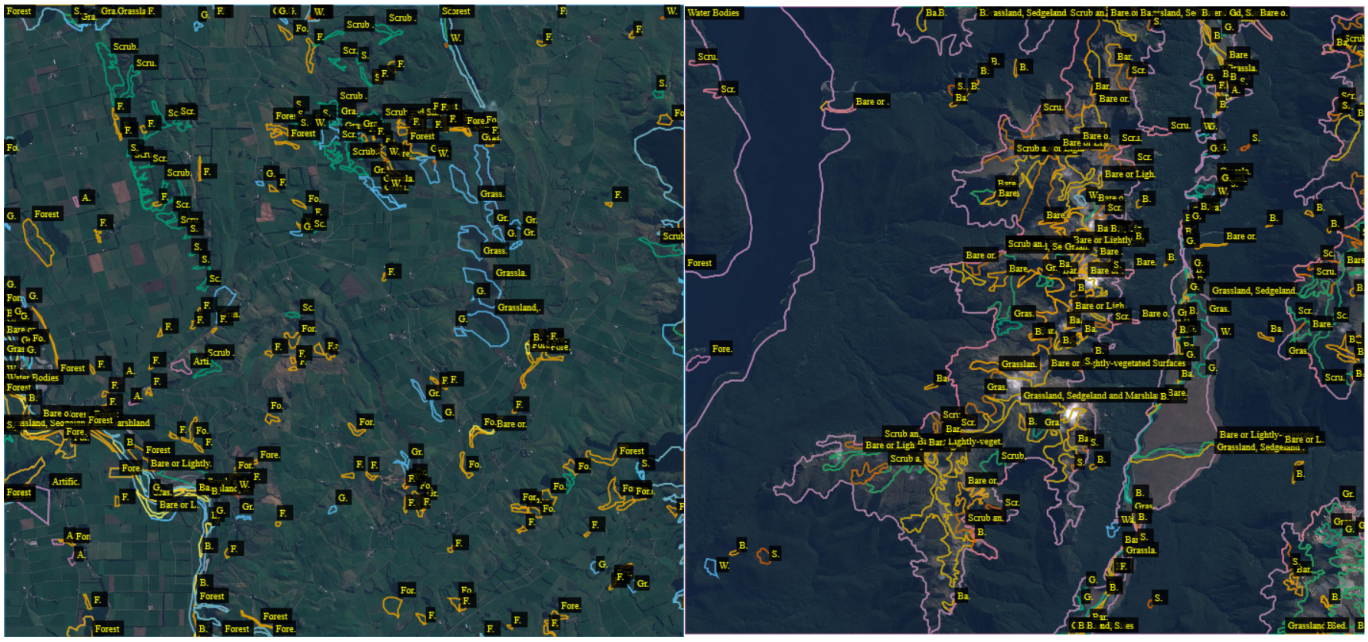
(a) Easier Example (b) Harder Example

Figure 16: Two Manually Annotated Images

land cover type of everywhere in (mainland) New Zealand for 1996, 2001, 2008, 2012, 2018. The 2018 classifications were extracted as these are the most up-to-date information. The previous classifications cannot be used with the Python scripts/functions developed during this project as the first Sentinel-2 satellite was launched in 2015, so a different source of satellite imagery would need to be used to match these time periods.

The python script then loops through the bounding boxes for each of the satellite images, checked which of the polygons (regions) overlapped with the given bounding box, extracted the location and class of the overlapping polygons, and then wrote this information to a JSON file (in the same format as the JSON files generated by the VIA website). Storing the bounding box of each of the images in their filename was useful, because finding the bounding boxes just required finding all the image filenames and parsing these strings for the relevant information. Storing the annotation JSON files in the same format meant that any of the Python code developed to run models can be tested with either the manual or Manaaki Whenua annotations.

Because this process of annotating the images could be performed computationally in a reasonable amount of time, it meant that all of the images could be annotated (took about 8 hours on a 10th Gen Intel Core i5 laptop). There was 1572 annotated images with 571,716 instances (six classes). Note that the process may have produced neighbouring instances of the same class, so the actual number of distinct/separate instances will be less than 571,716 (but will still be significantly more than 3,867). The same satellite images from Figure 16 are displayed in Figure 17, but with the labels extracted from the Manaaki Whenua land cover database. It



(a) Easier Example (b) Harder Example

Figure 17: Two Automatically Annotated Images

is hard to distinguish instances and read their type since the satellite images both have over 250 different regions, but the models should be capable of comprehending this amount of information (and hopefully capable of inferring some patterns in the information).

4.5 Data Ethics

Since the spatial resolution of the satellite images is 10 metres, there is no possibility of identifying individuals in the pictures. There is also extremely limited ability to identify larger personal objects, such as cars or houses. The Sentinel satellites do regularly capture remote sensing information (ie. electromagnetic waves), so this project could be continued with satellite imagery of the same regions across different time points. This could then be used to track changes in large properties, such as farmland/crops or building construction sites, which are cases that Stats NZ is interested in since it would reduce their reliance on surveying. Any long-term visual tracking of the daily activity of private property could concern some New Zealand residents, but this project did not have sufficient image resolution (or time) to establish tracking systems.

5 Model Setup and Results

5.1 Hyperparameters

Each deep neural network has a collection of parameters, and these are being constantly being changed during training so that the output predictions from the model better match the training data. However, there is a collection of hyperparameters that can be adjusted before training the model, and these affect the model's performance. Two key examples are the loss function and the optimiser algorithm, which will be fully explained in the next section. There is often model-specific examples, but some hyperparameters that are common to most CNNs are:

- number of training steps - at each step, the parameters in the network are changed to better match the training images. This is set through the number of epochs and steps per epoch (eg. 35 epochs and 2000 steps/epoch gives 70,000 training steps). At the end of each epoch, the model is assessed on the validation set, which is used to track the training progress but has no influence on the training process.
- batch size - the number of training images that the network simultaneously optimises against at each training step. The default for Mask R-CNN is 2, but it can be much higher (eg. 16, 32, 64).
- image resolution - the images should be scaled to a consistent image pixel resolution before being input into the networks. Square images are often used for simplicity, with black padding added to the sides to preserve the original aspect ratio. In this project, (1024×1024) was typically used.
- learning rate - essentially how fast the network tries to train. If too high, the network will diverge and stop producing sensible predictions. If too low, a very large number of training steps will be required to produce good predictions.
- weight decay regularisation - used to prevent the parameters from becoming too large, which can help to prevent bad predictions. It is the same concept as lasso (L1) and ridge (L2) regression.

5.2 Loss function and Optimiser

The loss function measures the difference between the actual and predicted segmentations, and is the objective function that the model training uses to decide how best to change the parameters. The Mask R-CNN model has a loss for the object bounding boxes, a binary cross-entropy loss for the masks, and a loss for the classification of each instance. These three losses are able to be defined separately due to the parallel structure of Mask R-CNN, as explained in Section 3.6, but

are combined to give an overall loss value (which is used by the optimiser). The Deeplab implementation can use any of the Keras loss functions, and was set to use *sparse_ categorical_ crossentropy* in this project. The softmax function can be used to obtain class probabilities, and is defined as

$$f(s)_i = \frac{e^{s_i}}{\Sigma_{j \in C} e^{s_j}}, \tag{1}$$

where s_j is the model's score for class j and C is the set of class labels. The categorical cross entropy for a pixel is defined as

$$CE = -\sum_{i \in C} t_i \log(f(s)_i), \tag{2}$$

where t_i is the groundtruth/actual label and f is the softmax function. CE is maximised, with a value of zero, if the (softmax) class probability for the actual label is one and all other class probabilities are zero. The *sparse* refers to how the class labels of the pixels were encoded using integers, as opposed to *categorical_ crossentropy* which uses one-hot encoded class labels. The results from these two loss functions will be equivalent given that the data is appropriately re-configured. For this multi-class prediction problem, the only other Keras loss function which can be used is *Poisson*.

The optimiser is the algorithm which is used to minimise/maximise the loss function during model training. The Mask R-CNN package uses *Steepest Gradient Descent* (SGD) and does not offer an option to choose another optimiser, so changing this would require altering the source code. The Deeplab implementation can use any of the Keras optimisers, and was set to use *Adaptive Moment Estimation* (Adam) in this project. Gradient methods are a set of algorithms which use a function and it's first derivative to find local minima of that function. SGD is the simplest of the gradient methods since, at each step, it just changes the parameters to give the steepest drop in function value according to some subset of the data (Wang 2008). SGD is highly intuitive and can be extremely computationally cheap, which is why it is often used when training neural networks. However, a model trained with SGD will often not perform as well as other gradient methods due to ill-conditioning, saddle points, and other potential complicating factors related to the loss function. Adam was used for Deeplab since it offers a computationally efficient algorithm with low memory requirements that tends to outperform SGD and other methods (Kingma and Ba 2014). It may be that one of the other optimisers available on Keras, or even switching from Adam to SGD partway through training as proposed by Keskar and Socher 2017, would provide better model performance. However, any model improvement would likely be marginal, so it was decided that only trying one optimiser was sufficient for this project.

5.3 Performance metrics

Since the desired output is a semantic segmentation for each image, the performance metrics need to calculate the predictive performance of the models based on their

per-pixel classifications. A *confusion matrix* is a table which compares the actual and predicted classifications, and can be constructed per-class or for all classes. Table 1 illustrates how a 2×2 confusion matrix can be used to get the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) rates for a binary response. For example, a confusion matrix like this can be constructed for the class *water*, and the *Yes* row and column would correspond to pixels that are actually and predicted water, respectively.

		Predicted	
		Yes	No
Actual	Yes	True Positive	False Negative
	No	False Positive	True Negative

Table 1: Generic Confusion Matrix

An intuitive measure for measuring the per-pixel performance of a model is pixel accuracy,

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}, \tag{3}$$

which simply measures the percentage of pixels that are classified correctly. However, accuracy does not account well for class imbalance, meaning that a model can score a high accuracy if it performs reasonably well on a majority class despite performing very poor on minority classes (Tiu 2019). For this reason, accuracy was not used to assess the performance of the networks in this project.

The two metrics which will be used to assess the models in this project is the Intersection-Over-Union (IOU) score and F1 score, which both better account for class imbalance. The IOU score is defined as

$$IOU = \frac{TP}{TP + FP + FN}, \tag{4}$$

which corresponds to the intersection of actual/predicted pixels divided by the union of the actual/predicted pixels for a given class.

$$F1 = \frac{2TP}{2TP + FP + FN}, \tag{5}$$

which corresponds to twice the intersection of actual/predicted pixels divided by the total number of actual and predicted pixels (ie. union plus intersection). To assess the model's overall performance, the IOU and F1 scores are averaged across all classes to give the mean IOU (mIOU) and mean F1 (mF1) scores, respectively. (Note that the mIOU and mF1 are constructed using *weighted* averages, where the weights are the number of actual pixels for each class.) As can be seen in (4) and (5), these metrics contain very similar information, but IOU tends to (quantitatively) penalise models more for single instances of bad segmentation. Figure 18 gives a visual illustration of the difference between IOU and F1 scores for different combinations

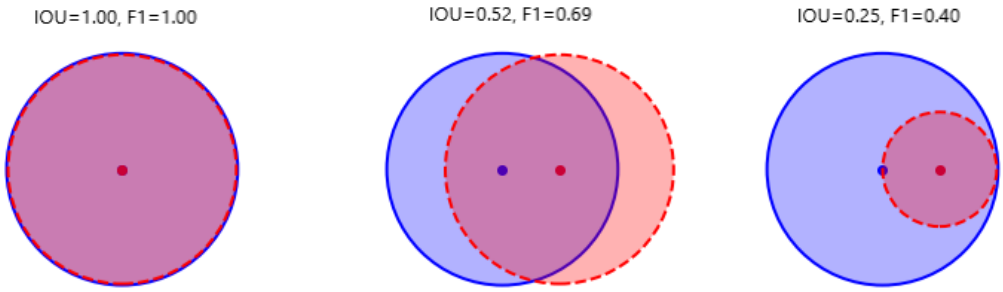


Figure 18: Illustration of IOU and F1 scores for Different Combinations of Actual (blue) and Predicted (red) Regions (Source Monteux 2020)

of actual and predicted regions (for a given class). Therefore, either metric could be used to compare the performance of different models, but mIOU will the primary metric used in this report, since this tends to be reported more often when semantic segmentation models are assessed against benchmark datasets (Papers with Code 2021).

5.4 Mask R-CNN

As explained in Section 3.6, this network performs instance segmentation so will produce a number of distinct instances/regions with classification. The implementation of Mask R-CNN that was used was the *mask-rcnn* Python package (Matterport 2021). This was the first network tried since it has been shown to be able to offer relatively accurate results (for instance segmentation problems) and the implementation used comes with well-documented examples of training and testing the model.

Initially, the inbuilt metrics were used to assess the model predictive performance, which was mean Average Precision (mAP) with given IOU thresholds (typically IOU=0.5 and IOU=0.75), and Arlen 2020 explains how this metric works and what exactly it measures. However, it was believed that mAP with IOU threshold was underscoring the model's performance due to overlapping instances (often with the same land cover type). Also, as was explained in Section 1, this project was aiming to achieve semantic segmentation rather than instance segmentation, and what this metric measures is related but not exactly equivalent to semantic segmentation performance. Therefore a post-processing step was added to convert the instance segmentation output into an equivalent semantic segmentation output, which meant that the predictions could then be assessed using the metrics discussed in Section 5.3. For each image, this post-processing overlaid all the instances onto a 2-D grid to get per-pixel classifications. Each instance come with a confidence score, which indicates how sure the model is that this area contains an object of that given class, and this was used to decide on the class for pixels which were part of overlapping instances of different classes.

First, different learning rates were tested with a small number of training steps to get a sense for which learning rates should be used for the longer tests. Table 2 outlines these preliminary results. Note that this is too small number of training steps for the results to be very conclusive, but it does give some baseline results. The learning rates 0.01 and 0.025 both have the lowest mIOU and mF1, which is because they are both predicting that every pixel of every (validation) image is background. This indicates that the model parameters have diverged, so the model has become useless, and therefore these learning rates are too high and should not be used in the longer tests.

Learning Rate	Training steps	mIOU	mF1	Training time (min:sec)
0.0001	10,000	0.25	0.31	6:03
0.0005	10,000	0.31	0.38	6:05
0.001	10,000	0.18	0.22	5:52
0.005	10,000	0.42	0.51	6:01
0.01	10,000	0.16	0.19	5:58
0.025	10,000	0.16	0.19	5:46

Table 2: Preliminary Results for Mask R-CNN (with manual annotations)

The results for the Mask R-CNN with more training steps are reported in Table 3. The learning rate of 0.001 yielded the best predictive performance, as measured by mIOU and mF1. It can be seen that the increase from 35,000 to 140,000 training steps gave an improvement in model predictions, but the increase from 140,000 to 240,000 training steps did not. This indicates that 140,000 training steps is sufficient for the model to have *converged* to it's best predictive performance given the hyperparameters. More fine-tuning of the hyperparameters is possible to improve the model even more, but the improvements are likely to be very modest. Therefore, it seems that the model can achieve about 70-80 % predictive performance, as measured by mIOU and mF1, on this dataset.

Note how the model trained with a learning rate of 0.0025 and 140,000 training steps gave mIOU of 0.16 and mF1 of 0.19. This model had diverged as it was predicting background for every pixel of every image. The model seemed to diverge at about training step 102,000, as the loss values after this point were all NAN (Not A Number), whereas they were valid numbers before this point.

Learning Rate	Training steps	mIOU	mF1	Training time (hr:min)
0.001	35,000	0.68	0.77	2:48
0.001	140,000	0.73	0.82	9:00
0.0001	140,000	0.69	0.79	11:21
0.0025	140,000	0.16	0.19	8:55
0.001	240,000	0.72	0.81	15:34

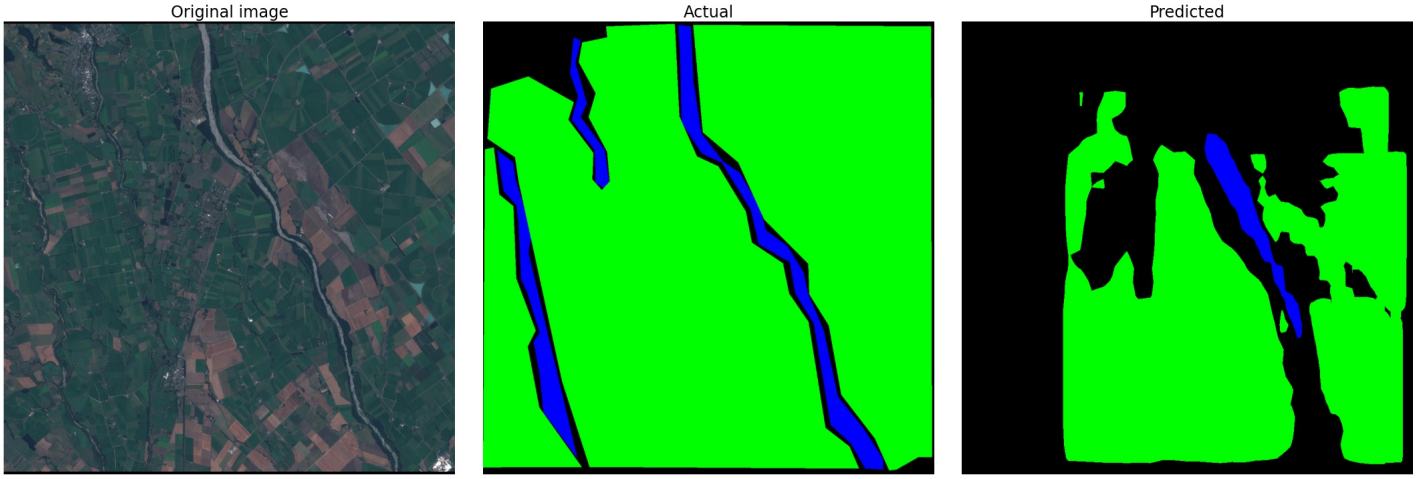
Table 3: Results for Mask R-CNN (with manual annotations)

Since learning rate of 0.001 and 140,000 training steps gave the best performance, this model will be used for further exploration. Table 4 breaks down the performance of this model for each class, averaged across all images. The model performed best on vegetation, which is probably because there is often large regions of this in images that should be relatively easy for the model to detect and then segment. The model performed worst on water, which is probably due to poor predictions around rivers and small water bodies. From visual inspection of the predictions, the model seems capable of detecting offshore regions of water (ie. sea) but is more inconsistent with any onshore regions of water that aren't large lakes.

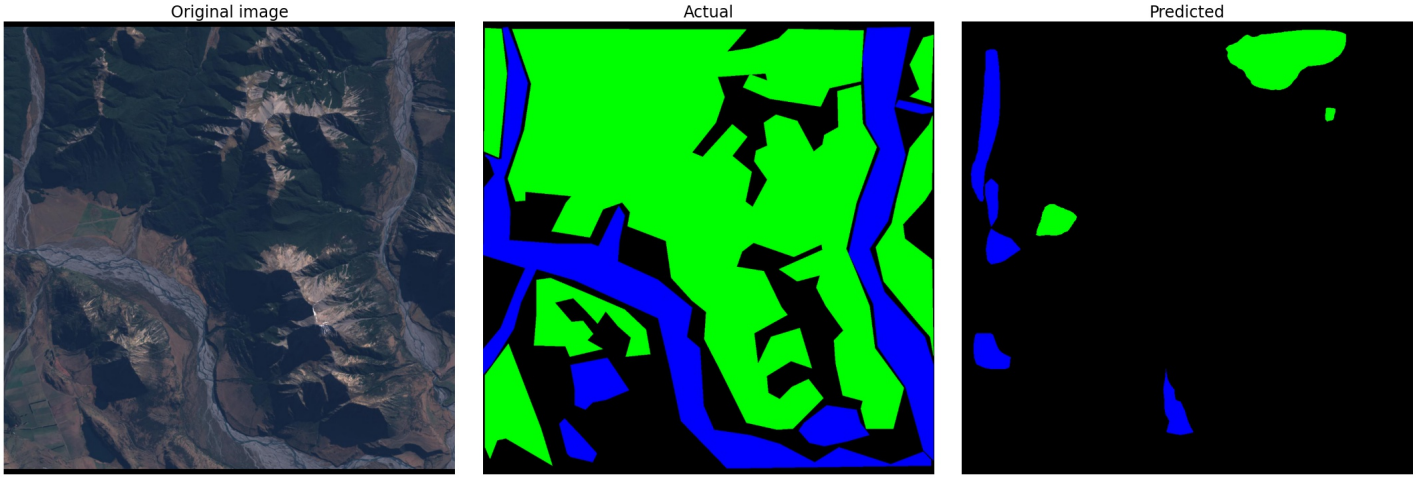
Figures 19 and 20 show some examples of predicted land cover maps using this model. It can be seen that the model is able to correctly capture the locations and shapes of land cover regions for some testing images. However, the region boundaries tend to be very rounded compared to the actual labelling. Also, the predicted regions rarely meet the image boundaries, which is a common problem when using CNNs due to their fundamental design. A remedy to this problem would be to introduce overlap between the satellite images, and then cropping the edges of the images after using the model to get predictions. This was not done in this project as the data collection and processing steps were very time consuming (but is recommended for future projects).

Class	IOU	F1
Background	0.46	0.61
Water	0.34	0.39
Vegetation	0.63	0.73

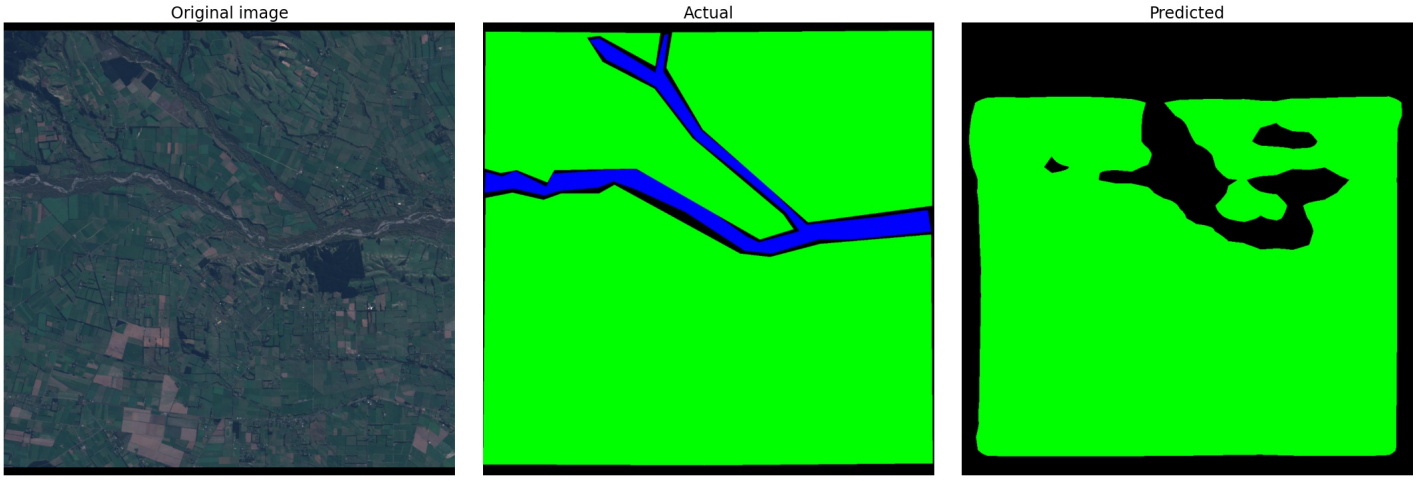
Table 4: Per-class Results for Best Mask R-CNN



(a) Prediction misses most of the rivers and a significant amount of vegetation (mIOU=0.41, mF1=0.57)

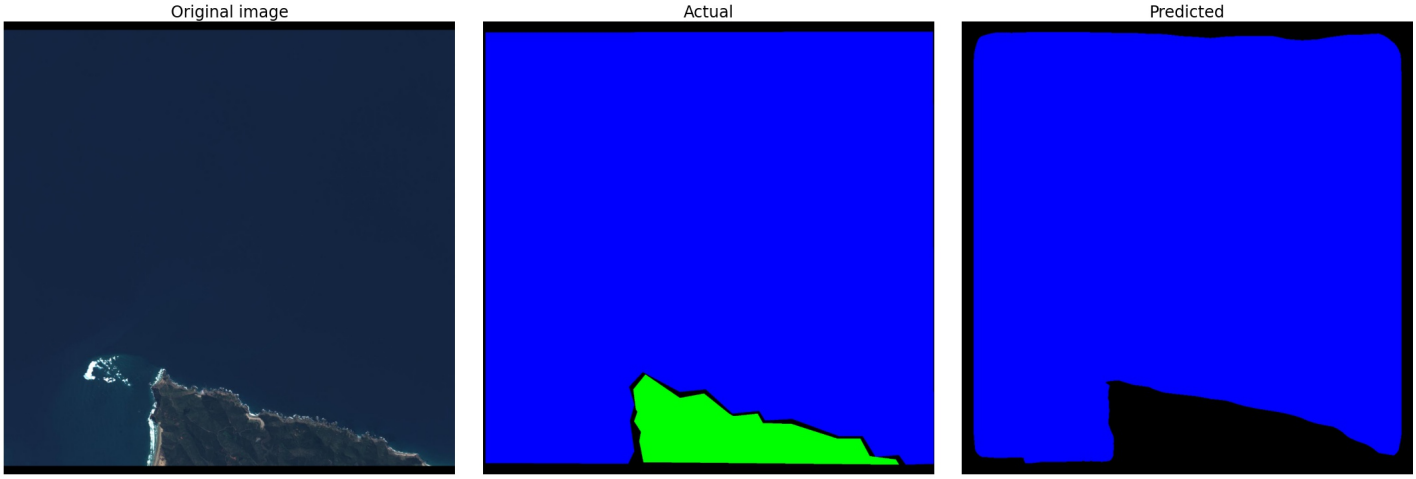


(b) Prediction misses most water and vegetation (mIOU=0.14, mF1=0.23)

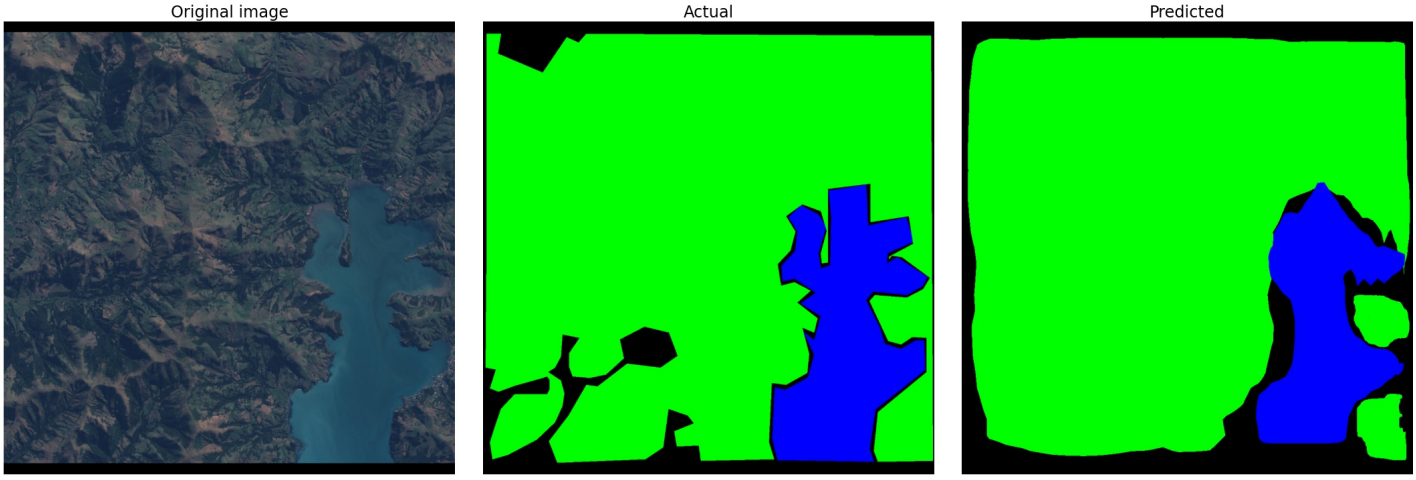


(c) Prediction misses river completely and upper area of vegetation (mIOU=0.61, mF1=0.73)

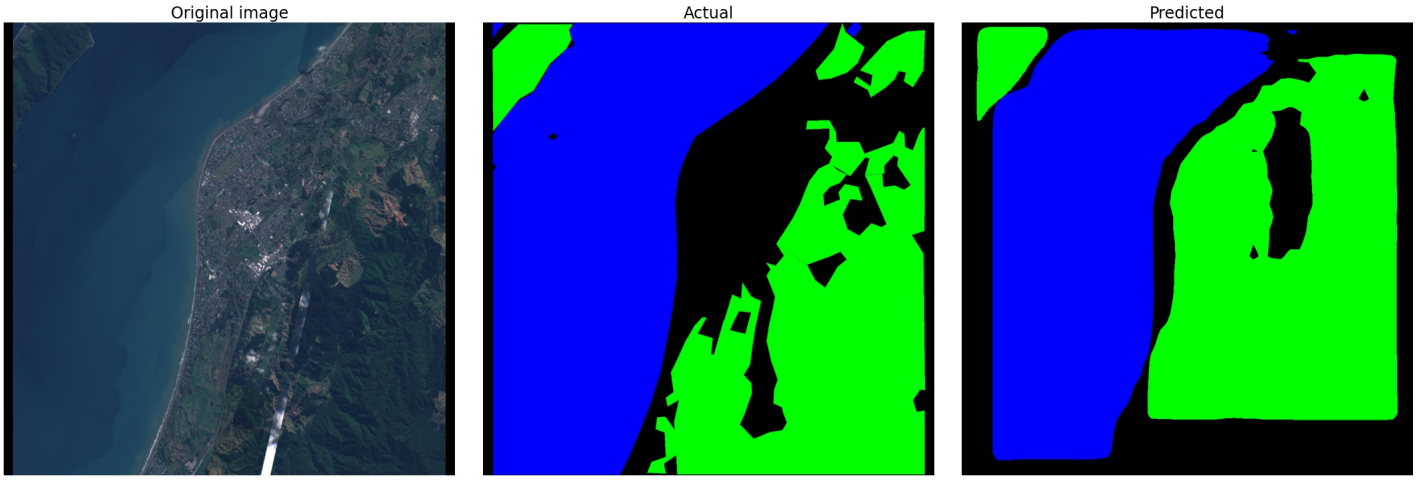
Figure 19: Examples of Predictions from Mask R-CNN model



(a) Ocean predicted almost-perfectly, yet vegetation was not detected (mIOU=0.84, mF1=0.88)



(b) Prediction captures correct locations and shapes, but boundaries are very imprecise (mIOU=0.80, mF1=0.88)



(c) Urban area predicted as vegetation and lower area of vegetation missed (mIOU=0.57, mF1=0.70)

Figure 20: More Examples of Predictions from Mask R-CNN model

5.5 Mask R-CNN with Augmentation

Image augmentation artificially creates training images by applying operations to the existing training images, such as random rotation, flipping, or blurring. This can be used to improve model performance, as it synthetically increases the size of the dataset, thus providing more images for the model to train and test on. From the original train set, 519 augmented images were generated using random left/right flips (30% of images), up/down flips (30%), Gaussian noise (3%), and added clouds (0.1%). After augmentation, there was 1,520 annotated images with 5,865 instances (2,283 water and 3,582 vegetation).

The augmented training of Mask R-CNN with learning rate 0.001, 140,000 training steps, and all other hyperparameters unchanged gave mIOU of 0.71 and mF1 of 0.81 (with training time 8 hours, 57 minutes). Therefore, the augmented model scored lower on the performance metrics than the un-augmented model (mIOU=0.72, mF1=0.82). However, it is extremely unlikely that providing a larger dataset via image augmentation actually worsens the model's predictive performance in general; the lower scores are likely due to variability in the model's performance caused by differences in which order the training images are used. However, it seems there does not seem to be any significant benefit to training the model with augmented images. Different types or amounts of image augmentation could be tried to see if they could improve the predictive performance of Mask R-CNN.

5.6 Deeplabv3+

Initially, the original Tensorflow implementation was attempted to be used (Tensorflow 2021). However, there was little information or examples for training with a dataset other than the two inbuilt benchmark datasets, so this was abandoned in favour of a Keras re-implementation which offered more intuitive/Pythonic access to the model (Zakirov 2020).

Table 5 outlines the preliminary results for the Deeplabv3+ model with a small amount of training. As discussed in Section 5.4, this is likely not enough training to have much confidence in the relative performance with different learning rates, but it does demonstrate that the model was working and there is some variation in results for different hyperparameters. Similar to Mask R-CNN in Table 2, the learning rates of 0.025 and 0.01 give significantly worse results, so are likely too high and should not be used in longer tests.

Learning Rate	Training steps	mIOU	mF1	Training time (min:sec)
0.0001	1,000	0.48	0.53	13:15
0.0005	1,000	0.44	0.46	13:07
0.001	1,000	0.29	0.32	13:07
0.005	1,000	0.44	0.46	13:22
0.01	1,000	0.25	0.28	13:17
0.025	1,000	0.09	0.11	13:17

Table 5: Preliminary Results for Deeplabv3+ (with manual annotations)

Unfortunately, any significant amount of training (ie. 10,000 steps or more) led to the model just predicting a single class for all pixels in all images (which was not always the same class between different runs). An extensive amount of time was spent trying to debug this via tuning of the model's setup/hyperparameters, but the cause of the problem could not be identified or resolved. The debugging included trying various combinations of learning rates (0.000001-0.01), batch sizes (1-6), image dimensions (512×512 - 1024×1024), initial weights (pre-trained and random), loss functions, optimisers, and weight decay values. This means that there is likely something fundamentally wrong with the way the model is being trained, such as the data being interpreted by the model incorrectly, but further tinkering with the initialisation of the model and data-generating process was not able to identify the cause.

5.7 UnDeep Learning: Watershed

When discussing the progress of the project with Stats NZ, they were interested in what approaches could be taken to segment and classify land cover types in a rural satellite image using predictive models that aren't deep neural networks (or machine learning). They suggested that one of these approaches could be tried to quantify the predictive benefits that the deep network models provide for this problem, relative to a non-machine learning baseline. The main approach to image segmentation that does not include deep networks is the Watershed algorithm with K-means clustering, so this model was briefly explored.

The Watershed algorithm treats an image as a topographic map, where the brightness of each pixel represents its height, and constructs segment-boundaries as the ridgelines along the pixel heights. This means that a true-colour image must be converted to an image where the brightness of each pixel represents the models confidence that the pixel is a region. The implementation in this project did not distinguish between classes when transforming the true-colour satellite images and produced a class-agnostic segmentation map. The steps performed by model were:

1. Convert satellite image to grayscale.

2. Apply an adaptive Gaussian threshold to separate pixels into foreground and background areas.

3. Noise removal. Apply an opening operation (erosion followed by dilation) to remove small black holes (background) inside the white regions (foreground). Apply a closing operation (dilation followed by erosion) so that the shape of the white regions (foreground) are not distorted.

4. Apply Euclidean distance transform to get border between bordering objects, and extract locations of the local maxima.

5. Use K-means algorithm to get a small (20-30) number of cluster centres from the local maxima. This clusters the local maxima together, which reduces the number of resulting regions.

6. Apply the Watershed transformation.

The process is currently performing unsupervised learning, which means that it does not using training images or annotations to inform the model and so only utilises the information in each image to generate segmentation predictions. Further work would be required to determine how the implementation could be adapted to give class predictions for each segment/region and how to utilise the training information to inform the predictions.

Figure 21 includes some the Watershed predictions for three satellite images. In (a), the land and sea are well separated in the segmentation mask, but this is not a very difficult satellite image to segment since the land and water are easily distinguishable and the land is almost completely vegetated. The prediction in (c) also includes land and sea, but the model was only able to isolate one region of land (island) and essentially classified the rest of the image as one region, despite it including a mixture of water and land. The prediction in (b) is representative of most of the outputs seen during testing; the model is able to separate the onshore and offshore regions, but the model seems to generate meaningless onshore regions, mostly fail to separate rivers and land, and mostly fail to separate vegetated and unvegetated (urban or barren) land.

Work to improve this model would include determining how to adapt the process to recognise class labels, and subsequently predict class labels. For example, this could be achieved by applying a final step to the predicted segmentation mask, which predicts per-region class labels by comparing the colour distribution in each region to the colour distribution of the classes in the training images (and then use some linear bounds/thresholds). This might improve the segmentations, since the model would be better informed about what constitutes background vs foreground (labelled regions). Work could be done to investigate how many clusters in the K-means algorithm gives the best result. However, if the model predicted class labels, then neighbouring instances of the same land cover type could be merged, which

would simplify the output segmentation mask and might reduce the importance of the preset number of clusters. Also, the noise removal step could be modified to be better suited to this problem, but this seems likely to only offer modest predictive improvements. Despite all of these adaptions and improvements, the Watershed model would be expected to still provide worse predictions than the deep networks.

5.8 Mask R-CNN V2.0

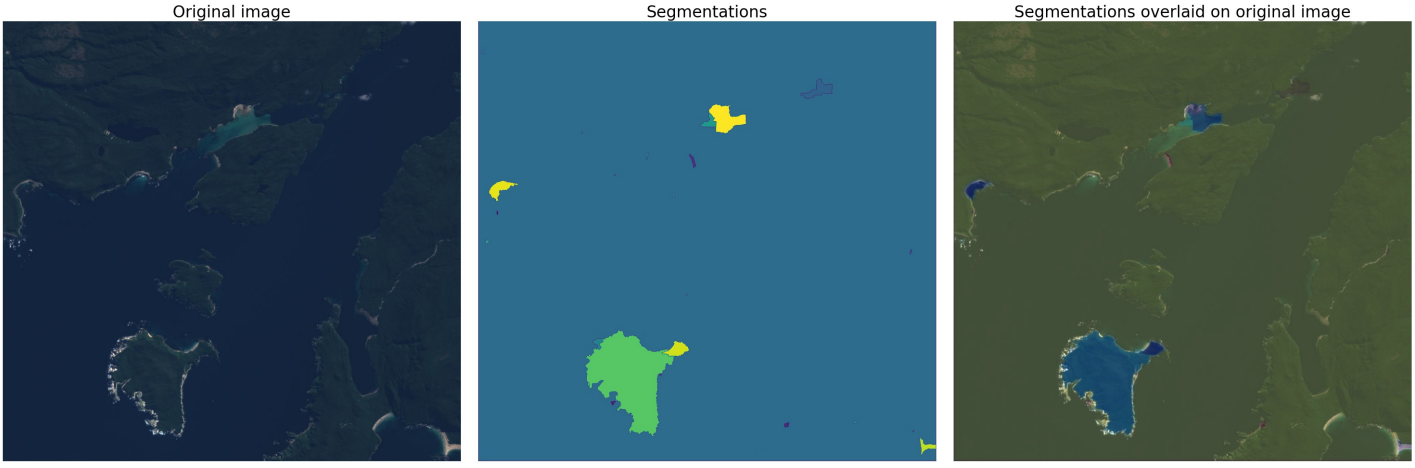
This is the results section for using the Mask R-CNN model with the annotations generated from the Manaaki Whenua land cover database. Since this included annotations for all 1572 images, and the manual annotations only included 1001 images, the training and validation image sets are not entirely equivalent, but the data splitting process attempted to keep these train/val sets as similar as possible.

This labelling set is expected to be reliable (better defined), as it was developed by land cover experts. However, the land cover regions tend to be smaller (partly due to there being more classes) which will mean that it is harder for any deep neural network (or other model) to learn about what indicates the different land cover types. This is because there is less pixels, and therefore less information, associated with each instance of land cover. Therefore, it was expected that the Mask R-CNN would need to be trained for more steps to converge to a set of parameters that (hopefully) give a reasonable predictive performance.

Results were not able to be obtained for this labelling set (in time for this report) since the model was freezing during training, but not throwing any errors. More time will need to be spent understanding why this could be, but it seems likely to do computer memory issues, although usually this would halt the training and inform the user via errors.

(a) Good Prediction

(b) Average Prediction



(c) Bad Prediction

Figure 21: Examples of Predictions from Watershed model

6 Discussion

6.1 Challenges

Some background tasks and unforeseen challenges:

- Deeplab in general. Difficult to understand how to train the original Tensorflow implementation on a custom dataset. Difficult to understand what was going wrong with the (no-longer maintained) Keras implementation.
- Mask R-CNN uses Tensorflow 1 whereas Deeplab uses Tensorflow 2, so they were required to be run in different Conda environments which led to time spent resolving package dependencies issues so that they could use the same metrics.
- Invalid-index errors thrown by the model, which were caused by the VIA website creating regions/annotations that were beyond the dimensions of the JPEG images, which required creating a Python script to push regions within the boundaries after exporting the JSON file from VIA.
- When manually annotating the images, distinguishing between dark vegetation and shaded unvegetated surface was not possible. Identifying most lakes and some rivers required referring to a separate map using coordinates.
- Mask R-CNN is a lot slower to train using the Manaaki Whenua annotations due to more classes and many more instances in each image. This training also froze without warning, so predictions/results were not able to be obtained.

6.2 Research question

This project did provide proof that deep networks are able locate and distinguish land cover types in satellite images of rural New Zealand. However, further work would be required to get the model to a stage where it is reliably identifying/predicting the boundary of land cover types, and to adapt the collection of Python scripts and Jupyter notebooks into a more coherent, robust system that can autonomously train the model on a larger dataset.

6.3 Further Work

The main recommended further work is using different types of deep networks. The obvious first step would to get Deeplab working. It is intuitive to use a model specifically designed for semantic segmentation, such as Deeplab or HRNet (Hierarchical Multi-Scale Attention), rather than adapting the output ad hoc from a model

designed for instance segmentation (Mask R-CNN). It seems likely that semantic models would provide better predictions for this problem/project. Alternatively, the CNN used in Mask R-CNN can be changed to be more or less complex, and provides a way to test the predictions from a slightly different deep network.

The fundamental way to improve the predictive performance of the deep networks is to provide more information in the training (and testing) datasets. To increase the resolution of the images and still cover the same physical area (ie. increase spatial resolution) would require paying for imagery from a different satellite. However, more information could be retrieved from the Sentinel satellites and sentinelhub API by requesting more electromagnetic waves than just the true-colour RGB bands, such as Near Infrared (NIR) or Short-Wave Infrared (SWIR), as these may improve the ability of the model to map shorelines and distinguish vegetation types. In any future data collection, overlap should be introduced into the satellite imagery to prevent poor predictions around image boundaries (by cropping the predicted segmentation). Another option is to retrieve Sentinel imagery of the same areas over different time periods, which would increase the size of the dataset and introduce more seasonal variety. This which may lead to a more robust model able to predict land cover for images from any time in the year, but it might also cause new problems, such as more clouds and more snow cover.

The second main recommended further work is getting the Mask R-CNN model to work with the Manaaki Whenua annotations. It is not clear what is causing the model to freeze during training so this would require general tinkering with the configuration and setup of the model. Accurate predictions for seven classes would be a lot more useful for Stats NZ, as this would provide the level of specificity required for real-world applications.

Another way to improve the performance of Mask R-CNN (and other deep networks) would be to spend more time fine-tuning the hyperparameters, such as batch size, weight decay regularisation, and learning momentum. Additionally, further testing of the image augmentation may reveal there is a specific type or amount of image processing which provides more informative synthetic data, thus improving model performance. However, this would likely provide less drastic improvements in the predictions than sourcing additional or higher-resolution satellite imagery.

This project aimed to test the predictive performance of deep networks. A train/-val split was used to obtain an estimate how well the model performed on unseen/testing data, but this was only a single pair of train/val sets. A better estimate of the networks predictive performance would be obtained by doing multiple train/val splits, such as k-fold cross-validation, but this would require training the model and generating the model multiple times (typically 5 or 10 times), so would significantly increase the time required to get results (by about 5 or 10 times).

SAIL and Stats NZ also discussed saving the model to AWS so that the model could provide predictions for arbitrary images provided to it via a web interface. This is more of a software engineering problem and does not contribute to improved

predictions of land cover types, but it would improve the approachability and utility of the work completed in this project (and related projects).

References

Aplin, Paul (2004). "Remote sensing: land cover". In: *Progress in Physical Geography* 28.2, pp. 283–293.

Arlen, Timothy (2020). *Understanding the mAP Evaluation Metric for Object Detection.* URL: `https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3` (visited on 01/27/2021).

Berner, Sarah (2020). *The Math Worked Out: Neural Networks.* URL: `https://medium.com/@berner.sarah/the-math-worked-out-neural-networks-5cd4229ed56c` (visited on 01/19/2021).

Chen, Liang-Chieh, George Papandreou, et al. (2017). "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4, pp. 834–848.

Chen, Liang-Chieh, Yukun Zhu, et al. (2018). "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 801–818.

Dumoulin, Vincent and Francesco Visin (2016). "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285*.

Girshick, Ross (2015). "Fast R-CNN". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

Girshick, Ross et al. (2015). "Region-based Convolutional Networks for Accurate Object Detection and Segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 38.1, pp. 142–158.

He, Kaiming, Georgia Gkioxari, et al. (2017). "Mask R-CNN". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.

He, Kaiming, Xiangyu Zhang, et al. (2015). "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 37.9, pp. 1904–1916.

Herold, Martin, Joseph Scepan, and Keith C Clarke (2002). "The use of remote sensing and landscape metrics to describe structures and changes in urban land uses". In: *Environment and Planning A* 34.8, pp. 1443–1458.

Kastens, Jude H and Davld R Legates (2002). "Time series remote sensing of landscape-vegetation interactions in the southern Great Plains". In: *Photogrammetric Engineering & Remote Sensing* 68.10, pp. 1021–1030.

Keskar, Nitish Shirish and Richard Socher (2017). "Improving generalization performance by switching from adam to sgd". In: *arXiv preprint arXiv:1712.07628*.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kussul, Nataliia et al. (2017). "Deep learning classification of land cover and crop types using remote sensing data". In: *IEEE Geoscience and Remote Sensing Letters* 14.5, pp. 778–782.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Li, Ethan Yanjia (2020). *Witnessing the Progression in Semantic Segmentation: DeepLab Series from V1 to V3+*. URL: https://towardsdatascience.com/witnessing-the-progression-in-semantic-segmentation-deeplab-series-from-v1-to-v3-4f1dd0899e6e (visited on 02/09/2021).

Lin, Tsung-Yi et al. (2014). "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer, pp. 740–755.

Matterport (2021). *Mask R-CNN for Object Detection and Segmentation*. URL: https://github.com/matterport/Mask_RCNN (visited on 02/01/2021).

Monteux, Angelo (2020). *Metrics for Semantic Segmentation*. URL: https://ilmonteux.github.io/2019/05/10/segmentation-metrics.html (visited on 01/27/2021).

Papers with Code (2021). *Semantic Segmentation*. URL: https://paperswithcode.com/task/semantic-segmentation (visited on 02/01/2021).

Ren, Shaoqing et al. (2016). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE transactions on pattern analysis and machine intelligence* 39.6, pp. 1137–1149.

Scott, Grant J et al. (2017). "Training deep convolutional neural networks for land–cover classification of high-resolution imagery". In: *IEEE Geoscience and Remote Sensing Letters* 14.4, pp. 549–553.

Tensorflow (2021). *DeepLab: Deep Labelling for Semantic Image Segmentation*. URL: https://github.com/tensorflow/models/tree/master/research/deeplab (visited on 02/01/2021).

Tiu, Ekin (2019). *Metrics to Evaluate your Semantic Segmentation Model*. URL: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2 (visited on 02/01/2021).

Wang, Xu (2008). "Method of steepest descent and its applications". In: *IEEE Microwave and Wireless Components Letters* 12, pp. 24–26.

Yen, Li (2018). *A Summary of Neural Network Layers*. URL: https://medium.com/machine-learning-for-li/different-convolutional-layers-43dc146f4d0e (visited on 02/09/2021).

Zakirov, Emil (2020). *Keras implementation of Deeplabv3+*. URL: https://github.com/bonlime/keras-deeplab-v3-plus (visited on 02/01/2021).