Building Segmentation from Satellite Images with Deep Neural Networks

Matthew McCarrison

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2019

Abstract

There has been a boom in the deep learning space due to the amount of labelled data that is currently being generated. This report investigates the use of deep learning in aerial imagery data, which is being captured every single day by many powerful satellites. Only recently has the data started to become publicly available. To exploit this data, Convolutional Neural Networks (CNNs) can be used to automatically extract the location of buildings from these images. This annotation can be useful for describing scenes after natural disasters or for more mundane applications such as urban planning. The model in this report is supervised and uses satellite data from the SpaceNet dataset which consists of a raw image alongside a JSON file. The JSON file describes the shape of the building labels. A regularisation technique is also implemented, with the use of an autoencoder, to improve the model beyond a baseline CNN and produce state of the art accuracy scores.

Acknowledgements

I would like to thank my supervisor, Dr. Hakan Bilen, for his assistance throughout the project and willingness to guide my understanding in a variety of topics. His knowledge and resources have been instrumental in the completion of this project. Additionally, thanks to Taha Koçyiğit for his help.

I would also like to thank my family and friends for the support and motivation they have offered me during my time in Edinburgh, especially during my final year.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Satellite images have been captured for a myriad of reasons and have been extremely important for weather forecasting and intelligence amongst other things. By combining satellite imagery and deep learning, automatic semantic segmentation can be achieved on aerial data. By producing a segmentation of the objects in these satellite images, many images can be analysed and described almost instantaneously.

In this case, by segmenting buildings from satellite images it is possible to produce masks (see Figure 1.1) displaying all the buildings present in an image. Identifying the buildings in a satellite image would allow for maps containing building footprints to be more easily produced. Also, in response to a disaster, a satellite image could be taken and the damage could be instantly analysed across a region. There are additional uses, such as town/urban planning, that require this kind of information.

Aerial imagery is being produced by satellites constantly [1]. In April 2018, there were 1,980 operational satellites orbiting the earth. Out of this number, 684 of the satellites were used for earth observation/imagery purposes. Given the large number of satellites, data is constantly being provided for scientific use. This offers huge opportunities in the deep learning domain, given that an abundance of data is vital to produce high quality predictions.

1.2 Overall Aim

The aim of this project is to implement a Fully Convolutional Network (FCN) [2] for object segmentation on satellite images. For background information on FCNs, read section 2.2.1. The FCN performance will be evaluated by measuring the Intersection over Union (IoU) score between the predicted segmentation masks and the true masks for the buildings. After this, an autoencoder will be trained and the decoder will be incorporated into the segmentation network in an attempt to improve the accuracy of

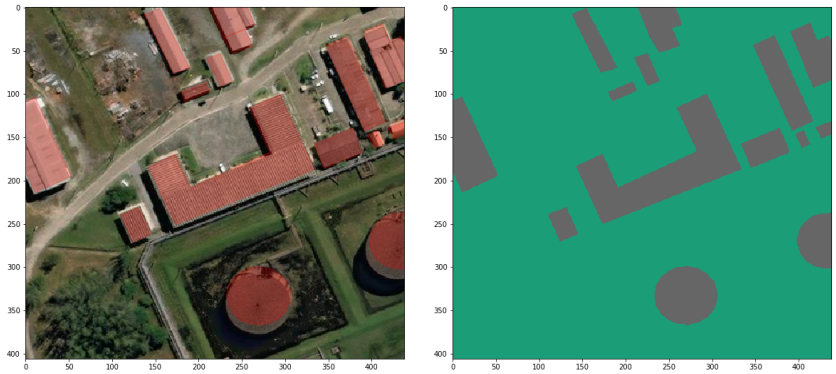


Figure 1.1: An example of a building footprint mask generated using the polygon training data from the SpaceNet dataset.

the model. This idea has been proposed and been successful on other computer vision tasks [3]. It relies on the fact that an autoencoder will learn latent features that are unlikely to be successfully handcrafted. The decoder will act as an additional branch in the segmentation network and will force the network to have sound justifications for its decisions. More information on autoencoders can be found in Section 2.3. The training data will consist of satellite images and building labels across the Las Vegas, USA area.

1.3 Implementation

The implementation of the models and the data analysis was all written in Python (version 3). The deep learning framework that was used is PyTorch [4]. Bash scripts were written for the Python experiments and to carry out the grid searches across the hyperparameters. The experiments were run on a University of Edinburgh Informatics server called `vico03`, that utilises 4 x NVIDIA GeForce GTX 1080Ti GPUs.

1.4 Objectives

By the end of the project, I hope to have satisfied the following objectives:

- Generate masks of building footprints from the SpaceNet training data by combining GeoTIFF and GeoJSON files. The combination of these files will produce a single image mask, akin to Figure 1.1. These masks are necessary for training the network and so that masks can be output by the model, when a raw satellite image is inputted.
- Implement and experiment with a segmentation network. In this case, a Fully Convolutional Network (FCN) [2]. This network has proved to produce good results for semantic segmentation and will be used in this report. This model will be analysed and act as a baseline for further experimentation.

- Create an autoencoder for the building masks. This will be necessary to extract and connect the decoder to the segmentation network.
- Train and tune a network combining the FCN and the decoder part of the autoencoder, then observe and analyse accuracy improvements over the baseline FCN. The decoder will offer an additional information path for the network ensuring the images model the building structure accurately. Analysis will be carried out into the importance of the additional decoder branch and whether the added complexity is worth it.

1.5 Structure of Report

This report can be viewed in 3 main sections:

- **Background and Understanding**
  In Chapter 2, background information about Convolution Neural Networks, Autoencoders, Semantic Segmentation among other things, are introduced to support the readers knowledge for the subsequent chapters. In Chapter 3, some of the current research related to this project is explored as well as further explanation into some of the techniques that will be used. In Chapter 4, information about the SpaceNet dataset is introduced as well as some of the metrics that will be used when evaluating the final models.
- **Experiments and Implementation**
  In Chapter 5, some initial experiments are carried out and investigated to evaluate the effect of preprocessing the data before feeding it into the models. In Chapter 6, a more thorough description of the architecture of the models used in this report is provided. Also, some initial experiments that lead to these architectures, are reported. In Chapter 7, the final experiments are carried out from the baseline up to the regularised CNN. The results from these experiments are reported using a variety of tables and figures.
- **Analysis and Conclusion**
  In both Chapter 7 and 8, an analysis is carried out based on the results from experiments in this report. The objectives and ideas that are mentioned will be critically evaluated. In the future work section, some methods and models will also be introduced that may have the potential to improve results further.

Chapter 2

Background

In this report, a range of topics will be covered, such as Fully Convolutional Networks and Autoencoders. To aid the reader, I have included background knowledge on these topics below.

2.1 Semantic Segmentation

Semantic segmentation is a major problem in regards to computer vision [5]. The goal is to understand the parts of an image at pixel level, knowing what object(s) are in the image and where in the image the object(s) are. This can be seen more clearly in Figure 2.1. In regards to this project, there will only be two classes, building and not building. In doing this, fine-grained inference and a classification for every pixel in the image (a dense prediction), will be achieved. By having to produce class predictions at a pixel level, it becomes a much more complex problem than simple classification i.e. one prediction per image (e.g. this image contains a dog or a cat). After a dense prediction is made, post-processing can be used to generate predicted bounding boxes/building footprints for the satellite image.

Semantic segmentation does not distinguish between objects of the same category. For example, with buildings, the segmentation map will only show whether there is a building or not. It will not individually classify the instances of of each object type e.g. building 1, building 2 etc. To do this, an instance segmentation model would be necessary.

2.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of supervised artificial neural network that is frequently used for image processing. Specifically, CNNs allow images to be processed on a per-pixel, proving to work very well in practice [6]. It is super-

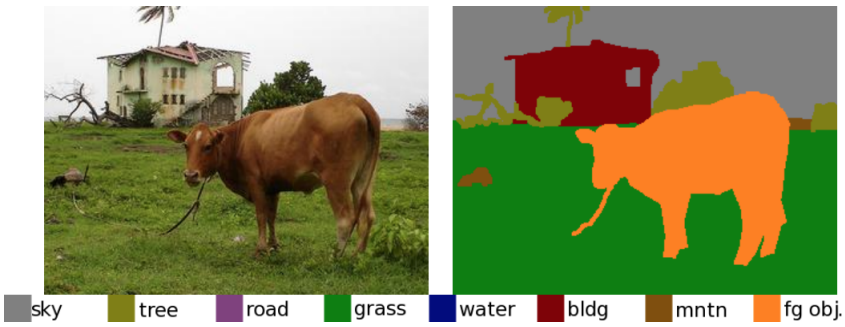


Figure 2.1: An example of semantic segmentation.

vised because the network is trained using both input images (raw satellite images) and labels (ground truth binary masks).

A CNN is an architecture that is often used in simple classification tasks, where a single class label would be given for each image e.g. cat or dog. However, they are also widely used for image segmentation tasks. A CNN is built from an input layer, some intermediate hidden layers and an output layer. The hidden layer usually consists of multiple types of layer such as convolutional, max pooling, fully-connected and up-sampling layers. The network will learn the filters that would otherwise have to be designed by a person, removing the need for feature engineering. This independence from human interaction/knowledge as well as an increase in efficiency, makes CNNs very advantageous.

2.2.1 Fully Convolutional Network

A Fully Convolutional Network (FCN) [2] is a type of Convolutional Neural Network (CNN) that was specifically designed for semantic segmentation tasks (see Figure 2.2). It is built from only locally connected layers such as convolutional, max pooling and up-sampling layers. Therefore, a FCN does not use fully-connected layers or multi-layer perceptrons. In a CNN, these are usually found at the end of the network to make one classification decision for a whole image. However, for semantic segmentation, the goal is to make predictions on a per-pixel basis. Therefore, the fact that fully-convolutional layers discard local spatial information in favour of a global predictions, makes them unsuitable in this problem domain.

The input and labels are both 3D images, where the dimensions are C x H x W. C is the channel dimension and H and W are the spatial dimensions, corresponding to height and width. In the label space, the color channel C, is 1 because it is a black and white mask. For the raw satellite images, the number of channels C is 3. The output of the segmentation network is a binary mask with the same dimensions as in the label space.

To obtain an output, the network must make predictions on a per pixel basis resulting in a segmentation map/mask. The FCN has two paths, an up-sampling path and a down-sampling path. The down-sampling path is used to extract semantic information from the whole picture, while the up-sampling path is used to produce precise localisation

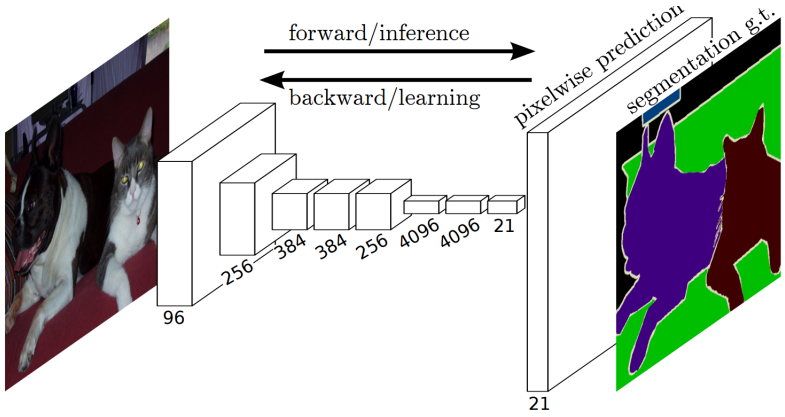


Figure 2.2: A Fully Convolutional Network architecture [2].

of the predictions. In the up-sampling path, skip connections can be used. These skip connections are only used in variants of the original FCN e.g. FCN-8s and FCN-16s, described further in Section 6.1. Skip connections allow for the recovery of fine-grained spatial information that is often lost in max pooling/down-sampling. The skip connection bypasses one or more layers, which allows merging/adding of the context (from down-sampling) and the spatial information (from up-sampling).

2.2.2 Convolutional Layer

A convolutional layer consists of filters/kernels and receives some input and bias (in recent years the bias has been removed the calculation). The data can propagate forward or backwards through the layer. In the forward pass, the kernel (see Figure 2.3) is convolved across the input. The kernel slides across the width and height of the input volume computing a dot product between the kernel values and the values of the input at that point (see Figure 2.3). The bias is then added to the result. This calculation produces an activation map of that kernel. The network then learns that these kernels activate when certain visual features are observed (such as an edge) in a certain position in the input space. The output volume consists of these activation maps stacked along the depth dimension.

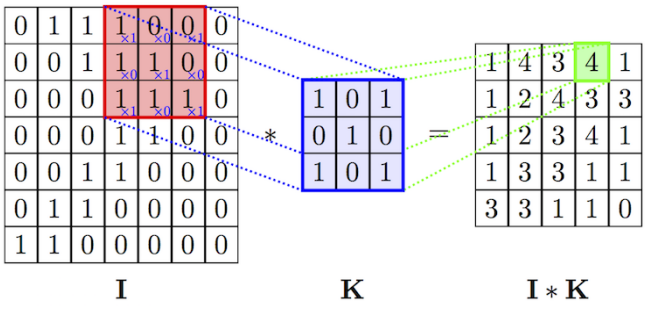


Figure 2.3: A filter/kernel (K) sliding across the input (I), producing the output (the bias is added to the result of the dot product).

2.2.3 Pooling Layer

A pooling layer can be used to reduce the spatial dimensions of a network. It is a form of down-sampling that doesn't affect the depth dimension. Pooling is used to achieve invariance to image transformations, improve computational performance of a network, to reduce the number of parameters (reducing overfitting) and provide robustness to noise and clutter. In a similar way to the convolutional layer, there is a window that slides across the inputs (see Figure 2.4). However, in this case, some function is applied to the window and the result is passed to the corresponding output location.

There are various types of pooling that can be used e.g. max pooling, where the maximum value in the window is the value used in the output. There is also an average pooling function which takes the average of the window values. Pooling layers are often used between convolutional layers. These functions are better explained by the example in Figure 2.4.

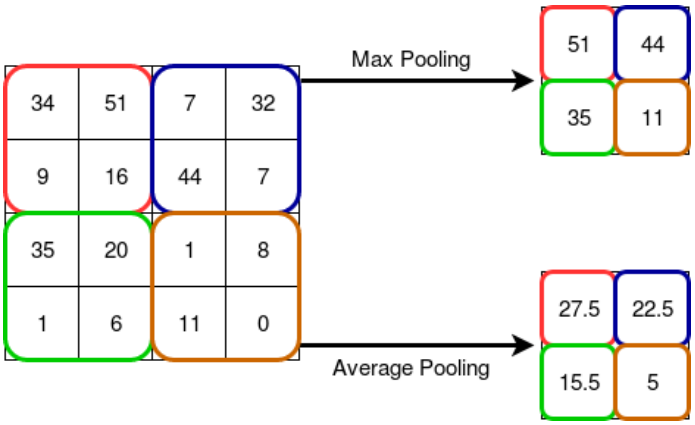


Figure 2.4: A 2x2 pooling window sliding across the input, producing a down-sampled output for both max and average pooling.

2.2.4 Upsampling layers

Both convolutional layers and max pooling layers work to reduce the dimensions of the image/previous layer. However, to produce a binary mask that is the same dimensions as the input, it is necessary to use upsampling on the compressed representation produced by downsampling.

Upsampling can be carried out using a variety of techniques. One popular technique is to use simple bilinear interpolation. Another is to use a deconvolutional layer, a deconvolution is not the opposite of a convolution and is often better known as a transposed convolution because it is actually the transpose (gradient) of a convolution.

Stride is applied at the (de)convolutional layer, instead of the kernel convolving across the input pixel by pixel, the stride factor can be set to more than 1. For example, if the stride is set to 2, the kernel would move 2 pixels across the image at each step, instead of 1. Therefore, stride can be used for upsampling with a deconvolutional layer

but also downsampling with a convolutional layer. Upsampling can also be done using unpooling, which is used to revert the effects of a max pooling layer. The unpooling layer works by remembering the location of the maxima for each window and copying the value to this exact location.

2.3 Autoencoder

An autoencoder is a type of unsupervised artificial neural network in which input data is compressed into some intermediate representation (or code). It is unsupervised because it requires no label information. It simply takes some data as input (in this case, a binary mask image) and attempts to recreate it after some compression. This representation is then uncompressed in an attempt to reconstruct the original data, hence providing some dimensionality reduction. The autoencoder method is an unsupervised technique that allows the structure of the input data to be learned and then leveraged when passing the input through the autoencoder's bottleneck (see Figure 2.5).

The weights of the autoencoder are updated by comparing the input against the output, and calculating the loss. In this case, since binary masks are fed into the network, the cross-entropy loss function can be used. The global aim of the autoencoder is to minimise the loss. At each iteration the values for the weights should be updated in an attempt to reduce this loss. The loss function is as follows:

$$L(x,y) = -\sum_{i=1}^{d_x} x_i \log(y_i) + (1-x_i)\log(1-y_i)$$

where x is the input data and y is the reconstruction.

As with a CNN/FCN, initially there are some random weights. These random weights are then updated by calculating a gradient in order to minimise the loss. This is done via backpropagation. The errors at the output are propagated back through the network so that the weights can be updated accordingly. This will hopefully produce a better model that will more accurately reconstruct the input. The weight updates are repeated many times and the loss should repeatedly get smaller until a certain point at which the loss function has been minimised.

The autoencoder used in the implementation of this project, will be a Convolutional Autoencoder (CAE), which typically consist of convolutional layers and max pooling layers (as described in Section 2.2.2 and Section 2.2.3, respectively). A CAE is much more appropriate for a pixel based approach. When designing the architecture of the network, it is important that enough dimension reduction is carried out so that the autoencoder can learn important latent features. If the network was able to simply recreate the image by having too much capacity, it would not learn anything useful about the underlying features of the image - even a simple linear autoencoder could produce an identical version of the image, acting as the identity function.

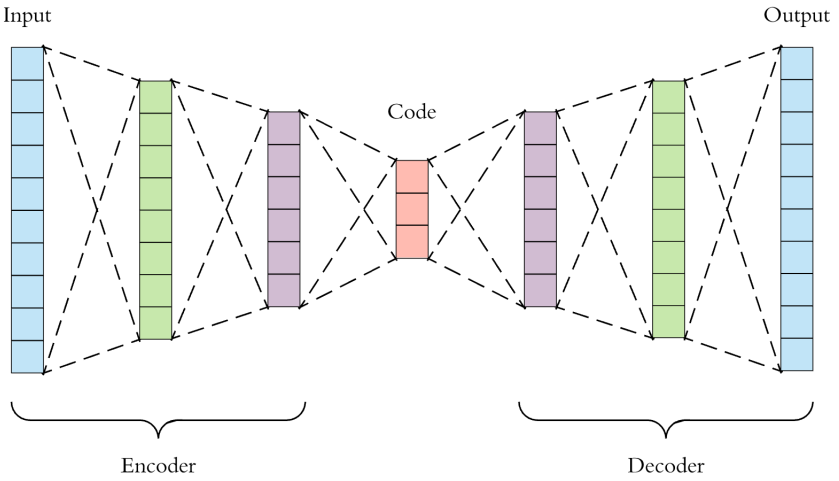


Figure 2.5: An example of an autoencoder, showing the encoder and decoder stage, with 5 fully connected hidden layers [7].

2.4 Regularisation

In the final FCN model, the autoencoder and FCN will be combined to try and improve on results from a standalone FCN, this is explained more thoroughly in Section 6.3. The use of the decoder as an auxiliary branch into the FCN can be thought of as a regularisation technique.

Regularisation is a technique used to prevent overfitting in the training data. If the model overfits to the training data then it will be less useful on the test data - the goal is to have a model that generalises well to any future data that it may receive. As mentioned in Section 2.3, loss functions are used to calculate the weight updates. The decoder provides an additional loss to the FCN on top of its standard loss function. This extra loss can be seen as regularising or penalising the main loss in the FCN. By including the decoder into the pipeline, it ensures the network has sound justifications for its weight updates and doesn't learn too much from the background noise in the data.

2.5 Structured Prediction

Structured prediction differs from typical machine learning problems such as classification and regression. The goal is to predict objects containing some internal structure [8]. Structured prediction is used for tasks with a more complex output (e.g. graph, sequence) than simple binary classification (yes/no) or regression, which often makes it applicable in the computer vision space. Instead of the model getting some input data x and predicting some class y, in structured prediction, the model will use the data x, to find the best structure y that is related to that data x. Therefore, in this task, the model will take the training data to predict the structure/large output space of the corresponding binary mask.

To combat a random scattering of pixel classifications and produce connected regions of the image where pixels have the same classification as their neighbours, there is a need to make use of some sort of post-processing technique to 'smooth' the image.

Statistical approaches are used such as Conditional Random Fields (CRFs) - which refine the labelling in the segmentation map using pixel similarities. This technique allows the removal of noise from the final segmentation map and refine in and around building boundaries [9]. This is how CRFs work at a high-level, the mathematics will not be described in this report. In some instances there has been success in combining CRFs, at the post-processing stage, with deep learning for semantic segmentation [10].

Chapter 3

Related Work

A similar building segmentation task has been attempted for the SpaceNet buildings challenge - albeit on a different dataset (Rio De Janeiro). Most of the solutions were based on random forest approaches [11]. The work done in that challenge used a different evaluation metric, F1 score, and so it is difficult to make definitive comparisons between their approaches and the work in this report. Random forest approaches can work well for certain tasks in the computer vision field, such as satellite imagery. However, using Convolutional Neural Networks, requires no feature engineering and allows the model to make its own assumptions. This, in theory, should provide better results than work using random forests. There has been work in the aerial imagery space that provide conclusive results that CNNs will boost accuracy over a random forest model [12].

However, in this work, more modern approaches will be taken, inspired by the latest research in the computer vision space. The idea of combining a FCN and an autoencoder originates from a paper that uses a custom regularisation function [3]. The results from this paper are impressive on the ImageNet dataset [13] and it would be interesting to experiment with the effectiveness of a similar approach in relation to satellite imagery.

This approach involves training an autoencoder on the ground-truth label maps i.e. the binary masks (as in Figure 4.2). As by the definition of an autoencoder, the binary masks are forced through a bottleneck in an attempt to reproduce the input, at the output stage. The code at the bottleneck will hopefully learn latent relationships that are not obvious to humans.

The general architecture of the network used in the paper [3] can be seen in Figure 3.1. The regulariser is implemented by connecting a CNN to the decoder part of the autoencoder. During the training phase, the parameters for the decoder are frozen. Therefore, the backpropagation of losses only effect the FCN/CNN parameter updates. This architecture ensures that any useful abstractions that the autoencoder learns also have to be worked through the FCN. This forces the FCN to have some high level justification for its predictions.

There are two phases when training a network in this way. The first phase learns an autoencoder that models the structure of the labelled binary masks. The second

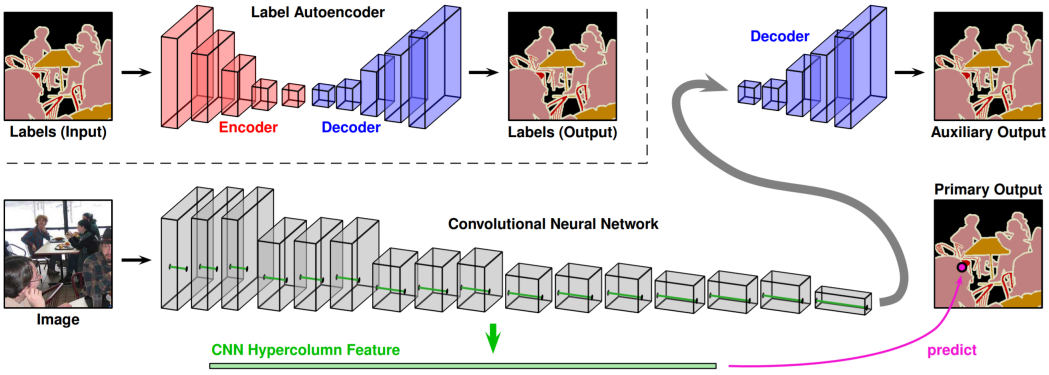


Figure 3.1: The architecture of the network used in the custom regularisation paper by Mostajabi [3].

phase uses an auxiliary task of predicting the output using the decoder part of the autoencoder. This auxiliary branch is viewed as a training regulariser and it can be discarded after training, therefore, there is no cost at test time.

Usually in recognition tasks, the output from the last layer of the CNN is used to make predictions. However, as this task is much more fine-grained, it requires a different approach. An approach is needed where the last layer is not invariant to things like precise location. A way of changing the architecture that is used in many reports, is to use a hypercolumn representation [14]. The paper defines a hypercolumn as: "a hypercolumn at a given input location is the outputs of all units above that location at all layers of the CNN, stacked into one vector.". The label for a pixel is then predicted using a per-pixel concatenation of all the CNN layers (see Figure 3.2).

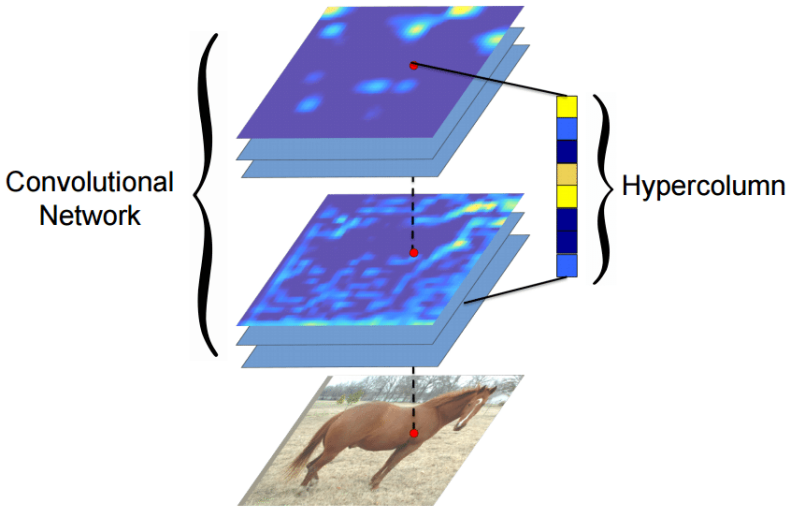


Figure 3.2: The hypercolumn representation. The bottom image is the input and above it are the feature maps of different layers in the CNN. The hypercolumn at a pixel is the vector of activations of all units that lie above that pixel [14].

Chapter 4

Dataset

For this project, the SpaceNet dataset was used [15]. In particular, there was a focus on the Las Vegas buildings dataset because of its abundance of labels and variance in type of building. It provides both city centre landscape as well as suburban and rural areas, allowing the network to learn a variety of different building types, in a developed area.

The dataset consists of images/tiles that cover 200x200m of land, in the Las Vegas area. In total there are 3851 of these images, each with corresponding polygon building labels. The data is split into the training set (60%), validation set (20%) and test set (20%). The tiles cover a region of Las Vegas in a consecutive order i.e tile 1 and tile 2 are geographically beside each other. If the training set was split with the first 60% of values, it could result in covering only a certain part of the city e.g. the centre. The test set could then cover a suburban region with farms, for example. This would lead to a model that produced poor classification results, as the training set does not contain a good representation of the whole dataset. Given this, the training, validation and test sets are all consist of random tiles sampled across the whole Las Vegas area - this should lead to more representative sets.

Figure 4.1: An example of a GeoTIFF and GeoJSON file overlaid on each other, viewed in the QGIS application.

The images are in the GeoTIFF format and are each 650x650 resolution. This format embeds georeferencing data within TIFF images, so each image has corresponding latitude and longitude coordinates. Each GeoTIFF image comes with a GeoJSON file. The GeoJSON file specifies the polygons (if any) that are present within that tile. Each polygon is the footprint of a building, which have been manually annotated by a human. This serves as the training information, before pre-processing. These files can be viewed in certain applications such as QGIS [16] (see Figure 4.1).

For training purposes, the data must be pre-processed. This is carried out by creating binary masks (see Figure 4.2) of the GeoTIFF and GeoJSONs. This required a number of scripts to be written, using a variety of useful packages such as GeoPandas [17] and Rasterio [18]. This allowed the GeoTIFF and GeoJSON images to be combined/overlayed in one image on a per-pixel basis, in a binary format - the buildings are white (positive class), everything else is black (negative class). It was decided to use these binary masks as they remove colour from the satellite image, allowing the network to learn and focus on more informative features like shape. The masks are also reduced in size to 256x256 to reduce overhead when training the network while keeping the majority of the information - using 650 x 650 images caused the network to have low throughput and low utilisation. Also, using large batch sizes with 650 x 650 images would cause the GPUs to run out of memory.

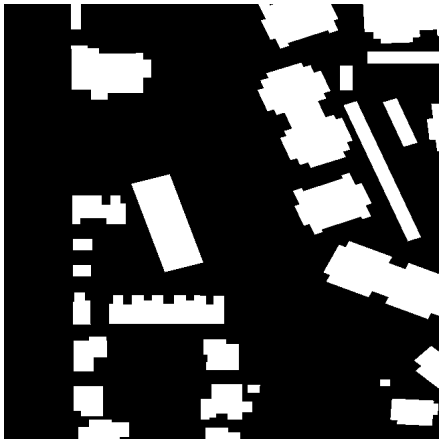


Figure 4.2: A binary mask applied to the satellite image in Figure 4.1.

Each pixel in each mask is described by its own label. It is either in the positive (building) or negative (not building) class. There is a heavy imbalance in the number of negative labels compared to the number of positive labels. This is simply due to the fact that most of the images are not buildings. A lot of the landscape comprises of farm land, parking lots, roads, areas of open space etc. In fact, the percentage of positive labels in the training set is 20.5%. This causes many problems in training, as the network can learn to classify every pixel as the negative class and still gain good results. To combat this, experiments had to be carried out with regards to the loss function, which you can read about in Section 5.1.

To evaluate the models on the dataset, the test set and validation set are necessary. To produce the final experimentation results, the accuracy of the different models will be evaluated on the test set. The accuracy of the model will be evaluated using an IoU

(Intersect over Union) score (also called the Jaccard Index) [19]. The Jaccard Index produces a value between 0 and 1 (where closer to 1 is better). The equation can be seen below:

$$J(x,y) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

where A and B are both sets. In this project, A and B are the ground truth mask and the predicted masks, respectively. IoU essentially measures the overlap between the two images and can be explained further in Figure 4.3.

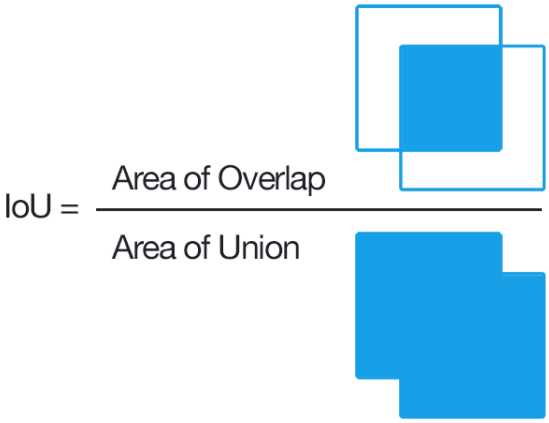


Figure 4.3: A simplified version, showing the calculation for the IoU evaluation metric.

Chapter 5

Preprocessing

5.1 Loss Function

Two loss functions were attempted, Cross-Entropy loss and the Lovasz-Softmax loss [20]. Pixel-wise cross-entropy loss has proved to be very popular and successful in semantic segmentation tasks [21]. For this reason and because it was used in the original FCN paper [2], the experiments began with the use of cross-entropy. The imbalance in the dataset still caused problems for the cross-entropy function. Therefore, weights had to be added to the loss function to improve the semantic segmentation of the network. The weights are assigned to each class, for example, weights of [1, 100] would assign a weight of 1 to the negative class and a weight of 100 to the positive class. A variety of weights were tested on the baseline FCN (see Table 5.1), and it was observed that using weights equivalent to the ratio between the positive and negative label counts resulted in the highest IoU scores. These IoU scores recorded in Table 5.1 were carried out over 3 different random seeds and before the final hyperparameter tuning was carried out. They were ran on default hyperparameters i.e. using the RMSProp optimiser, a batch size of 32 and a learning rate of 0.0001. Early stopping was also used which is explained further in Section 7.2.1. However, these results show the differences and importance of adding weights to the cross-entropy loss function.

Weights [Negative, Positive]	IoU Accuracy
[1, 1]	0.6494
[1, 3.5]	0.6786
[1, 3.88]	**0.6932**
[1, 4]	0.6929
[1, 4.5]	0.6831
[1, 5]	0.6903
[1, 6]	0.6653
[1, 8]	0.6020

Table 5.1: IoU accuracies for the baseline FCN model, varying the weights in the pixel-wise Cross-Entropy loss function.

The original equation for Cross-Entropy loss is described in Section 2.3. This can also be used in the semantic segmentation part of the combined network. However, since weights are used to add balance to the training labels, the equation will differ slightly. The equation for pixel-wise Cross-Entropy loss with class weights can be seen below:

$$Loss(x, class) = weight[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right)$$

where x is the input pixel and $class$ represents whether the pixel contains a building or not.

In Table 5.1, the impact of adding weights to the cross-entropy loss function can be observed. Without adding a weight to the positive class, a small accuracy drop occurs. When no weights are used (i.e [1, 1]), the average IoU accuracy on the validation set for the default parameters is 0.6494. By adding a weight to the positive class that is representative of the ratio of the number of positive to negative class labels (i.e. 3.88:1), accuracy improvements of around 5% can be seen. Adding this weight of [1, 3.88] improves the accuracy by the greatest amount, therefore it will be used in the final experiments for cross-entropy.

The Lovasz-Softmax loss [20] is a loss function that is used to directly optimise the IoU loss and was designed for use in the semantic segmentation domain, unlike IoU, it is differentiable. As IoU is being used as the evaluation metric for this report, it is an interesting loss metric to investigate - also, in many cases, it has performed better than cross-entropy for models that evaluate using the Jaccard Index/IoU. To see the differences in IoU accuracy score, between the two loss functions, see Section 7.3.

5.2 Data Augmentation

Currently the dataset contains 2311 training images - although this appears a relatively large amount of images, it is still limited in size compared to models that use tens/hundreds of thousands of images. In the case of VGG (described in Section 6.1), 1.2 million images were used to train the network. To increase the size of the dataset, a variety of data augmentation techniques [22] can be used e.g. rotation - a satellite image rotated 45 degrees is still a satellite image and the buildings will still maintain their structure and shape. By increasing the amount of training data, IoU accuracy boosts are expected. The data augmentation techniques are only carried out on the training set (identical transformations for both the image and the label) and not the validation set or test set.

Normalisation is carried out on the image. The mean and standard deviation is calculated across the 3 channels of the raw satellite image. The training, validation and test set were all normalised. This normalisation provides scaling and centering. By scaling the data, it can speed up training and the time to convergence. The centering of the data helps keep features in the same range and stops gradients from spiralling out of control.

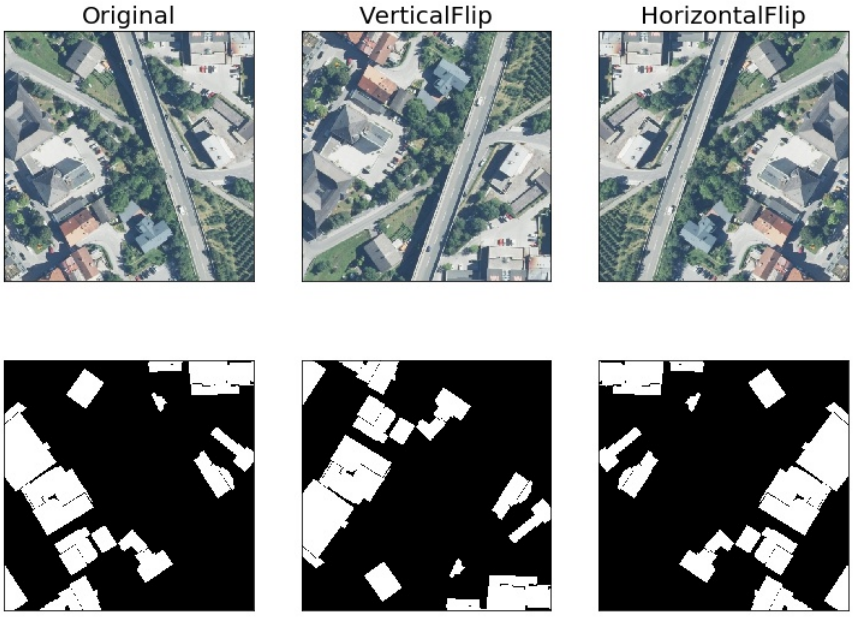
For satellite images, the rigid structure of the buildings needs to be preserved. If some distortion technique is used, such as Gaussian noise or an elastic transform, it may produce an image that is not representative of any image that will be seen in the future - buildings are rarely found with abnormal shapes. For satellite images, certain transformation techniques are better than others, such as; cropping, rotations, reflections and scaling [23]. An example of these sort of techniques can be seen in Figure 5.1, taken from the DSTL Satellite Imagery detection challenge on Kaggle [24].

After some brief qualitative experiments into various data augmentation techniques, it was decided that the approaches that would be experimented with were: Horizontal Flip, Rotation and Vertical Flip. These methods were tested both individually and in combination to measure the impact of their addition. These transformations take place as the data is loaded and passed into the epoch for training. All of these transformations take place randomly, with some probability. In the experiments carried out, this probability was set to 0.5. Therefore, at every epoch, the training data would be slightly different each time, due to the randomness that is inherent in each of the operations. For rotations, a maximum degree was set, which was 45°. Therefore, the original image could only be rotated clockwise or anti-clockwise by at most 45° (but also any value $\pm 45°$). The effect of adding each of these transformations on the IoU accuracy can be viewed in Table 5.2. These experiments were carried out on default hyperparameters until early stopping (i.e. using the RMSProp optimiser, a batch size of 1, a learning rate of 0.0001 and the cross-entropy loss function with [1, 3.88] weighting) and an average was taken over 3 runs.

Augmentation	IoU Accuracy
No Augmentation	0.6932
Horizontal Flip	**0.7173**
Rotation	0.6341
Vertical Flip	0.6440
Horizontal Flip + Rotation	0.6566
Horizontal + Vertical Flip	0.6898
Rotation + Vertical Flip	0.6925
All 3 techniques	0.6489

Table 5.2: IoU accuracies for various data augmentation techniques over 3 runs, on the validation set.

The results from the data augmentation experiments were surprising. By increasing the size of the dataset, using data augmentation and varying the images that were trained with from epoch to epoch, accuracy boosts were expected for all techniques. However, as seen in Table 5.2, using rotations made results significantly worse. Additionally, using vertical flips seemed to deteriorate the IoU accuracy. However, the addition of horizontal flips improved the performance of the system by 3.6%. When calculating the final results for the FCN and the combined architecture, the use of horizontal flipping will be incorporated at the data loading stage.



(a)

(b)

Figure 5.1: A variety of useful data augmentation techniques, used on the DSTL Kaggle challenge dataset [24].

Chapter 6

Architecture of the Models and Preliminary Experiments

6.1 Fully Convolutional Network

The FCN architecture was analysed and varied to find a network with the best possible results on the dataset. For the downsampling phase, a pretrained model was used: VGG-16 network initialisation, with the fully-connected layer at the end removed. This helped to streamline the network so that it could be trained faster than an end-to-end network, given that, the weights/features have already been learned. VGG-16 [25] is a popular image classification network that is used as a pretrained model in many domains. It uses a uniform architecture, which is 16 layers deep, stacking convolutional layers with 3x3 receptive fields. Between these convolutional layers are some max pooling layers. This pretrained model was trained on over 1 million images from the ImageNet dataset [13]. These pretrained models are beneficial because many features that are learned from the ImageNet dataset will be transferable to many domains, including satellite imagery. The pretrained model contains weights and biases that represent the features of the dataset. These features can be very low level and transferable such as edges or horizontal lines.

The FCN architecture uses hypercolumns [14], which were described in Chapter 3. These hypercolumns have worked well in practice for many other semantic segmentation tasks [26]. To include a hypercolumn implementation, concatenation of intermediate layers in the model is necessary. This will, in turn, give every pixel a feature vector that is made of a concatenation of a local slice from each layer of the FCN.

A number of experiments were carried out to optimise the upsampling path of the network, given that it was decided to use a pretrained model for downsampling. There were 3 main options; FCN-8s, FCN-16s and FCN-32s. These are all variants of each other that can be visualised in Figure 6.1. In Figure 6.1, the top line represents FCN-32s, which is a linear model that uses no skip connections. FCN-16s (on the second line), use a skip connection from the 4th pooling layer of the downsampling path (VGG-16) and combines its predictions with the predictions of the final convolutional

layer. By adding a skip connection, it allows for more localisation and hopefully less coarse predictions. In the satellite domain, these coarse predictions can cause unwanted merging of buildings in the output segmentation maps - it is important that gaps between buildings are sufficiently represented at the final stage. Additionally, there is FCN-8s, which also uses skip connections. In this case, it uses the skip connection that FCN-16s uses as well as a second skip connection taken from a higher level (3rd pooling layer) than in FCN-16s, and combines it with the final convolutional layer. By combining both coarse layers and fine layers the FCN can make more precise local predictions while still respecting the overall global structure of the original image. Although not attempted in the original paper, in this report, additional models are investigated - which will be called FCN-4s and FCN-2s. These models also combine predictions from the 2nd pooling layer and the 1st pooling layer, respectively.

In the previous experiments into the loss functions, in Section 5.1, and data augmentation, in Section 5.2, FCN-32s was used as the default architecture i.e no skip connections were added.

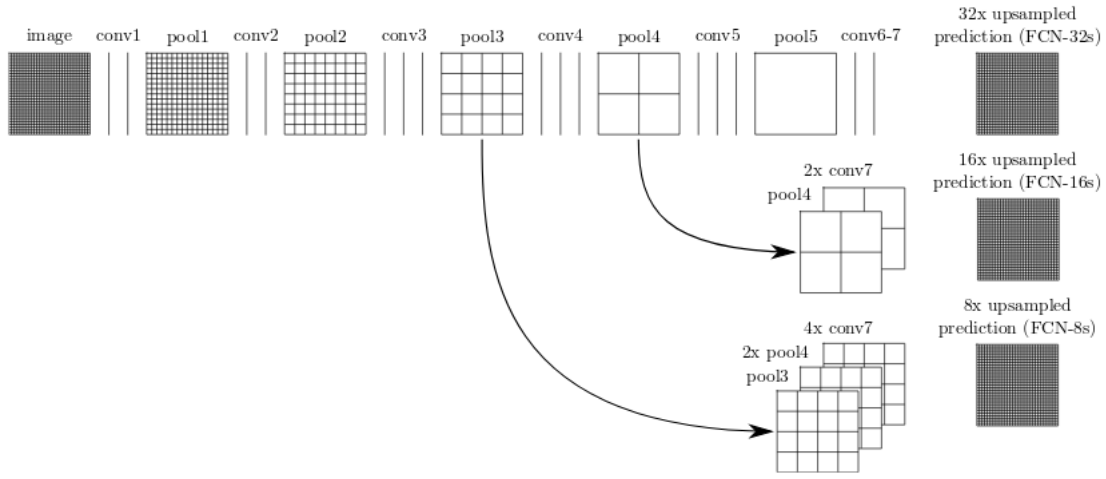


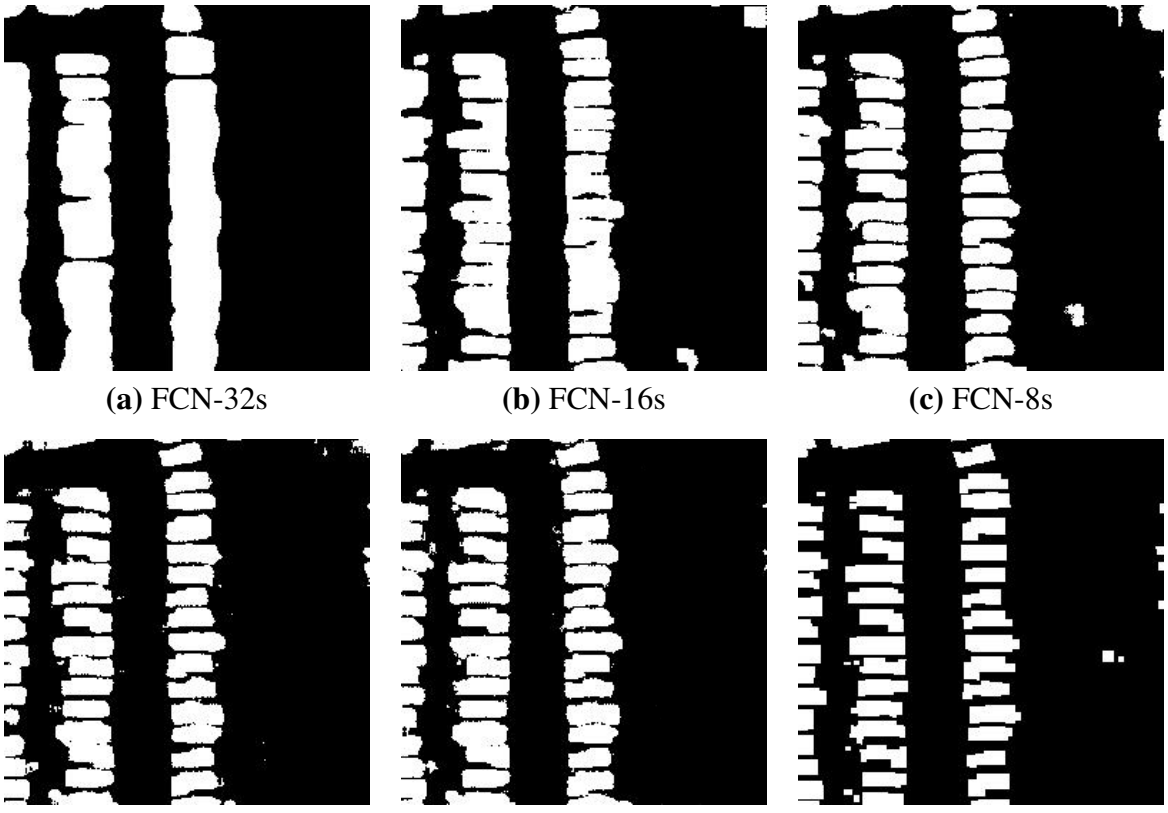
Figure 6.1: The difference between FCN-8s, FCN-16s and FCN-32 [2].

These 5 models were ran 5 times each, with different random seeds, on default hyperparameters for the network (i.e. using the RMSProp optimiser, a batch size of 1, a learning rate of 0.0001 and the cross-entropy loss function with [1, 3.88] weighting) and the average IoU results can be seen in Table 6.1, as well as, the time taken to train the network over 100 epochs. The differences in the coarseness of the predictions as more skip connections are added can be noticed, compared to the ground truth label (see Figure 6.2). The more skip connections that are added (FCN-2s has the most skip connections), the sharper the image becomes and the higher the IoU accuracy becomes. The added localisation from the skip connections is clearly a benefit for this task. It is clear from Table 6.1 and Figure 6.2 that adding skip connections improves the performance of the model. However, there is definitely a trade-off. By adding these skip connections, the complexity of the models are increased as well as the time it takes to train them. In order to choose a model for the final experiments a balance must be found between IoU accuracy and training time.

As seen in Figure 6.2, the segmentation attempt is quite poor for FCN-32s because of

FCN Type	IoU Accuracy	Time to 100 Epochs (hours)
FCN-2s	**0.7324**	20.35
FCN-4s	0.7287	20.19
FCN-8s	0.7232	19.38
FCN-16s	0.7139	18.44
FCN-32s	0.6932	18.03

Table 6.1: IoU accuracies and time to 100 epochs, for the 5 different FCN types showing the affect of adding skip connections to the network.



(a) FCN-32s **(b)** FCN-16s **(c)** FCN-8s

(d) FCN-4s **(e)** FCN-2s **(f)** Ground truth mask

Figure 6.2: The different FCN Types and their segmentation prediction on the same satellite image compared with the ground truth, as more skip connections are added the prediction becomes more sharp.

the lack of localisation, often forming one contiguous block for an array of separated buildings. The prediction from FCN-8s, FCN-4s and FCN-2s are all very similar and have good localisation, compared to FCN-16s and FCN-32s. The added complexity that is present in FCN-4s and FCN-2s does not seem to have enough impact on the prediction to justify using either of them over FCN-8s. Given this and the timing results from Table 6.1, FCN-8s are used in the final model on the test set.

6.2 Autoencoder

In terms of the architecture of the autoencoder, there are no skip connections and a mirrored encoder and decoder are used. The encoder comprises of five 3x3 convolutional layers (see Section 2.2.2) with 2x2 max pooling layers (see Section 2.2.3) in between the convolutional layers. The decoder uses 2x up-sampling (see Section 2.2.4) followed by a 3x3 transposed convolution. The autoencoder can therefore be referred to as a Convolutional Autoencoder (CAE). The upsampling used in this architecture is bilinear interpolation at a scale factor of two. The autoencoder does not use any pre-trained model and was trained from scratch. This CAE should have enough layers for it to learn useful representation of the label maps in lower dimensional space and not just reproduce the labels using some identity function. The design for this autoencoder was derived from the architecture used in a paper from CVPR 2018 [3]. As mentioned in Chapter 4, it was decided to reduce the 650x650 images to 256x256 images. Given the size of the label space and the use of max pooling and upsampling, the autoencoder network that is being used can be visualised in Figure 6.3. At the very end of the network there is a Tanh layer.

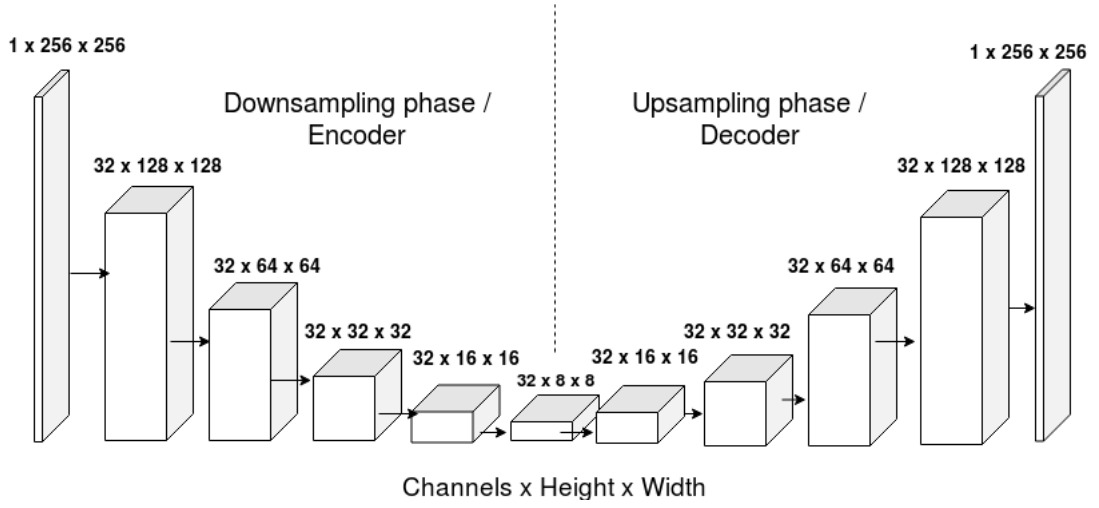
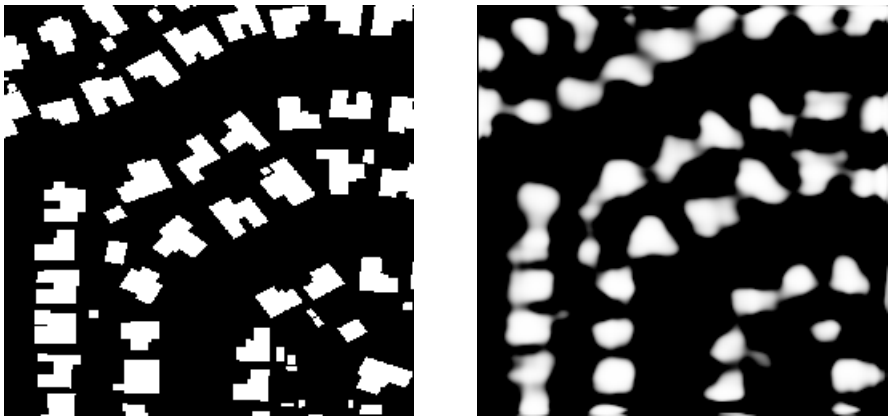


Figure 6.3: The change in the dimensions of the original image as it passes through the network, via the convolutional layers, max pooling layers, upsampling layers and transposed convolutional layers.

The final autoencoder will use the described 5 layer architecture above and it's hyper-parameters such as learning rate, batch size etc. will be tuned to try and reduce the

loss between the reconstructed label and the ground truth label. With minimal loss, a good reconstruction should be produced while learning the modes of the data in the bottleneck layer (32 x 8 x 8 size). An example of the sort of reconstructions that the autoencoder makes can be seen in Figure 6.4.



(a) Ground truth (b) Reconstruction

Figure 6.4: An example of a reconstruction carried out by the autoencoder with low loss, the 256x256 image is compressed into an 8x8 representation and then uncompressed back to 256x256.

From the initial experiments carried out using an autoencoder with cross-entropy loss, unsatisfactory results were gained. In light of this, the mean squared error loss function was attempted. This is a popular loss function that is often used with autoencoders. The initial experiments were re-run using mean squared error loss and there were significant improvements in the reconstruction. Therefore, for the final autoencoder experiments, mean squared error is used as the loss function.

6.3 Combined Architecture

As described in Chapter 3, a combined architecture will be used in this project, inspired by a recent paper [3]. This model consists of two training phases (as seen in Figure 3.1).

The first phase uses the convolutional autoencoder to model the ground truth annotations (binary masks) while ignoring the actual raw satellite images. The second phase involves training the FCN as usual but this time employing the convolutional autoencoder as a regulariser, by attaching the decoder to the FCN. This results in two prediction pathways; a primary and auxiliary. The primary prediction pathway involves using the hypercolumn feature for each pixel to produce a binary mask. The auxiliary pathway involves generating some abstract representation that is compatible with the input of the decoder. From there, backpropagation occurs through both of these paths. The backpropagation through the decoder updates the parameters of the FCN but not

the parameters of the decoder - the parameters of the decoder are frozen during training. After training has finished, the decoder can be discarded along with the output from the auxiliary path.

Chapter 7

Final Experiments and Results

7.1 Baseline

For the baseline, it was decided that a random scattering of predictions would be used in the output space. This should aid the reader in understanding how the implemented models improved the predictions over random predictions. Two random predictions were used; 50% of the pixels were classified as building and in the other case, 20.5% of the pixels were classified as building. 20.5% was used because this corresponds to the prior probability of a pixel being a building. As mentioned in Chapter 4, the prior probabilities of the two classes are the following: $P(Background) = 0.795$ and $P(Building) = 0.205$. Also, to put these baseline values in perspective, Table 7.1 includes results for the cases when all pixels were classified as building and no pixels were classified as building.

PREDICTION TECHNIQUE	IOU ACCURACY
ALL PIXELS ARE NOT BUILDING	**0.2852**
ALL PIXELS ARE BUILDING	0.2045
50% ARE BUILDING	0.1474
20.5% ARE BUILDING	0.0864

Table 7.1: IoU accuracies for baseline predictions.

The IoU accuracy accounts for correctly classified buildings labels. If no positive labels are given at all, the IoU score returned is the highest. This is expected because most pixels in the image will generally be not buildings - as described above. The random baselines will produce a scattering with no structure and hence will perform no better than every pixel being classified with a negative label. An additional baseline technique that could be included in the future is to produce random clusters of positive labels beside each other, in an attempt to model building structure. These clusters would have a minimum amount of pixels because almost no buildings will be represented by a single isolated pixel - the building would be simply too small. When all pixels are classified as buildings i.e the image is completely white, it results in

an IoU score of 0.2045 which is expected as it is equivalent to $P(Building) = 0.205$, mentioned above.

7.2 Autoencoder

7.2.1 Hyperparameters

To ensure that the autoencoder would add value in the combined stage, it was necessary to get a model that would be able to reconstruct the labels with as little loss as possible. This, in turn, should lead to an encoded representation that would be effective as a regulariser in the FCN training stage. The autoencoder had to be tuned via a hyperparameter search to find the best possible model. The search was carried out across several different hyperparameters. For each experiment across the grid search, 75 epochs of training were used. However, additionally, early stopping was implemented so that models that were poorer than expected were cancelled early to prevent unnecessary GPU and time usage.

The early stopping algorithm stops training when the validation loss is lower, for every epoch, than the current lowest loss after n epochs. n is the patience variable and can be customised. For this report, a patience value of $n = 5$ was used. Early stopping stops training when it starts to realise that the model is overfitting to the training set i.e. it has learned too much information from the specifics of the training data and these specifics don't necessarily apply to the validation/test set.

Learning rate, batch size and the optimiser were changed across all of the experiments. The learning rate was varied within the following range: [0.001, 0.0001, 0.00001, 0.000001]. This range was decided after a number of initial experiments in which it was realised that a learning rate of 0.01 was either producing black images (no building classifications) or there was too much fluctuation for it to be useful. A learning rate any less than 0.000001 needed too many training epochs to reach a significantly low loss.

In terms of batch size, it was decided that a range of [1, 32, 64, 128, 256], would be used. A batch size of 512 was causing too many memory errors on the GPU and so the batch size was capped at 256. If the batch size is 256, a gradient will be computed over 256 images before the weights are updated - this acts like an average gradient update, producing smoother weight updates. If the batch size is 1, it is called online learning. Online learning will be a lot slower than with larger batch sizes, often leading to researchers straying away from smaller batch sizes. The weights are updated after every image sample - this can lead to high fluctuation in weights/accuracy but can also produce better results. Although many researchers use a batch size between 64 and 512, it has recently been shown that batch sizes of 32 are actually better in practice [27]. It was decided to vary the batch size alongside the learning rate because increasing the batch size can lead to similar learning curves as you would get by continually reducing learning rates [28].

In this report, experimentations were done into the best optimiser for this task. The two optimisers that were attempted were RMSProp [29] and the Adam optimiser [30]. These are two popular optimisers that work well in many different tasks therefore they were chosen for this report. Both the learning rate and batch size were varied as described above, for each of the two optimisers. The standard values for the RMSProp algorithm of 0.9 for momentum and 0.00005 for weight decay were used as recommended by the author. For the Adam optimiser, the given values for beta 1 and beta 2 also remained as recommended in the original paper [30].

7.2.2 Results

After varying the batch size, the learning rate and the optimiser, a number of conclusions could be made. In the 5 models with the highest accuracy (known as the top models), a batch size of 1 was used in 4/5 of the cases. A clear trend developed, as the batch size increased, the average loss across all experiments for that particular batch size also increased. This can be viewed in Table 7.2.

BATCH SIZE	AVERAGE CROSS-ENTROPY LOSS
1	**0.0492**
32	0.0878
64	0.0995
128	0.1173
256	0.1233

Table 7.2: Average cross-entropy loss for different batch sizes, across all autoencoder hyperparameter experiments - on the validation set.

In the top 5 models, 4/5 of them made use of the Adam optimiser over the RMSProp optimiser. For this task, the Adam optimiser seemed to perform a little better - perhaps its hyperparameters were more well suited. The learning rate used in all of the experiments did not seem to have much impact on the final loss produced by the autoencoder. All of the learning rates, except for 0.000001 were found in the top 5 models. Using a learning rate of 0.000001 caused too slow of a convergence for the designated epoch budget (75).

The model with the lowest validation loss (**0.0392**), called Autoencoder 2, was saved and used for the combined architecture. This models hyperparameters were a batch size of 1, mean-squared error loss, a learning rate of 0.0001 and using the Adam optimiser. The learning graph for the top 3 models can be seen in Figure 7.1, Autoencoder 1 and 3 converge and stop due to early stopping. However, Autoencoder 2, the best model, runs for all 75 epochs and doesn't stop. The weights of this model are saved/checkpointed at each layer - once training has finished, the final combined architecture can then freeze and use the decoder half of the Autoencoder 2 model.

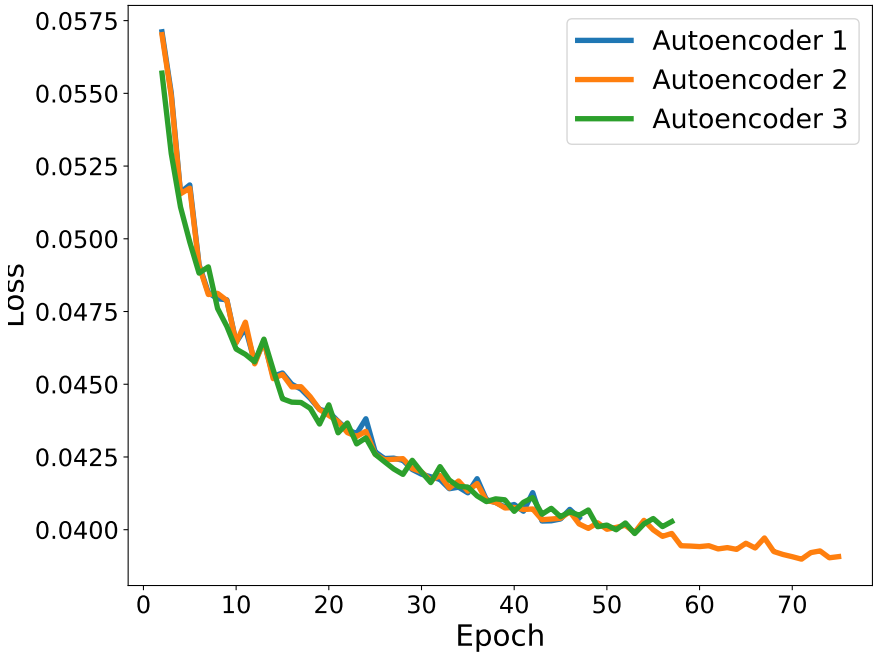


Figure 7.1: The loss vs epoch graph for the top 3 autoencoder models, on the validation set.

7.3 FCN

7.3.1 Hyperparameters

In terms of the experiments carried out for the Fully Convolutional Network, a grid search was also implemented similar to the autoencoder (as described in Section 7.2.1). However, there are a few changes due to the difference in model/architecture. The learning rate is varied in a different range, for the FCN, this range is [0.0001, 0.00001, 0.000001, 0.0000001] - these were chosen based on some initial experiments. In terms of batch size, the same values were used for the FCN as for the autoencoder: [1, 32, 64, 128, 256]. Similarly to the autoencoder, both RMSProp and the Adam optimisers were used in the experimentations. For the FCN, two different loss functions are investigated - as described in Section 5.1. These loss functions are Cross-Entropy loss and the Lovasz-loss function, both were included in the grid search. Early stopping was also implemented for FCN training. However, instead of using the loss as an indicator of when to stop, the IoU accuracy is used - this is because the goal is to maximise IoU accuracy and not minimise loss.

7.3.2 Results

After all experiments were run and the results were output to CSV (Comma-separated values) files. An analysis of the results was carried out using Jupyter Notebook and Python (version 3). Initially, it was clear that the batch size was an important factor and that a batch size of 256 produced very poor results. Perhaps an average over 256 examples did not offer a clear path to update the gradients. It was found that a batch size of 1 produced the best results, across the board - similar to the results for the autoencoder in Section 7.2.2. In fact, as the batch size was increased the average IoU across the experiments started to decrease (as summarised in Table 7.3). These results are in line with recent studies into the impact of small batch sizes [27].

BATCH SIZE	AVERAGE IOU ACCURACY
1	**0.6795**
32	0.5943
64	0.5612
128	0.4114
256	0.2439

Table 7.3: Average IoU accuracies for different batch sizes, across all FCN hyperparameter experiments - on the validation set.

As described in Section 5.1, the Lovasz loss function is compared to the cross-entropy loss function. After running all of the experiments for both Lovasz-loss and cross-entropy, it was found that the Lovasz loss produced significantly worse results than cross-entropy. Most of the cross-entropy models did reasonably well with a few completely failing. However, for Lovasz loss, most of the models failed giving an accuracy worse than when all pixels are given negative labels (black image). However, there were a few models that used Lovasz loss that produced IoU accuracies of over 0.60. Overall, on average, the cross-entropy loss function produced much better results across their respective top 5 models - this is reflected in Table 7.4. Note that these results are higher than in other tables because in the other cases an average was taken over all experiments and not just the top 5. This was carried out because the failure rate of Lovasz loss was very high and an average over the best models was more representative of its potential performance. Therefore, although the Lovasz-loss function looked particularly promising for this domain, it is recommended that the cross-entropy loss function should be used. However, a more extensive hyperparameter search may be required to optimise the Lovasz-loss function that seems to have local optima that are more difficult to reach than cross-entropy.

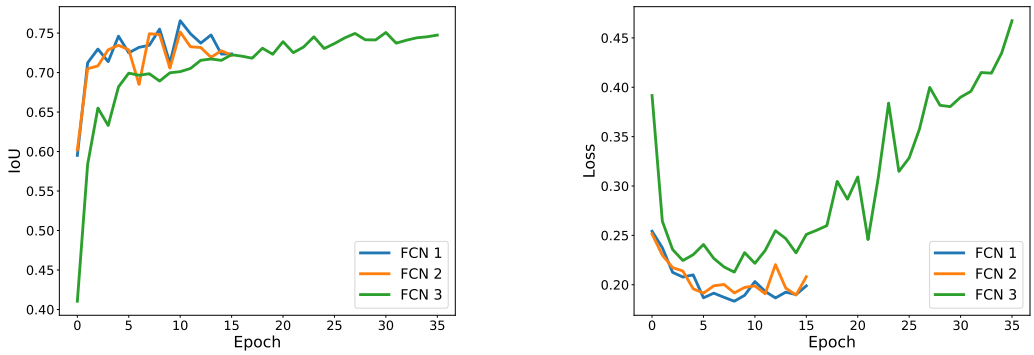
LOSS FUNCTION	AVERAGE IOU ACCURACY
CROSS-ENTROPY	**0.7377**
LOVASZ LOSS	0.5591

Table 7.4: Average IoU accuracies for the **top 5 models** for each loss function, across all FCN hyperparameter experiments - on the validation set.

In terms of learning rate, none in particular worked the best for this model. In the 10

models with the highest accuracy, learning rates of [0.0001, 0.00001, 0.000001] were used. However, none of those best models used a learning rate of 0.0000001. The learning rate was too low for the 75 epochs that were used for training, potentially if a larger epoch budget was used this model may have performed better. However, given the length of each experiment (around 2 hours, in parallel across the 4 GPUs), increasing the epoch budget was unfeasible. Also, both the RMSProp and Adam optimisers performed well in training, with neither performing significantly better than the other. In the top 10 models, 6/10 used the RMSProp optimiser and therefore 4/10 used the Adam optimiser. These top 10 models had an IoU accuracy on the validation set in the range 0.704 - 0.766.

The model that achieved an accuracy of 0.766 on the validation set, called FCN 1, had the following hyperparameters: a learning rate of 0.0001, a batch size of 1, used the Adam optimiser and the cross-entropy loss function. It reached this value after only 10 epochs, which can be seen by the learning graph in Figure 7.2, along with the other the other 2 best models. It is interesting to analyse the two graphs for FCN 3. The accuracy on the validation set steadily increases until it stops at a near optimal point. However, the loss graph shows a model that starts a dramatic increase in loss from epoch 8 onwards. An interpretation of these results is that the model is producing more accurate results that have a better overlap with the ground truth. However, when the model is making an incorrect prediction, it is making increasingly more incorrect predictions. This shows that the relationship between validation accuracy and validation cross-entropy loss is not as strongly correlated as one might have originally thought.



(a) Epoch vs IoU Accuracy

(b) Epoch vs Cross-Entropy Loss

Figure 7.2: The learning graphs for the best 3 FCN models, on the validation set, halting prematurely due to the early stopping algorithm.

The top 5 models (5 models with the highest IoU accuracy on validation set) were retrained with 3 different seeds. This produced 15 final FCN models for all of FCN 1-5. These 15 models were then used for inference on the held-out test set, and an average was taken over the 3 seeds for the IoU accuracy at the last epoch (either 75 or early stopping). For these final tests, the patience variable in early stopping, was increased from 5 to 7 to give models further chance to reach a more optimal IoU. The

final results for the top 5 FCN models on the test set, compared with the baseline, can be seen in Table 7.5.

MODEL	AVERAGE IOU ACCURACY
ALL PIXELS ARE NOT BUILDING	0.2852
FCN 1	0.6831
FCN 2	**0.7041**
FCN 3	0.6103
FCN 4	0.6140
FCN 5	0.5336

Table 7.5: Average IoU accuracies for the top 5 FCN models on the **Test** set.

As seen in Table 7.5, the best model on the validation set was not the best model on the test set. Therefore, it is important to evaluate a number of the best models that were trained. FCN 2 was the only model to achieve over 0.70 IoU accuracy on average and an example of one of its predictions can be viewed in Figure 7.3.

(a) Raw Image (b) Ground Truth (c) FCN 2 prediction

Figure 7.3: An example of a prediction made by the best model on the test set, FCN 2, compared with the ground truth prediction.

7.4 FCN with Combined Autoencoder

7.4.1 Hyperparameters

To be consistent and for a fair test, all of the same experiments were carried out as with the standalone FCN (described in Section 7.3.1). However, the batch size was decreased to a new range, given that a batch size of over 32 produced poor results in all of the previous experiments. The new batch size range that was used for the final combined FCN experiments was: [1, 4, 8, 16, 32]. Early stopping was also used with IoU accuracy. Additionally, for this combined FCN it was decided that another experiment should be carried out. Due to the fact that the autoencoder uses mean squared error loss and the FCN uses cross-entropy loss, a weighting factor, α, was added to the auxiliary Mean-Squared error loss. By default this value is $\alpha = 1$, however, additional experiments were ran in an attempt to boost accuracy. The weighting factors that are investigated are $\alpha = [1, 2, 5, 10, 15, 20, 30, 50]$.

The total loss used in backpropagation is therefore:

$$L(x,y) = L_1(x,y) + \alpha L_2(x,y)$$

where $L_1(x,y)$ = Cross-Entropy Loss Function from the FCN and $L_2(x,y)$ = Mean-Squared Error Loss Function from the autoencoder. x and y are the predictions and ground truth labels, respectively.

7.4.2 Results

As with the previous experiments, the batch size had a large impact on the final accuracy of the model - even with the reduced batch size range of [1, 4, 8, 16, 32]. Out of the top 5 models, a batch size of 1 was used in 4/5. It is clear that for this task a lower batch size is much better for optimising the models. Also, 4/5 of the top models used the RMSProp optimiser - similar to the FCN experiments. There was no significant difference in results for different learning rates as other hyperparameters seemed to have a larger effect on the final accuracy.

After running all of the experiments using the combined network, the results were actually worse than the baseline FCN. The best combined FCN models produced segmentation maps with around 0.71 IoU accuracy. The best baseline FCN models produce around 0.75 IoU accuracy on the validation set. After consideration, it was decided that the autoencoder was not producing an accurate enough representation of the original image. The structure and shape of the buildings were not being represented clearly enough. It was then decided that the architecture of the model should be changed from the previous architecture described in Section 6.2 and in Figure 6.3. More filters were needed in the architecture to improve its accuracy. After some experiments and analysis of the results, it was decided to use an architecture primarily using 128 filters/channels at each layer. However, 32 channels are used at the first layer. The new network can be visualised in Figure 7.4.

In the old architecture, the best model had a validation loss of 0.0392, using Mean-Squared error. The improved, new, architecture significantly reduced this loss. On the validation set, the best model using the new architecture achieved a loss of **0.0128**. Although this is much lower, it is still not conclusive that the final experiments on the test set will improve. The reconstruction is still not perfect, as seen in Figure 7.5. For the final experiments, this architecture was preferred because of its lower loss.

In terms of the weighting factor on the auxiliary loss α, it was found that the optimal weighting factor was ~ 10. The effect of the weighting factor on the final IoU accuracy followed a bell shaped curve, with 10 at the peak of the curve. This can be seen in Figure 7.6. Therefore, for the final experiments on the test set, a weighting factor of 10 was used.

After the previous conclusions, the final experiments were then ran on the test set. The performance of the best 3 models on the validation set can be seen in Figure 7.7. The best performing model, FCN-COMB 1, had the following hyperparameters: a batch

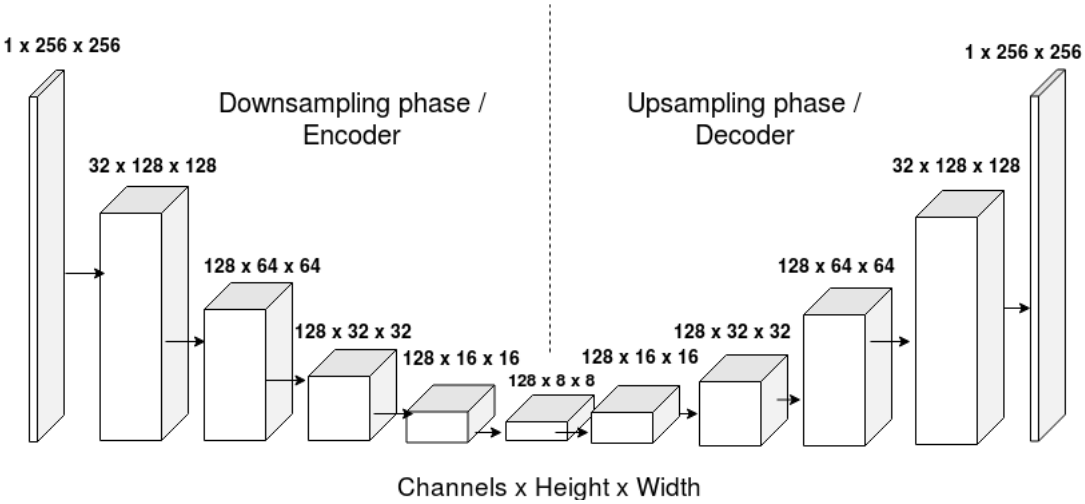


Figure 7.4: The change in the dimensions of the original image as it passes through the **new** network, via the convolutional layers, max pooling layers, upsampling layers and transposed convolutional layers.

(a) Ground truth (b) Reconstruction

Figure 7.5: An example of a reconstruction carried out by the **new** autoencoder with the lowest loss, the 256x256 image is compressed into an 8x8 representation and then uncompressed back to 256x256 as seen in Figure 7.4.

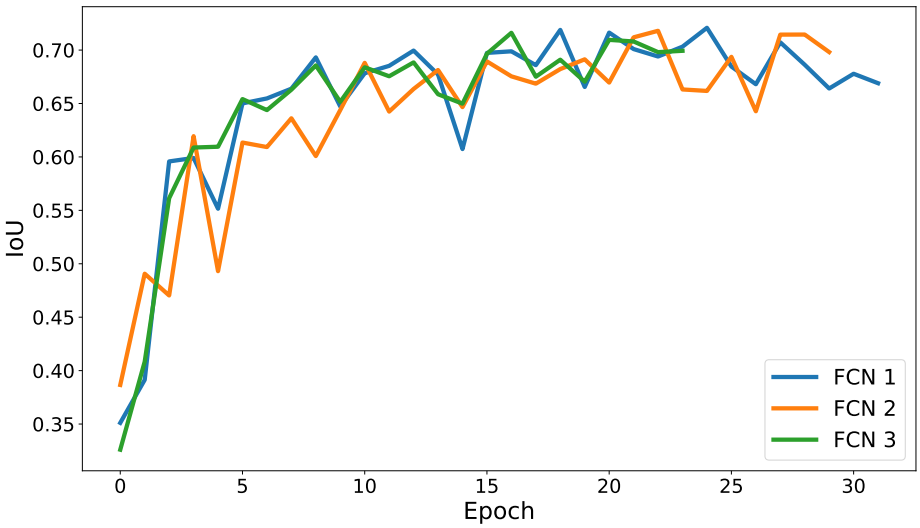
Figure 7.6: The change in IoU accuracy as the weighting factor α is varied, it can be seen that a weighting factor of around 10 is the optimal weighting factor.

size of 1, a learning rate of 0.0001, using the RMSProp optimiser, Cross-Entropy loss function for the main branch, Mean-Squared error loss for the auxiliary branch and a weighting factor of $\alpha = 10$ on the auxiliary loss. It produced an IoU accuracy of 0.7181 on the validation set. As seen in Figure 7.7, the loss value for the combined architecture is not shown. The reason for this is because two different loss functions are used and the weighting factor of 10 causes the total loss to be very temperamental and is therefore not useful.

As with the FCN, the top 5 models on the validation set were ran for 3 different seeds on the test set. This produced 15 test results, and an average was taken for each of the 5 models. The final experiment results on the test set, compared with the two previous baselines, can be seen in Table 7.6.

Model	Average IoU Accuracy
All pixels are not building	0.2852
FCN 2	**0.7041**
FCN-Comb 1	0.6931
FCN-Comb 2	0.6701
FCN-Comb 3	0.6103
FCN-Comb 4	0.6140
FCN-Comb 5	0.6259

Table 7.6: Average IoU accuracies for the top 5 FCN models on the **Test** set, compared with the previous baselines.



(a) Epoch vs IoU Accuracy

Figure 7.7: The learning graph for the best 3 **combined FCN** models, on the validation set, halting prematurely due to the early stopping algorithm.

Chapter 8

Conclusions

8.1 Conclusion

From the results in this report, it is clear that CNN architectures are well suited to this semantic segmentation task. After the implementation of a FCN was carried out and certain parameters were learned, very good baseline results were found on the test set i.e. a 0.7041 IoU accuracy. This model was able to identify most of the buildings without any issues. However, many of its predictions were somewhat imprecise. The network struggled to represent the rigid structure of the buildings and instead produced masks that were blurred around the edges and rounded in shape. The fact that the model was able to locate the buildings, in the raw image, still provides useful information. However, there was potential to improve on this. With extra information, it is possible to not just describe the buildings location but also accurately describe its footprint.

In theory, the addition of the decoder into the training pipeline should have allowed for some of these accuracies to be ratified. The autoencoder should have learned more information about the structure and shape of the expected building masks.By passing this information into the FCN through an auxiliary branch, the segmentation maps could be refined by predicting more regular shaped buildings. However, as seen in the final test results for the combined architecture, the addition of the decoder did not improve the results over the baseline FCN. The reason for this is because the autoencoder is providing a reconstruction that is too lossy to be useful, even with the new architecture (Figure 7.4) and reduced loss. By feeding the FCN with the auxiliary branch with too much loss, the segmentation network will not learn the rigid structure of the buildings to improve IoU accuracy over the baseline FCN. By using a smaller number of layers or more filters, it may have been possible to reconstruct a more accurate representation of the original image. However, by reducing the number of layers, there is a chance that not enough latent useful information will be learned at the intermediate stage. In the original paper [3], the authors primarily reconstructed single large objects such as a plane or car. However, in this task, lots of smaller and nearby buildings had to be reconstructed while preserving the buildings shape. This is a more complex task and will cause problems when trying to learn a useful intermediate representation. In the future and with a more in-depth architectural search, it may have been possible to re-

duce the loss even more than in the new architecture. This, in turn, is likely to have improved the IoU accuracy of the combined model.

8.2 Future Work

In the future, there are a number of ways to extend this project. One approach is to move away from hypercolumns. These offer a simple method to provide localisation. Instead a more complex multigrid architecture [31] could be used, proposed by some of the same authors who came up with the regularisation technique used in this paper [3]. This multigrid architecture would allow the layers in the network to operate and manipulate representations on a pyramid of grids, rather than just having a single grid space, as was used in this report. This multigrid architecture forms spatial pyramids in which network messages can be passed across, this allows context to be rapidly integrated into the architecture as the receptive field will grow exponentially with the depth of this pyramid. An example of this pyramid structure can be viewed in Figure 8.1, taken from the original paper [31]. Pooling and upsampling techniques now work on a multigrid level, reducing the size of the whole pyramid with each operation.

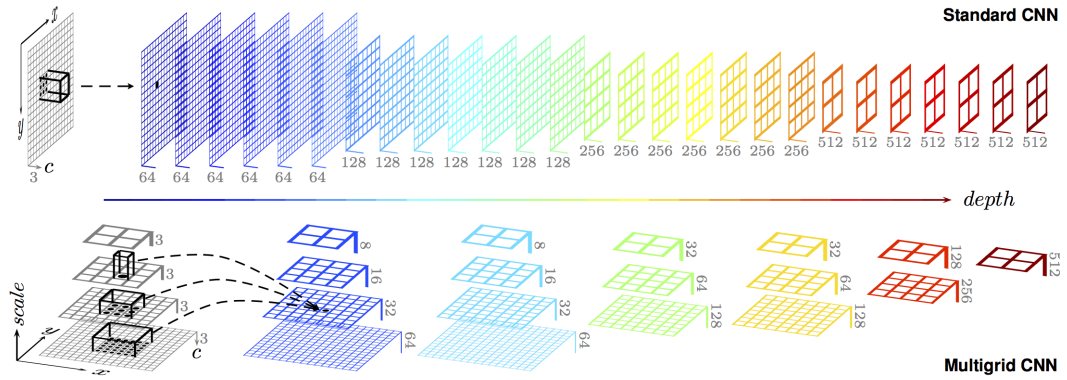


Figure 8.1: An example of a multigrid network (Bottom) [31], compared to regular CNN (Top).

It would be interesting to try a different type of model, called Polygon-RNN [32]. This architecture no longer models the task as a pixel prediction task but instead models it as a polygon prediction task. Given that the SpaceNet dataset consists of polygons in a GeoJSON format, there is potential to predict these polygons directly instead of first overlaying them, converting them to images and then creating binary masks. This model uses a Recurrent Neural Network (RNN). A RNN is different to a CNN as it uses an internal memory and can process sequences of inputs, instead of 1 input at a time (like a CNN). The Polygon-RNN can make use of this architecture by taking crops of an image as input and use them to sequentially produce output in the form of polygon vertices - outlining the buildings in the image. In this paper, a human must draw a box around an object that they want to generate a closed polygon for, as seen in Figure 8.2, taken from the original paper [32]. Therefore, the Polygon-RNN model would have to be extended for this task, as it does not make use of human annotators. This may require some of the data augmentation techniques described in Section 5.2 to help

isolate buildings into single images that could be annotated in this way. As no square box could be drawn around the buildings on the satellite image, a method would need to be developed to extract buildings automatically. Perhaps this could be modelled as a two-phase approach where the building masks produced by the FCN mentioned in this report are used as the cropping boundaries before being passed into the Polygon-RNN as input. The Polygon-RNN would then be able to refine these masks and produce a more tight/closed polygon, reducing the incorrectly classified pixels along building boundaries and consequently improving the IoU score.

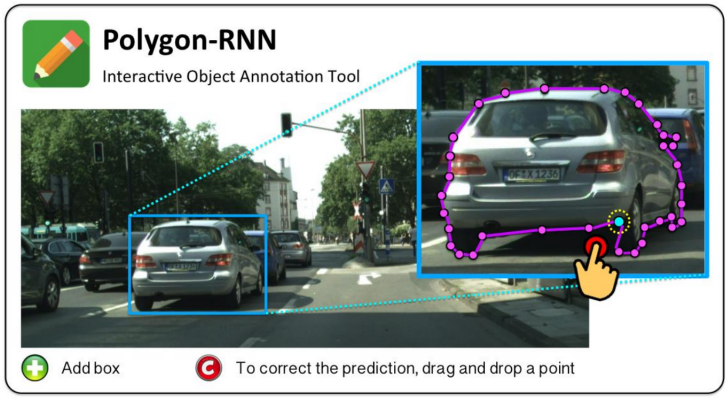


Figure 8.2: An example of a box drawn by a human and the resultant closed polygon formed by the RNN [32].

Bibliography

[1] Earth Observation Satellites in Space. Pixalytics. https://www.pixalytics.com/eo-satellites-in-space-2018/?format=pdf. Accessed on 2019-02-24.

[2] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *CVPR 2015*, 2015.

[3] Mohammadreza Mostajabi, Michael Maire, and Gregory Shakhnarovich. Regularizing Deep Networks by Modeling and Predicting Label Structure. In *CVPR*, 2018.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[5] Tingwu Wang. Semantic segmentation - university of toronto computer science, 2018. URL http://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf.

[6] Md. Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Mahmudul Hasan, Brian C. Van Esesn, Abdul A. S. Awwal, and Vijayan K. Asari. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *CoRR*, abs/1803.01164, 2018.

[7] Arden Dertat. Applied Deep Learning - Part 3: Autoencoders. Medium. Accessed on 2019-03-02.

[8] Harold Charles Daumé. *Practical Structured Learning Techniques for Natural Language Processing*. PhD thesis, University of Southern California, 8 2006.

[9] Sanjeevan Shrestha and Leonardo Vanneschi. Improved fully convolutional network with conditional random fields for building extraction. *Remote Sensing*, 10: 1135, 2018.

[10] Anurag Arnab, Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Måns Larsson, Alexander Kirillov, Bogdan Savchynskyy, Carsten Rother, Fredrik Kahl, and Philip H. S. Torr. Conditional Random Fields Meet Deep Neural Networks for Semantic Segmentation: Combining Probabilistic Graphical Models with Deep Learning for Structured Prediction. *IEEE Signal Processing Magazine*, 35:37–52, 2018.

[11] Building Detectors. github.com/SpaceNetChallenge/BuildingDetectors/tree/master/wleite, 2017.

[12] Jordan M. Malof, Leslie M. Collins, Kyle Bradbury, and R. G. Newell. A deep convolutional neural network and a random forest classifier for solar photovoltaic array detection in aerial imagery. *2016 IEEE International Conference on Renewable Energy Research and Applications (ICRERA)*, pages 650–654, 2016.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[14] Bharath Hariharan, Pablo Andrés Arbeláez, Ross B. Girshick, and Jitendra Malik. Hypercolumns for Object Segmentation and Fine-grained Localization. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 447–456, 2015.

[15] SpaceNet on Amazon Web Services (AWS). https://spacenetchallenge.github.io/datasets/datasetHomePage.html. Accessed on 2018-10-20.

[16] QGIS Documentation. https://www.qgis.org/en/docs/index.html.

[17] GeoPandas docs. http://geopandas.org.

[18] rasterio docs. https://rasterio.readthedocs.io/en/stable/.

[19] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Introduction to Data Mining. 2005.

[20] Maxim Berman, Amal Rannen Triki, and Matthew B. Blaschko. The lovasz-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4413–4421, 2018.

[21] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan Loddon Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2015.

[22] Bharath Raj. Data augmentation — how to use deep learning when you have limited data, 2018. URL https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-dat\\a-part-2-data-augmentation-c26971dc8ced.

[23] Alexander V. Buslaev, Alex Parinov, Eugene Khvedchenya, Vladimir I. Iglovikov, and Alexandr A. Kalinin. Albumentations: fast and flexible image augmentations. *CoRR*, abs/1809.06839, 2018.

[24] Vladimir I. Iglovikov, Sergey Mushinskiy, and Vladimir Osin. Satellite imagery feature detection using deep convolutional neural network: A kaggle competition. *CoRR*, abs/1706.06169, 2017.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[26] Leonid Kozinkin Mikhail Karchevskiy, Insaf Ashrapov. Automatic salt deposits segmentation: A deep learning approach. *ArXiv e-prints*, 2018.

[27] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *CoRR*, abs/1804.07612, 2018.

[28] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489, 2017.

[29] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[31] Tsung-Wei Ke, Michael Maire, and Stella X. Yu. Multigrid neural architectures. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4067–4075, 2017.

[32] Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Annotating Object Instances with a Polygon-RNN. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4485–4493, 2017.