Application of Convolutional Neural Networks to Building Segmentation in Aerial Images

by

Kayode Kolawole Olaleye

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Mathematics in the Faculty of Science at Stellenbosch University

Supervisor: Dr. Yabebal Fantaye

December 2018

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: ......December 2018..........

Abstract

Application of Convolutional Neural Networks to Building Segmentation in Aerial Images

Kayode Kolawole Olaleye

Department of Mathematical Sciences,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MSc

December 2018

Aerial image labelling has found relevance in diverse areas including urban management, agriculture, climate, mining, and cartography. As a result, research efforts have been intensified to find fast and accurate algorithms. The current state-of-the-art results in this context have been achieved by deep convolutional neural networks (CNNs). This has been possible because of advances in computing technologies such as fast GPUs and the discovery of optimal architectures. One of the main challenges in using deep CNNs is the need for a large set of ground truth labels during the training phase. Moreover, one has to choose optimal values for the many hyperparameters involved in the model construction to get a good result. In this thesis we focus on building segmentation from aerial images, and study the effect of different hyperparameter values, paying particular attention to the generalisation ability of the resulting models. For all our experiments we use the same architecture and performance metric as the one used in Mnih & Hinton (2012). Our investigation found the following main results: 1) when it comes to the size of CNN filters, small size filters perform as good or even better than large sized filters; 2) the LeakyReLU activation functions lead to a better precision-recall curve than ReLU (Rectified Linear unit) and Tanh activation functions; 3) batch-normalization leads to a slightly poor break-even point than without batch-normalization - this is contrary to what has been found in other studies with different architectures. In addition, we also investigate how well our models generalise to the task of interpreting contexts that are different from the training sets. Drawing from our findings, we gave recommendations on how to make deep CNN models more robust to variations in aerial images of other continent such as Africa where annotations are either unavailable or in short supply.

Uittreksel

Toepassing van Konvolutionele Neurale Netwerke om Segmentasie in Lugfoto's te bou

("Application of Convolutional Neural Networks to Building Segmentation in Aerial Images")

Kayode Kolawole Olaleye

Departement Wiskuudige Wetenskappe,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MSc

Desember 2018

Lugfoto-etikettering het relevansie gevind in verskeie gebiede, insluitende stedelike bestuur, landbou,klimaat, mynbou en kartografie. As gevolg hiervan is navorsingspogings versterk om vinnige en akkurate algoritmes te vind. Die huidige state-of-the-art resultate in hierdie konteks is bereik deur diep konvolusie neurale netwerke (CNNs). Dit is moontlik as gevolg van vooruitgang in rekenaar tegnologie soos vinnige GPU's en die ontdekking van optimale argitektuur. Een van die grootste uitdagings in die gebruik van diep CNN's is die behoefte aan 'n groot aantal grondwaarheidetikette gedurende die opleidingsfase. Daarbenewens moet mens optimale waardes kies vir die baie hiperparameters wat by die modelkonstruksie betrokke is om 'm goeie resultaat te kry. In hierdie proefskrif het ons fokus op die bou van segmentering van lugfoto's en bestudeer die effek van verskillende hiperparameterwaardes, met spesiale aandag aan die veralgemeningsvermoe van die gevolglike modelle. Vir al ons eksperimente gebruik ons dieselfde argitektuur en prestasiemetriek as die een wat in Mnih en Hinton (2012) gebruik word. Ons ondersoek het die volgende hoofresultate gevind: 1) As dit by die grootte van CNN-filters kom, doen klein grootte filters so goed of selfs beter as groot grootte filters; 2) die LeakyReLU aktiverings funksies lei tot ′n beter presisie-herhalingskromme as ReLU (reggestelde lineere eenheid) en Tanh aktiverings funksies; 3) batch-normalsering lei tot ′n effens swak gelykbreekpunt as sonder batch-normalisering dit is strydig met wat in ander studies met verskillende argitekture gevind is. Daarbenewens ondersoek ons ook hoe goed ons modelle veralgemeen in die interpretasie van kontekste wat verskil van die opleidingsstelle. Op grond van ons bevindinge, het ons aanbevelings gegee oor hoe om diep CNN-modelle sterker te maak vir variasies in lugfoto′s van ander vastelande soos Afrika waar annotasies of onbeskikbaar of in gebreke is.

Acknowledgements

My heartfelt appreciation goes to my supervisor, Dr. Yabebal Fantaye for providing valuable feedback and support in the course of my research as well as making sure I have all the necessary tools required to carry out my research.
I want to appreciate the Cosmology Group of AIMS for accommodating me in the group and providing me with valuable feedbacks.
AIMS is a cozy environment for research - special thanks to the administration and logistics units as well as the entire AIMS community for the wonderful experience.
A special thanks to the South Africa's Centre for High Performance Computing (CHPC) for providing access to the CPU and GPU clusters to run my experiments.
I want to specially thank my family for their prayers, support and understanding.
Above all, my deepest appreciation is to Almighty God for seeing me through this phase - Thank you!

Dedications

To my family, mentors, and friends

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Satellites orbiting the earth capture important information about different parts of the world, making available an enormous amount of images that can be tapped to provide insights into the dynamics impacting the earth and its inhabitants. Such data can serve as an alternative to infering the urban dynamics, for example for helping African cities optimize limited resources. In particular, remotely sensed data provides near-real-time support to farmers, urban-planning specialists, and cartographers. One of the ways this is made possible is through creating systems designed to delineate objects of interest from a massive amount of remotely sensed images in an automatic fashion. Advances in deep learning architectures and hardware such as GPUs have contributed immensely to the conversion of image data to high impact products and services. Before their take-off, remote sensing practitioners rely on hand-engineered computer vision techniques for insights extraction from these data. Hand-engineered approaches require human expertise, rendering the process error-prone, slow, and expensive. Convolutional Neural Networks (CNN) - a class of neural networks designed to perform well with data that has a grid-like topology (for example images) - eliminates the need for hand-engineered discriminative feature extraction. These techniques have continuously shown human-level performance and outperform many computer vision techniques on diverse real-life tasks.

The availability of the OpenStreetMap (OSM) [1] project - a crowdsourcing project for creating a free editable map of the world - contributes to the success of CNN in the task of aerial image labelling. CNNs thrive under rich

[1]https://www.openstreetmap.org

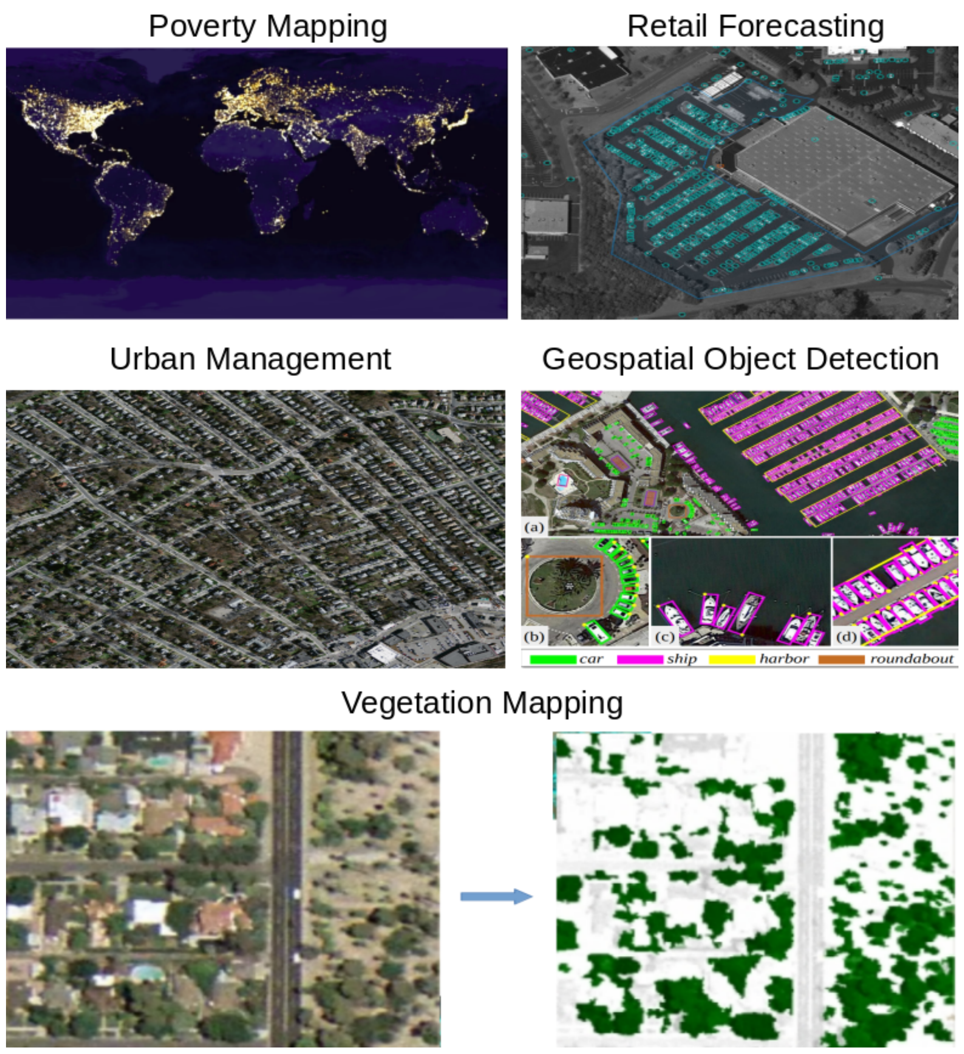


Figure 1.1: Some applications of CNNs to aerial imagery. Neal et al. [31], Gui-Song et al. [74], Volodymyr [50], DigitalGlobe [15], Walker et al. [70]

supervision and abundance of data. While ground truth available on OSM is fraught with human-error, a number of research results in this domain are based on it. Some of them ([51, 50, 61]) devise means of mitigating the effect of such error on the resulting trained models.

The goal of this thesis is to investigate the generalization capability of CNN models trained on the Masachusset Building Dataset [50] and identify a set of hyper-parameters that are particularly well-suited for the task of labelling aerial images. This thesis focuses on:

- **Generalisation Potential of CNN models**: A possible limitation of the application of existing CNN models for aerial image labelling is that they are trained on a limited amount of data covering few regions in North America. In this thesis, we will develop CNN-based models using existing ideas from the literature and evaluate such models on aerial images from another continent. The aerial images used in the investigation of the generalization capability of the models are not included in the training sets.

- **Empirical analysis of different hyper-parameters choices**: When training CNN, finding good hyper-parameter choices determines the performance of the network. Some of the key hyper-parameters that influence the performance of CNN include the number of convolution layers, number of hidden filters, filter size, normalizing or renormalizing the output of each convolution layers, learning rate and activation functions to use. Additionally, it might be useful to understand how different choices of setting up dataset pipeline in terms of the amount of training, validation and test sets impact the efficiency of the experimentation.

The rest of the thesis is presented as follows:

- Chapter 2 provides a review of aerial datasets and existing methods for analysing them.
- Chapter 3 presents the formulation of aerial image labelling task that we use as well as details about the choice of CNN architecture and parameters investigated in this work.
- Chapter 4 presents the models we investigated and the generalisation potential of the models.
- Chapter 5 presents a conclusion for this thesis and highlights some of the possible future directions.

Chapter 2

Review

Several methods for learning from vision datasets have been developed over the years. The methods are generally classified into image classification [65], object localization and detection [25], semantic segmentation [41], and instance segmentation [69]. These methods have been extended to the task of labelling objects of interest in satellite and aerial imagery over the past decades. Throughout the rest of this thesis, we use aerial imagery to refer to any type of two dimensional and possibly multi-band data collected by an airborne or spaceborne sensor. The sensors could be visible light, infrared, hyperspectral sensors, as well as airborne LIDAR, which measures the distance to objects from the sensor [50]. The labelling of aerial images is useful for a wide range of applications such as geospatial object detection and segmentation [23, 44, 51, 50, 46, 3, 61], land use and land cover analysis [48, 58, 6], natural disaster detection [47, 7, 66], and vegetation mapping [71, 57, 49, 36].

The origin of public aerial image for machine learning training could be traced to the cassification datasets [77, 63, 83, 80, 73]. These datasets consist of high-resolution aerial images divided into object classes, where images are assigned to a class based on the most noticeable object in the image. Available classes include but not limited to buildings, forest and roads.

Classification of aerial images [6, 56, 54, 73, 77, 81, 79, 78] consists of making a prediction for a whole input image, i.e., predicting the probability that a given input image belong to a given object class. The drawback of classification algorithms is that the location of the identified objects in the image are not known.

Localization and detection techniques [4, 57, 36, 5, 16, 71, 66], on the other hand, are designed to learn the bounding boxes around the identified objects in the image to extract the location information. Semantic segmentation [51, 56, 26] takes a further step towards a fine-grained localization of objects of interest. It entails grouping parts of images so that each pixel in a group corresponds to the object class of the group as a whole. In a binary class setting, the dataset contains two distinct classes: a class for the object of interest and another class for the background. A multi-class setting is characterised by more than two classes. In particular, the goal of semantic segmentation is to predict the class of each pixel in the input aerial image.

One of the reasons for the increasing popularity of research in this field is the recent advances in remote sensing instruments that has made it possible to generate more and more different types of aerial images with different spatial, spectral and temporal resolutions. Another reason is the advances of image processing algorithms, particularly CNNs. The consistent success of CNNs in achieving super-human results in computer vision tasks has inspired researchers in the field to adopt them as an algorithm of choice for aerial images classification tasks. The availability of powerful machines like GPUs that make it possible to train very deep CNNs for classification tasks further fuels the growing interest.

In aerial imagery, three types of resolution are common: the spatial resolution, the spectral resolution, and the temporal resolution. The spatial resolution encodes the visible details about the pixel space, spectral resolution specifies the electromagnetic bands in the image, and the temporal resolution specifies the revisit time period of the satellite, airplane or drone.

Publicly available aerial images are of low resolution; starting from 10 metres per-pixel and hence not suitable for most real-life applications. The high-resolution imagery is made available for a fee by private aerial images companies, at less than 0.5 metres per-pixel.

A study of existing methods for dealing with aerial images is performed in [8] and information about existing dataset is provided. However, none of the datasets is suitable for a semantic segmentation task (More details in Section 2.1). In 2013, [50] released a benchmark dataset for training and evaluating deep networks on aerial dataset.

In the rest of this chapter we review the publicly available datasets for the study of classification, detection and segmentation tasks. Additionally, we discuss methods that are commonly applied to the datasets for a given application as well as some of the known performance metrics.

2.1 Aerial Image Datasets

Publicly available datasets in this field of research can be classified into three categories: **classification**, **detection**, and **segmentation** datasets. In the following subsections, some of the properties, use-cases and drawbacks of these datasets are presented. A summary of the presentation is provided in Table 4.3.

2.1.1 Classification Datasets

UC Merced Land-Use

The widely used [6, 56, 54, 73, 77] UC Merced land-use dataset [77] is composed of 2100 image patches divided into 21 classes. Each class has 100 images of size 256 $\times$ 256 pixels with a spatial and spectral resolution of 0.3 metres and RGB respectively. Images were manually extracted from the United States Geological Survey (USGS) National Map and covers several US regions. The issues with this data are the small-scale size of the dataset and unavailability of object contours, hence, not suitable for a semantic segmentation task. However, due to their nature and relatively high resolution, these images share many low-level features with general-purpose optical images making them good candidates for fine-tuning a pre-trained CNN [6].

WHU-RS19

The WHU-RS19 dataset [63] is composed of 19 scene classes, including airport, beach, bridge, commercial area, desert, farmland, football field, forest, industrial area, meadow, mountain, park, parking lot, pond, port, railway station, residential area, river, and viaduct. It was extracted from a set of aerial images exported from Google Earth with a spatial and spectral resolution of 0.5m and RGB. Each class has about 50 images of size 600 $\times$ 600

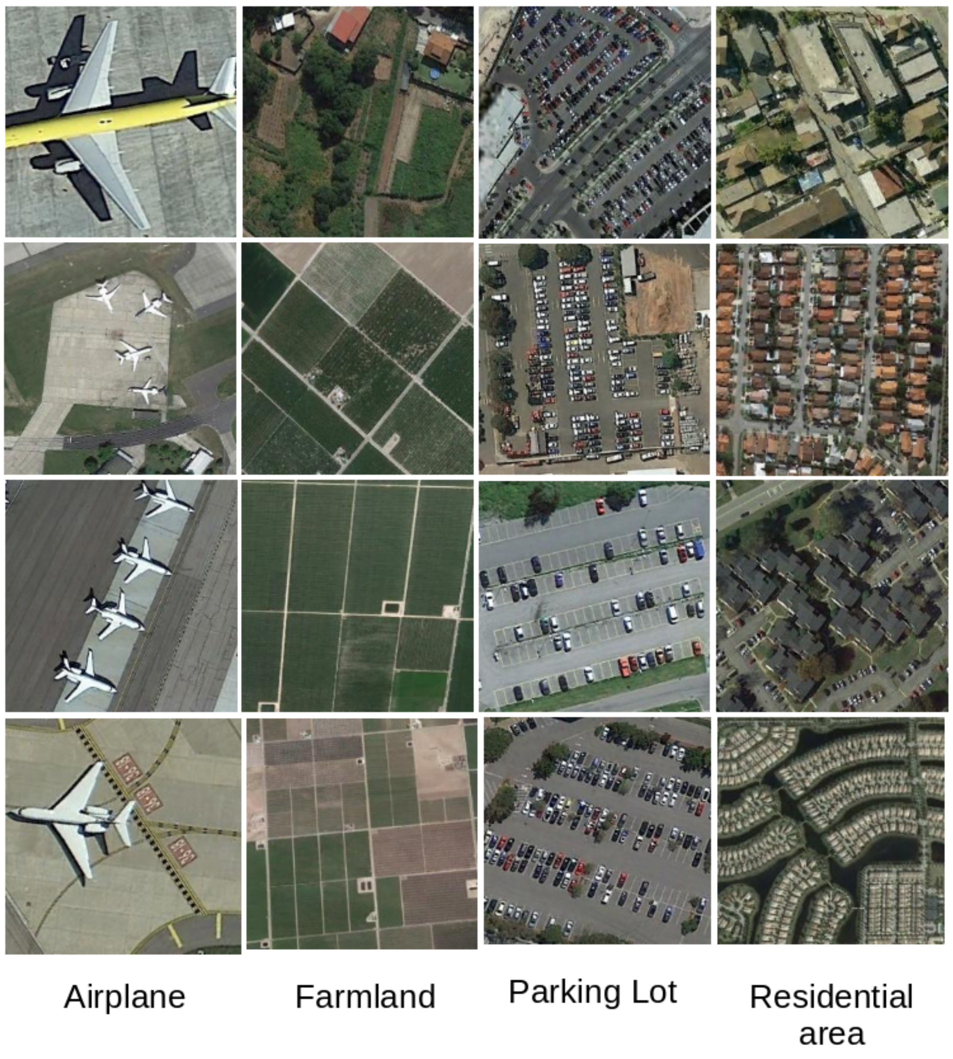


Figure 2.1: Samples of classification dataset from NWPU-RESISC45 Dataset [8].

pixels. A drawback to this dataset is that per-pixel labelling is not available, hence, not suitable for a semantic segmentation task. The high variations in resolution, scale, orientation, and illuminations of the images further make the data less attractive for some applications. The number of images per class of this dataset is relatively small compared with UC-Merced dataset [77]. However, it has also been widely adopted to evaluate scene classification methods [10, 75, 82, 11, 27].

SIRI-WHU

The SIRI-WHU dataset [78] is composed of 12 scene classes including agriculture, commercial, harbor, idle land, industrial, meadow, overpass, park, pond, residential, river, and water. Each class consists of 200 Google Earth images with a spatial resolution of 2 m and a size of 200 $\times$ 200 pixels. While this dataset has been tested by several methods [81, 79, 78], the number of scene classes is relatively small and per-pixel labelling is not available. Moreover, it mainly covers urban areas in China and hence lacks diversity.

RSSCN7

The RSSCN7 dataset [83] is a 7-class images collected from the Google Earth including grassland, forest, farmland, parking lot, residential region, industrial region, and river/lake. Each class consists of 400 images that are cropped on four different scales with 100 images per scale. Each image has a size of 400 $\times$ 400 pixels. A drawback to this dataset, aside from the absence of per-pixel labels, comes from the scale variations of the images [72] but can be used to fine-tune pretrained networks.

RSC11

The RSC11 dataset [80], collected from Google Earth, is composed of 11 classes including dense forest, grassland, harbor, high buildings, low buildings, overpass, railway, residential area, roads, sparse forest, and storage tanks. Each class consists of 100 images with size 512 $\times$ 512 pixels and a spatial resolution of 0.2 metres. A challenge with this dataset is that the classes are quite similar in vision, which increases the difficulty in classifying the scene images. RSC11 is not suitable for segmentation task.

Aerial Image Dataset (AID)

AID [73] is a new large-scale aerial image dataset created by collecting sample images from Google Earth imagery. It has 30 different scene classes and about 200 to 400 samples of size 600 $\times$ 600 in each class. The images in AID are multi-source, as Google Earth images are from different remote imaging sensors. This brings more challenges for scene classification than the single source images like the UC-Merced dataset. Moreover, all the sample images per class in AID are carefully chosen from different countries and regions around the world, namely China, the United States, England, France, Italy, Japan, Germany; and they are extracted at different time and seasons under different imaging conditions, which increases the intra-class diversity of the data.

NWPU-RESISC45

NWPU-RESISC45 dataset [8] is a 45-class publicly available benchmark for REmote Sensing Image Scene Classification (RESISC), created by Northwestern Polytechnical University (NWPU), China. The classes include airplane, airport, baseball diamond, basketball court, beach, bridge, chaparral, church, circular farmland, cloud, commercial area, dense residential, desert, forest, freeway, golf course, ground track field, harbor, industrial area, intersection, island, lake, meadow, medium residential, mobile home park, mountain, overpass, palace, parking lot, railway, railway station, rectangular farmland, river, roundabout, runway, seaice, ship, snowberg, sparse residential, stadium, storage tank, tennis court, terrace, thermal power station, and wetland. This dataset contains 31500 images with spatial resolution of 0.3 to 0.2 metres has 700 images in each class. This dataset is not suitable for segmentation task but can serve as a good candidate for fine-tuning pretrained networks.

Datasets and Reference	Images per class	Classes	Total images	Spatial resolution (metres)	Image Size	Year
UC Merced Land-Use [77]	100	21	2100	0.3	256 × 256	2010
WHU-RS19 [63]	50	19	1005	up to 0.5	600 × 600	2012
SIRI-WHU [78]	200	12	2400	2	200 × 200	2016
RSSCN7 [83]	400	7	2800	-	400 × 400	2015
RSC11 [80]	100	11	1232	0.2	512 × 512	2016
AID [73]	400	30	10000	-	600 × 600	2016
NWPU-RESISC45 [8]	700	45	31500	0.3 to 0.2	256 × 256	2016
DOTA [74]	-	15	2806	-	800 × 800 to 4000 × 4000	2018
NWPU VHR-10 [38]	-	3	-	-	-	2016
COWC [52]	-	2	-	0.15	-	2016
Massachusetts Buildings Dataset [50]	-	2	151	1	1500 × 1500	2013
ISPRS (Potsdam and Vaihingen) [30]	-	6	-	(0.05, 0.09)	6000 × 6000	2012
Inria [45]	-	2	-	0.3	5000 × 5000	2012

Table 2.1: Comparison between the different RGB aerial datasets classification, detection and segmentation.

2.1.2 Object Detection Datasets

COWC

COWC [52] is extracted from 6 distinct locations: Toronto Canada, Selwyn New Zealand, Potsdam and Vaihingen Germany, Columbus and Utah United States. It has a spatial and spectral resolution of 15cm and RGB-grayscale respectively. It contains 32716 annotated cars and 58247 negative examples It is specifically created for car detection and counting, making its application limited in scope.

NWPU VHR-10

NWPU VHR-10 [38] is an aerial image containing 3 categories of objects including vehicles, ships and airplanes collected from different locations on Google Earth. Specifically, the vehicles (recently 12000 instances) are collected from an urban area in Beijing, China. The ships (3000 instances) are collected from near the wharfs and ports besides the Changjiang River, the Zhujiang River, and the East China Sea. The airplanes (2000 instances) are collected from the images of 15 airports in China and America. Information about the size and location of image bounding box are available, hence a good candidate for training an object detector.

DOTA

DOTA [74] is a large-scale Dataset for Object deTection in Aerial images. It can be used to develop and evaluate object detectors in aerial images just like Pascal-VOC [18] is used in general object detection. It contains 2806 aerial images from different sensors and platforms. Each image ranges from about 800×800 to 4000×4000 pixels. It currently includes 15 common object categories including plane, ship, storage tank, baseball diamond, tennis court, basketball court, ground track field, harbor, bridge, large vehicle, small vehicle, helicopter, roundabout, soccer ball field and swimming pool. Spatial resolution for each image is provided. Spatial resolution implies the actual size of an instance. Fully annotated DOTA images contain 188282 instances. It is specifically created for multi-class object detection tasks in

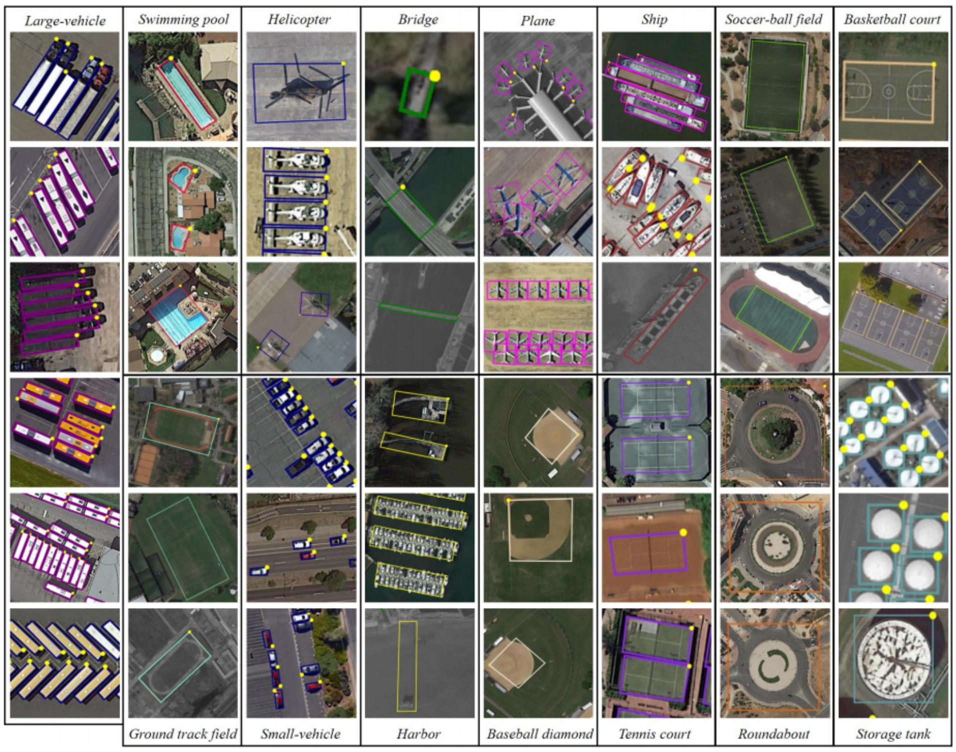


Figure 2.2: Samples of detection dataset from DOTA [74].

aerial images. Currently, this dataset is still of a smaller scale compared to Pascal-VOC.

2.1.3 Segmentation Datasets

Massachusetts Buildings Dataset (MBD)

The MBD consists of 151 aerial images of the Boston area, with each of the images being 1500 $\times$ 1500 pixels for an area of 2.25 square kilometers. The entire dataset covers roughly 340 square kilometers. A training set of 137 images, a test set of 10 images and a validation set of 4 images are randomly generated from the data. The dataset covers mostly urban and suburban areas and buildings of all sizes, including individual houses and garages, are included in the labels. Figures 6.1(a) and 6.1(b) show two representative regions from the Massachusetts Buildings dataset. While this dataset provides building contours suitable for training a segmentation system, it captures a limited number of building structure and types which might make any network trained on the dataset struggle in labelling buildings on aerial images of other types. Another issue with this dataset is its limited scope of application as it provides contours for only one object class.

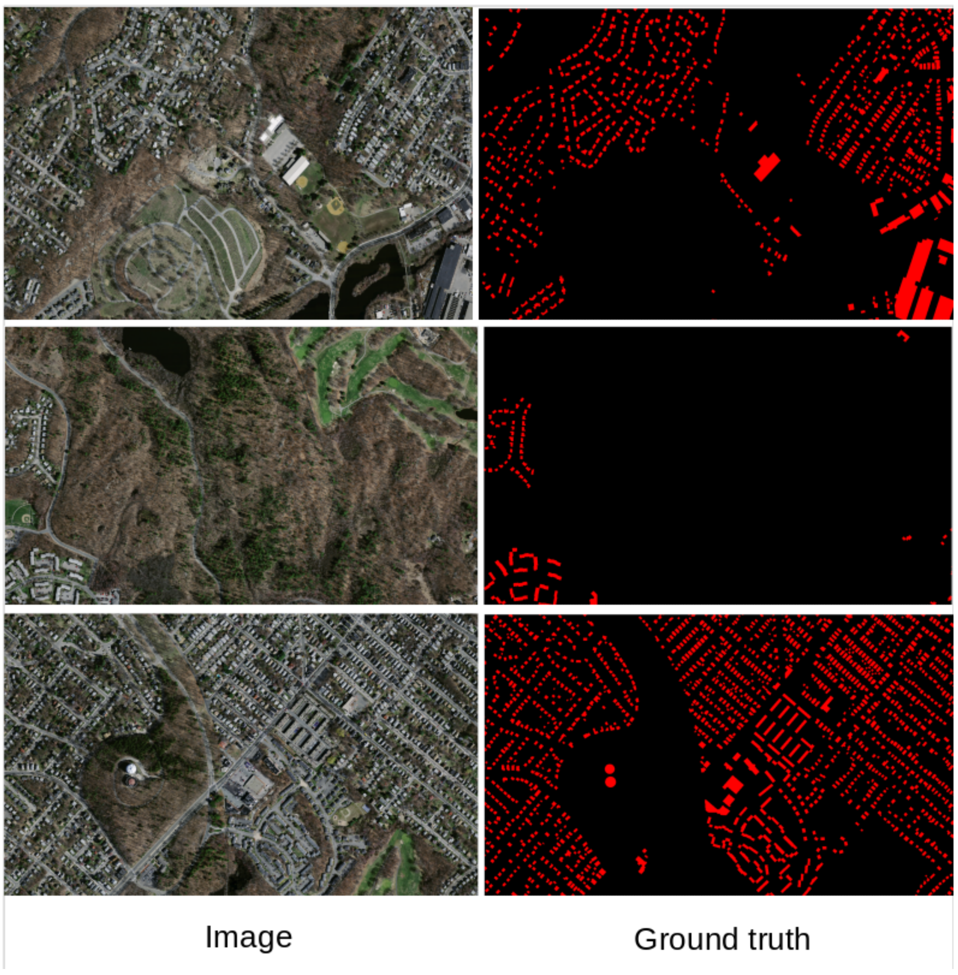


Figure 2.3: Samples of segmentation dataset from MBD [50].

Inria (Aerial Image Labelling) Dataset

Inria dataset was constructed by combining public domain imagery and public domain official building footprints. The dataset has the following features: a coverage of 810 square kilometers - 405 square kilometers each for training and testing), a spatial resolution of 0.3 metres, and ground truth data for two semantic classes: building and not building (publicly disclosed only for the training subset). The images cover dissimilar urban settlements, ranging from densely populated areas (e.g., San Francisco's financial district) to alpine towns (e.g,. Lienz in Austrian Tyrol). The drawbacks of this dataset are similar to MBD's.

International Society for Photogrammetry and Remote Sensing (ISPRS) Datasets

ISPRS provided two state-of-the-art aerial images datasets - Vaihingen and Potsdam - consisting of very high resolution true orthophoto (TOP) tiles and corresponding digital surface models (DSMs) derived from dense image matching techniques. Both datasets cover urban scenes. While Vaihingen is a relatively small village with many detached buildings and small multi story buildings, Potsdam shows a typical historic city with large building blocks, narrow streets and dense settlement structure. Each dataset is classified manually into six most common land cover classes: Impervious surfaces, Building, Low vegetation, Tree, Car, Clutter/background. The groundtruth for the test sets are not publicly available which makes the evaluation of models trained on the training sets a bit challenging.

2.2 Machine Learning (ML) Methods for Aerial Image Labelling

ML is a subfield of artificial intelligence. The goal of ML is to enable computers to learn on their own which is in contrast with traditional computational approaches where algorithms are explicitly programmed. ML tasks are generally classified into **supervised, unsupervised** and **reinforcement learning**. In supervised learning, an algorithm is trained using parallel input data, x and desired output, y. During training, the algorithm learns (in an iterative fashion) a mapping, f from x to y ($y = f(x) + \epsilon$ where ϵ - an arbitrary small positive quantity - accounts for the inherent noise present in the input). A loss function that computes the discrepancy between y and $f(x)$ is defined and the goal of the training is to minimise the loss function. Supervised learning tasks are further divided into **classification**, and **regression**. In classification, the variable y is the class label of x chosen from two classes (binary classification) or more classes (multiclass classification). In regression, y is a continuous/real value (e.g. the price of a commodity). Examples of supervised learning algorithms include: nearest neighbours, support vector machines (SVM), and artificial neural networks (ANNs) etc. However, some of these algorithms are not exclusive to supervised learning as they can be used as unsupervised learning algorithm as well. In particular, ANN - a deep learning (DL) algorithm - can serve to classify data, or discover interesting structure in data. DL is a subfield of ML that attempts to imitate how the human brain processes signals. A deep learning architecture is inspired by biological neural networks and consists of one or more nonlinear hidden unit layers performing a transformation from the input space to the output space. ANNs' human-level performance on a number of real life tasks make them the most prominent in the ML space. In this work, we used the ANNs variant called CNNs (see Subsection 2.2.5 for more details). In unsupervised learning, there is no y. The algorithms are fed with only the input x and the task is to discover interesting patterns in the data that can explain the underlying input data. In reinforcement learning an algorithm trains itself (in a trial and error fashion) through interactions with a dynamic environment in which it must perform a particular goal (such as playing a game with an opponent or driving a car).

The main goal of this section is to present how aerial image labelling has progressed over the years from simply assigning each image an object class to detecting the location of each object in the image and more recently, labelling each pixel in an image. The literature is currently dominated by methods based on the classification dataset because it is more readily available compared to detection and segmentation datasets. There is an ongoing effort to create more large-scale detection and segmentation datasets in the aerial images space that compare relatively well with the volume of datasets that exist in the general image space [50, 30, 45, 74]

Algorithms reviewed in this section are similar to those used for general image datasets such as ImageNet [14], Pascal-VOC [18] and Microsoft Common Objects in COntext (MSCOCO) [37]. These algorithms, developed to perform classification, detection and segmentation tasks have been extended to aerial datasets through different research efforts motivated by urban management, disaster monitoring, and agriculture. The performance of these algorithms on the respective dataset is highlighted.

2.2.1 Classification Methods

Table 2.2 presents a number of methods based on the different datasets described in Section 2.1.1. A comparison of different design modalities on UC Merced dataset is done in [6]. The authors motivated their use of pre-trained deep network (CaffeNet [32] and GoogleNet [68]) on the dataset by its relatively high resolution (0.3m) which makes it quite comparable to the low-level features of general-purpose images. The model obtained from fine-tuning the weights of a pre-trained GoogleNet generalizes better than model designed from scratch. They further highlighted that it is computationally cheaper to train such a model than it is to train a model from scratch (see Figure 2.4 the archtecture of their CaffeNet). A similar approach is adopted by [56]. State-of-the-art performance of their approach on the UC Merced dataset was obtained through the combination of two deep CNN models: OverFeat [62] and CaffeNet [32].

[54] compared state-of-the-art baselines on UC Merced dataset - SIFT [42] and Local Binary Pattern [55] - with deep CNN pre-trained on general-purpose image and fine-tuned on the target dataset. The classification of deep features extracted from the fine-tuned model with linear SVM achieved the highest accuracy (99.47 ± 0.50). [73] benchmarked different architectures on various data sets. They categorized the different methods into three groups: Low-level methods, mid-level methods and high-level methods. They described low-level methods as methods that often first divide aerial images into small patches, then use low-level visual features such as spectral, texture or structure information to group patches and finally output the distribution of the patch features as the scene descriptor. Mid-level methods build aerial images representation by encoding low-level feature descriptors and high-level methods encode the mid-level feature descriptors to provide a better aerial images representation.

2.2.2 Object Detection Methods

Table 2.2 presents a number of methods based on the aerial images object detection datasets presented in Subsection 2.1.2.[38] applied three different bounding box object detection approaches - Regions with CNN (R-CNN) [19], Single Shot Multibox Detector (SSD) [40], and Detection Rotatable bounding Box (DRBox) [38] - to NWPU VHR-10 dataset. R-CNN proposes few regions on the image to run the convnets on, rather than running sliding windows on every single region in the image. It then proceeds to classify each proposed region one at a time and outputs the label alongside

Method	Dataset	Accuracy	Approach
CNN [6]^{*classification}	UC Merced	97.10 ± 00	Pretrained CNN (CaffeNet) with fine-tuning on UC Merced dataset
CNN [56]^{*classification}	UC Merced	99.43 ± 0.27	Concatenation of the feature vectors computed by CaffeNet and OverFeat
CNN [54]^{*classification}	UC Merced	99.47 ± 0.50	Pretrained CNN (GoogleNet) with fine-tuning on the target dataset with linear SVM
SIFT [73]^{*classification}	UC Merced	32.10 ± 1.95	Scale-Invariant Feature Transform
LBP [73]^{*classification}	UC Merced	36.29 ± 1.90	Local Binary Patterns
SIFT [73]^{*classification}	WHU-RS19	27.21 ± 1.77	Scale-Invariant Feature Transform
LBP [73]^{*classification}	WHU-RS19	44.08 ± 2.02	Local Binary Patterns
SIFT [73]^{*classification}	RSSCN7	32.76 ± 1.25	Scale-Invariant Feature Transform
LBP [73]^{*classification}	RSSCN7	60.38 ± 1.03	Local Binary Patterns
SIFT [73]^{*classification}	AID	16.76 ± 0.65	Scale-Invariant Feature Transform
LBP [73]^{*classification}	AID	29.99 ± 0.49	Local Binary Patterns
BoVW [73]^{*classification}	UC Merced	75.52 ± 2.13	Bag-of-Visual-Word, SIFT
BoVW [73]^{*classification}	WHU-RS19	82.58 ± 1.72	Bag-of-Visual-Word, SIFT
BoVW [73]^{*classification}	RSSCN7	81.28 ± 1.19	Bag-of-Visual-Word, SIFT
BoVW [73]^{*classification}	AID	68.37 ± 0.40	Bag-of-Visual-Word, SIFT
BoVW [73]^{*classification}	UC Merced	78.12 ± 1.38	Bag-of-Visual-Word, LBP
BoVW [73]^{*classification}	WHU-RS19	75.89 ± 2.40	Bag-of-Visual-Word, LBP
BoVW [73]^{*classification}	RSSCN7	81.40 ± 1.09	Bag-of-Visual-Word, LBP
BoVW [73]^{*classification}	AID	64.31 ± 0.41	Bag-of-Visual-Word, LBP
CNN [74]^{*detection}	DOTA	10.59 ± 00	Single shot multibox detection oh oriented bounding boxes
CNN [74]^{*detection}	DOTA	26.79 ± 00	Regional Fully convolution network on oriented bounding boxes
CNN [74]^{*detection}	DOTA	21.39 ± 00	You only look once on oriented bounding boxes
CNN [74]^{*detection}	DOTA	10.94 ± 00	Single shot multibox detection oh horizontal bounding boxes
CNN [74]^{*detection}	DOTA	47.24 ± 00	Regional Fully convolution network on horizontal bounding boxes
CNN [74]^{*detection}	DOTA	39.20 ± 00	YOLO on horizontal bounding boxes
CNN [50]^{*segmentation}	MBD	91.50 ± 00	patch-based training of CNN
CNN [50]^{*segmentation}	MBD	92.11 ± 00	patch-based training of CNN + Conditional Random Field (CRF)
CNN [61]^{*segmentation}	MBD	92.30 ± 00	patch-based CNN with Maxout activation function and Dropout
CNN [46]^{*segmentation}	MBD	94.23 ± 00	Modifying and joining VGGNet with AlexNet and fine tuning on MBD

Table 2.2: Comparison between the different methods for classification, detection and segmentation of aerial images. Accuracy here is reported as defined by each paper.

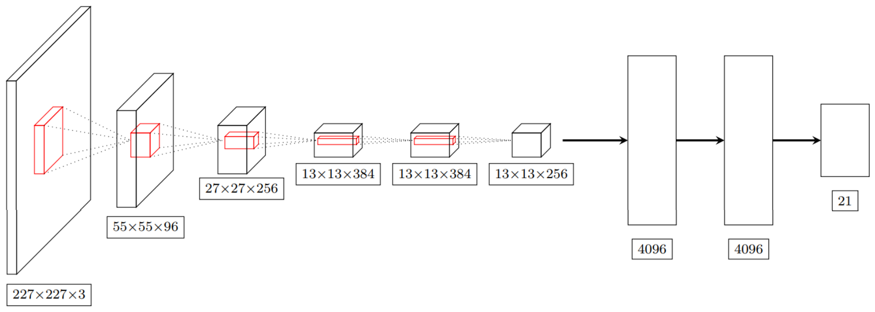


Figure 2.4: The CaffeNet architecture used in [6]. The first convolutional layer filters the $224 \times 224 \times 3$ input image with 96 kernels of size $11 \times 11 \times 3$ with a stride of 4 pixels. The second convolutional layer takes as input the output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each. The last layer of the network with a different fully-connected layer, with 21 outputs instead of 1000, followed by a softmax classifier. Maxpool layers are not shown. ReLU is applied to the output of every convolutional and fully connected layers.

the bounding box. SSD uses a single neural network and discards the idea of region proposal in R-CNN but instead uses different bounding boxes and then adjust the bounding box as part of the prediction. DRBox migrates from the use of bounding box and attempts to tackle the difficulties faced with locating objects with different orientations using rotatable bounding box (RBox). DRBox was found to have the best performance. Evaluation of state-of-the-art object detection algorithms (Faster R-CNN [60], R-FCN [12], YOLO [59] and SSD) on DOTA is investigated in [74] by proposing detection on horizontal and oriented bounding boxes. The authors established a benchmark for object detection in aerial images and show the feasibility to produce oriented bounding boxes by modifying a mainstream detection algorithm.

2.2.3 Segmentation Methods

In this thesis, we use semantic segmentation. In this section, we present an overview of the different deep learning based semantic segmentation

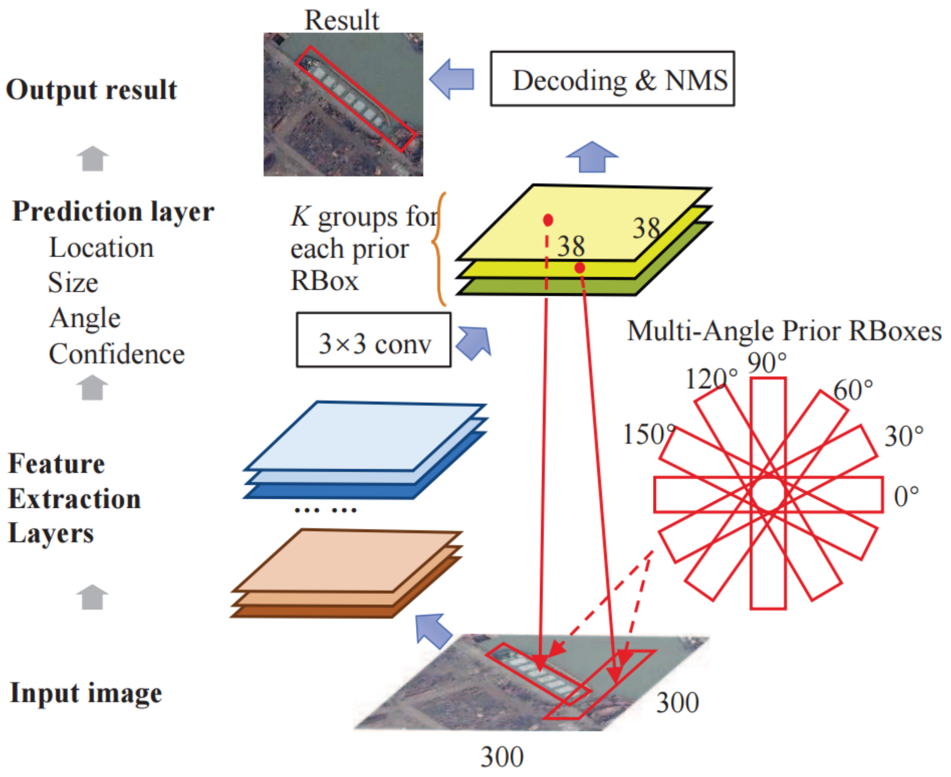


Figure 2.5: The architecture used in [38]. The input image is fed into the multi-layer CNNs to generate detection results. Features are extracted from the input image by the hidden convolution layers and the last convolution layer is for prediction. The prediction layer includes K groups of channels where K is the number of prior RBoxes in each position. For each prior RBox, the prediction layer output a confidence prediction vector indicating whether it is a target object or background, and a 5 dimensional vector which is the offset of the parameters between the predicted RBox and the corresponding predefined prior RBox. The decoding layer transforms the offsets to the exact predicted RBoxes. In the output layer, the predicted RBoxes are sorted with their confidence and passed through non-maximum suppression (NMS) to remove repeated predictions.

algorithms for aerial image labelling.

A patch-based semantic segmentation technique is proposed in [50, 51]. This approach entails training CNN using small squared patches, if necessary by cutting out patches of the same size with or without overlap from larger aerial images. [51] trained a deep neural network on aerial images and the corresponding groundtruths collected from crowdsourcing platforms such as OpenStreetMap. Other works in the literature that used a similar patch-based framework exist: [61] (see Figure 2.7) use CNN to learn a mapping from raw pixel values in aerial images to three object labels (buildings, roads, and others). The contribution is the extension of [50] to three classes namely: roads, buildings and others. [76] proposed using nighttime light intensities to supervise the training of fully-connected CNN to predict nighttime light from daytime imagery. They train their model to simultaneously learn poverty discriminative features using logistic regression classifier pre-trained on limited poverty data. The problem is set up as a combination of deep learning and transfer learning - a model trained on the ImageNet is modified and adapted to the task of predicting nighttime light intensity from daytime imagery. [46] (see Figure 2.6) proposes a dual-stream deep neural network model that processes information along two independent pathways. They train their model to combine local object appearance as well as information from the larger scene, in a complementary way, so that together, they form a powerful classifier. A similar approach is used by [44] for high-resolution semantic segmentation. In addition to learning features at different resolutions, the CNN framework learns how to combine these features (local and global) in an efficient and flexible manner. [23] identified two limitations in previous methods. First, the failure of feature representation technique to capture the spatial and structural patterns of objects and the background regions. Second, inadequacy and unreliability of training data with manual annotation; hence, proposed a method combining the use of unsupervised feature learning approach via Deep Boltzmann Machine (DBM) and a weakly supervised learning approach to object detection in optical aerial images where the training sets require only binary labels indicating whether an image contains the target object or not. [6] explored the use of pre-trained networks - CaffeNet [32] and GoogleNet [68] to highlight that training a deep CNN from scratch is not always advisable with limited sized datasets currently available in the field. The networks are

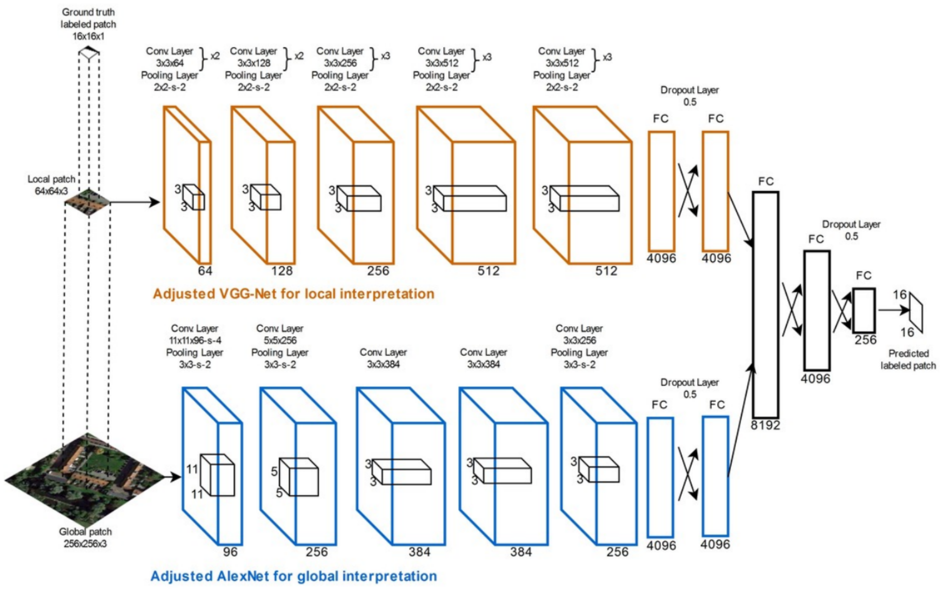

Ground truth labeled patch 16x16x1
Conv. Layer 3x3x64 Pooling Layer 2x2-s-2 x2
Conv. Layer 3x3x128 Pooling Layer 2x2-s-2 x2
Conv. Layer 3x3x256 Pooling Layer 2x2-s-2 x3
Conv. Layer 3x3x512 Pooling Layer 2x2-s-2 x3
Conv. Layer 3x3x512 Pooling Layer 2x2-s-2 x3
Dropout Layer 0.5
FC
FC
Local patch 64x64x3
64
128
256
512
512
4096
4096
FC
Dropout Layer 0.5
FC
FC
256
16
16
Predicted labeled patch
4096
8192
Adjusted VGG-Net for local interpretation
Conv. Layer 11x11x96-s-4 Pooling Layer 3x3-s-2
Conv. Layer 5x5x256 Pooling Layer 3x3-s-2
Conv. Layer 3x3x384
Conv. Layer 3x3x384
Conv. Layer 3x3x256 Pooling Layer 3x3-s-2
Global patch 256x256x3
96
256
384
384
256
4096
4096
Adjusted AlexNet for global interpretation

Figure 2.6: The architecture used in [46].

fine-tuned only on the target data to avoid overfitting and reduce training time. The experiment was performed on two aerial images datasets - [77] and [56].
These works outperform much of the earlier attempts that used ad-hoc and knowledge-based approach on the task of semantic segmentation [28, 2, 33].

The basic architecture of the CNN comprises stacked convolutional layers and spatial pooling layers often followed by one or more fully connected layers. The feature extraction over the input image is done in the convolutional layer which contains several convolution filters. This layer is followed by an optional pooling layer to make the network's output invariant to small translations in the input space. A non-linear activation function is applied to the output of each convolution to make the network learn non-linear patterns in the input dataset.

2.2.4 Image Preprocessing Techniques

Popular preprocessing techniques applied to images include: *data augmentation*, *dimensionality reduction*, *normalizing image inputs*, *mean and standard deviation of input data*, *uniform aspect ratio* and *image scaling*. These preprocessing techniques or a subset of them, depending on the ones necessary to apply on the underlying dataset, can improve the way the model learns

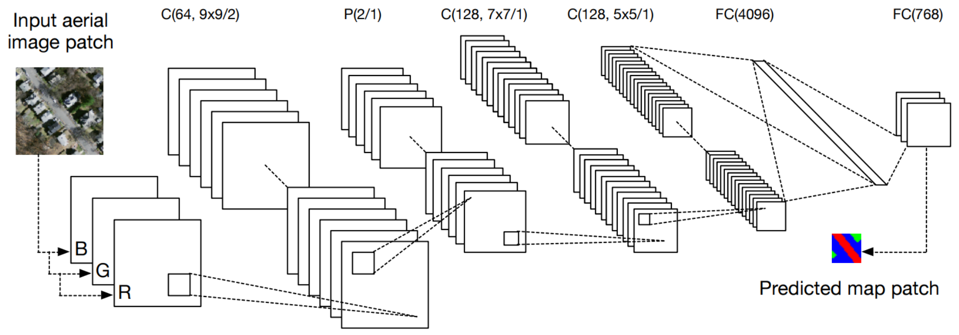


Figure 2.7: The architecture used in [61]. It consists of a convolution layer with 64 filters of size 9×9 with stride 1 followed by a maxout layer followed by a 2×2 max pooling with stride 1. The next layer is a convolutional layer with 128 filters of size 7×7 with stride 1 followed by a maxout layer followed by a convolutional layer with 128 filters of size 5×5 with stride 1 and a maxout layer (All maxout layer performs pooling across 4 feature maps (see Figure 2.8). The next and final hidden layer is a fully-connected layer of 4096 units. The output layer is a fully-connected layer of 768 softmax units.

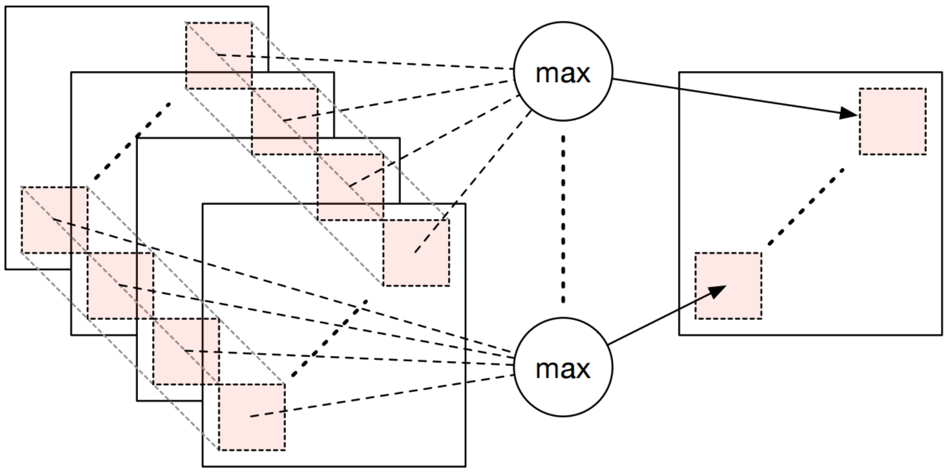


Figure 2.8: Maxout layer pooling across 4 feature maps. The left image shows an example of input aerial image, and the right image depicts the corresponding label image. [61].

and fast-tracks convergence. In general, building a high-performing neural network requires - among other considerations - careful selection of input data format. Typically, the parameters of input image data to neural network include: the number of images, image height, image width, number of channels (RGB), and number of levels per pixel.

Data augmentation is a popular preprocessing technique performed on images. Augmentation involves transforming available images through scaling, rotations and other transformations and including them in the dataset fed into the neural network. This preprocessing technique - among other benefits - exposes the neural network to more training examples, potentially enhancing its generalization ability.
Normalizing image inputs is applied to ensure that input images have similar color range and histogram distributions. This results in faster convergence while training the neural network. Input image normalization is achieved by subtracting the mean of each image from each pixel, and then dividing the result by the standard deviation to obtain data with distribution resembling a Gaussian curve centered at zero.
Image scaling involves resizing images in neural network application. This technique is commonly employed to experiment with different scales of the input image data. For example, scaling an image of size 64×64 to 32×32 involves scaling the width and height by a factor of $0.5(32/64)$.
Uniform aspect ratio involves cropping the input images to square size following neural network models assumption that the input image data have the same size and aspect ratio (square shape input images).

2.2.5 Convolutional Neural Network Architecture

Below we describe individual modules that make up the basic architecture of existing state-of-the-art methods for labelling aerial images.

CNNs are artificial neural networks specially designed to process data that come in the form of multiple arrays. Common examples include a multi-spectral image which can be thought of as a 2-D array of pixels in multiple channels. CNNs differ from neural networks due to the mathematical operation called **convolution** used in place of the weight matrix multiplication in one or more of the network's layers. A rigorous definition of the convolution operation and a principled motivation for its application in CNN is presented in Ian Goodfellow book on deep learning [21].

Convolution Operation: Given an input aerial image;

$$\mathbf{S} \in \mathbb{R}^{n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}} \tag{2.1}$$

and a linear image kernel

$$\mathbf{K} \in \mathbb{R}^{f_h^{[l]} \times f_w^{[l]} \times n_c^{[l-1]}} \tag{2.2}$$

the convolution operation involves convolving kernel $\mathbf{K}$ with $\mathbf{S}$ to produce a new image $\mathbf{S}' \in \mathbb{R}^{n_h^{[l]} \times n_w^{[l]}}$ as depicted by Figure 2.9. $f_h^{[l]}$ and $f_w^{[l]}$ represent respectively the kernel's height and width in layer l, $n_c^{[l-1]}$ the number of channels in the input image, $n_h^{[l-1]}$ and $n_w^{[l-1]}$ are the input image's height and width. The feature map[1]),$\mathbf{S}'(x,y)$, representing the intensity at position (x,y), is calculated by point-wise multiplication of one kernel element with one element of input image $\mathbf{S}$:

$$\mathbf{S}'(x,y) = (\mathbf{S} * \mathbf{K})(x,y) = \sum_i \sum_j \sum_c \mathbf{S}(x+i, y+j, c) \cdot \mathbf{K}(i,j,c) \tag{2.3}$$

The spatial size of $\mathbf{S}'$ can be obtained mathematically as follows:

$$n_h^{[l]} \times n_w^{[l]} = \left\lfloor \frac{n_h^{[l-1]} + 2p^{[l]} - f^{[l]}}{\sigma^{[l]}} \right\rfloor + 1 \times \left\lfloor \frac{n_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{\sigma^{[l]}} \right\rfloor + 1 \tag{2.4}$$

where σ known as **stride**, controls how the kernel slides across the input image $\mathbf{S}$ and p known as **padding**, prevents the feature maps from shrinking too fast after each convolution operation (see Figure 2.9. Another motivation for the use of padding is that it helps preserve important signal that may be present at the boundaries of input image.

[1]also known as activation map

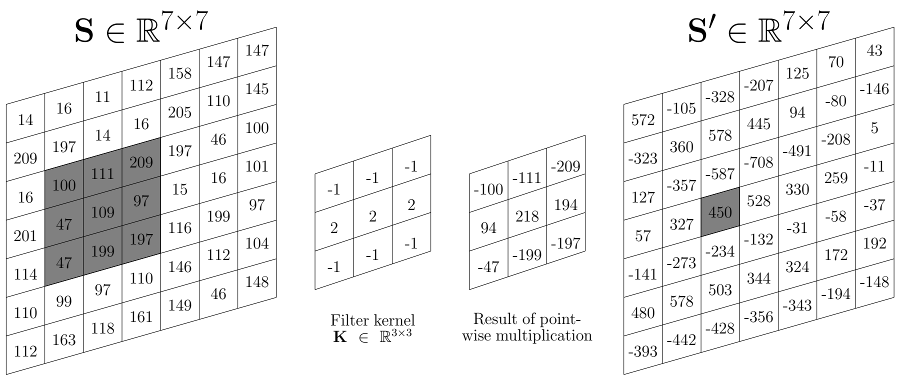

$\mathbf{S} \in \mathbb{R}^{7\times7}$
$\mathbf{S}' \in \mathbb{R}^{7\times7}$
Filter kernel
$\mathbf{K} \in \mathbb{R}^{3\times3}$
Result of point-wise multiplication

Figure 2.9: application of a linear $3 \times 3 \times 1$ image kernel to an input image of size $7 \times 7 \times 1$. Setting $\sigma^{[l]} = 1$, $p^{[l]} = 1$, $n_h^{[l-1]} = n_w^{[l-1]} = 7$, and $f^{[l]} = 3$ in Equation 2.4 yields an output of size 7×7

Convolution Layer: The convolution layer accepts an input image **S** and four hyper-parameters that affects the conv layer's outputs: n, the number of kernels, f_h and f_w, the size of each kernel, σ the stride and the non-linear activation to produce feature maps $\mathbf{S}'$ of size $n_h^{[l]} \times n_w^{[l]} \times n^{[l]}$. One of the properties that make CNNs powerful is **parameter sharing** - a procedure applied in convolution layer that helps to reduce the number of parameters. A reasonable justification for parameter sharing is that if a feature is useful to compute at some spatial position (x, y), then it should also be useful to compute at a different position (x_2, y_2). The parameter sharing assumption results in a dramatic reduction in the number of parameters in the network. This in turn result in less demand for training data to prevent the network from overfitting during training. Typical settings are $n \in \{32, 64, 128\}$, $k_w = k_h = k \in \{1, 3, 5, 7, 9, 11\}$, $\sigma = \{1, 2\}$ and **ReLU** activation.

Pooling Layer: The pooling layer is stacked next to the convolutional layer activations to further reduce the amount of parameters and computation in the network, and hence controls overfitting. Specifically, pooling layers are applied mainly to reduce the feature map dimensionality. Pooling also helps to make the network invariant to small changes in the feature map. It accepts as input, the activations of the previous convolution layer of size $n_h^{[l]} \times n_w^{[l]} \times n$ and two hyper-parameters: the spatial extent $\mathbf{K}_e$ and the stride σ_e to produce an output of size $n_h'^{[l]} \times n_w'^{[l]} \times n'$ where: $n_h'^{[l]} = (n_h^{[l]} - K_e)/\sigma_e + 1$, $n_w'^{[l]} = (n_w^{[l]} - K_e)/\sigma_e + 1$, $n = n'$. While there are other functions used for pooling such as: average pooling and $l2$-pooling, we make use of **max pooling** throughout this work.

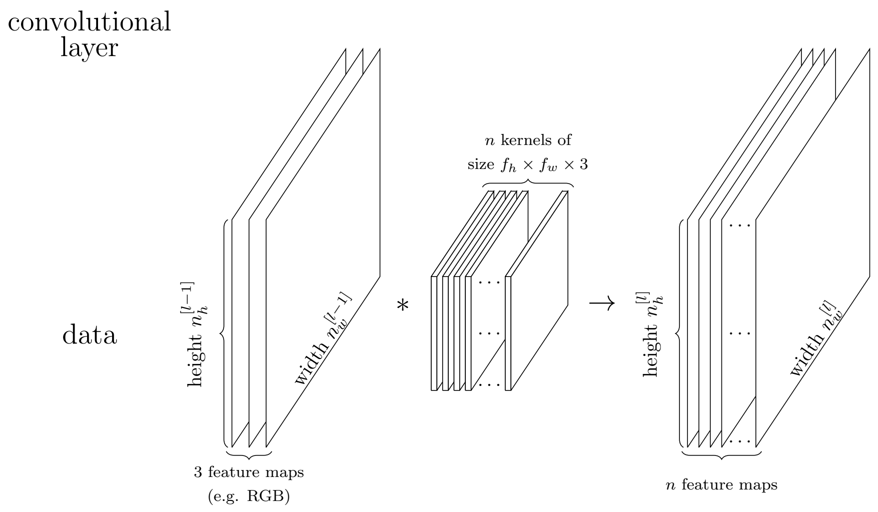


Figure 2.10: A single convolutional layer with n filters of size $f_h \times f_w \times 3$ with stride $\sigma = 1$ applied to input data of size $n_h^{[l-1]} \times n_w^{[l-1]} \times 3$, producing feature maps of size $n_h^{[l]} \times n_w^{[l]} \times n$.

Maxout Layer: A maxout layer [22] is a hidden layer of a feed-forward architecture such as deep CNNs where the activation is the maximum of the inputs to the layer across k feature maps. Given an input $x \in \mathbb{R}^d$, a maxout layer implements the function:

$$h_i(x) = \max_{j \in [1,k]} z_{ij} \tag{2.5}$$

where $z_{ij} = x^T W_{\dots ij} + b_{ij}$, and $W \in \mathbb{R}^{d \times m \times k}$ and $b \in \mathbb{R}^{m \times k}$ are learned parameters.
A unit in a maxout layer can serve as a substitute for ReLU and othe hidden layer activation functions.

Activation Functions

Activation functions are linear or non-linear functions that decide whether a neuron should be activated or not. Non-linear activation function, commonly used in CNN, makes a transformation of the input signal allowing the neural network to solve complex problems. Back-propagation is possible in neural networks because of the use of non-linear activation functions as they provide the neural network with a differentiable nonlinear function that makes it possible for gradients of the cost function to be computed. These gradients are used to update the weights and biases.

ReLU [53]: It is the Rectified Linear Unit used as non-linear activation function in the hidden layers of CNNs

$$f(x) = max(0, x)$$

The main advantage of using the ReLU function over other functions is that it does not activate all the neurons at the same time. If the input is negative, it will convert it to zero and the neuron does not get activated. This means that at a time, only a few neurons are activated making the network sparse making it efficient and easy for computation.
Caveats: It is susceptible to vanishing gradients. The gradients are zero in the negative side of the function which means the weights are not updated during back-propagation. This can create dead neurons which never get activated.

LeakyReLU [43]: This is an improved version of the ReLU function. For ReLU function, the gradient is 0 if $x < 0$ which made the neurons die for activation in that region. To address the ′dying′ problem, LeakyReLU, redefines the ReLU function as a small linear component of x.
Mathematically,

$$f(x) = ax, \quad x < 0, \quad a = 0.3$$
$$f(x) = x, \quad x \geq 0$$

Sigmoid Function: A sigmoid function is a bounded differentiable real function that is defined for all real input values [24] and has a return value from 0 to 1. Mathematically, it is represented as follows:

$$S(x_i) = \frac{1}{1 + \exp(-x_i)}$$

Tanh: The tanh function is the scaled version of the sigmoid function.

$$tanh(x) = 2 * sigmoid(2x) - 1$$

It can be directly written as

$$tanh(x) = \frac{2}{(1 + \exp(-2x))} - 1$$

Tanh works similar to the sigmoid function but its output is symmetric over the origin because it ranges between -1 to 1. Hence, optimization is easier. It basically prevents all the output values to be of the same sign.

2.3 Evaluation Measures

In this section, a review of some commonly used evaluation metrics for semantic segmentation (global precision, class-wise precision, confusion matrix, F-measure or the Jaccard index (also called intersection over union)) are presented.

2.3.1 Region-based accuracies

Most semantic segmentation measures evaluate a pixel-level classification accuracy.The use of pixel-level confusion matrix as a measure for the performance of semantic segmentation systems was perhaps most widely used in the early days of development of this framework. The research community has in recent times gravitated towards the use of more sophisticated accuracy measures. The following paragraphs will give a brief summary of some of the measures developed and applied in the semantic segmentation literature.

Precision measures the labels belonging to the object of interest that are correctly predicted from the total number of predicted object of interest.
The Overall Pixel (OP) accuracy measures the ratio of correctly labelled pixels to the total number of pixels. This score was used in [64] to evaluate the performance of the learned model on automatic visual recognition and semantic segmentation of photographs. One notable drawback to the use of this measure is its bias in dealing with dataset containing imbalanced classes.
The Mean Per-Class (PC) accuracy measures the ratio of correctly labelled pixels to the total number of pixels for each class and then averages over the classes. It is less affected by imbalanced class frequencies than the OP. Instead of measuring the ratio of correctly labelled pixel to the total number of pixels, [35] proposes to measure the ratio of correctly predicted pixels to wrongly predicted pixels in a region surrounding the class boundaries.
Receiver Operating Characteristics (ROC) Curve measures the discrepancy between the ground truth pixels and the predicted pixels of the segmentation method based on confusion matrix. Confusion matrix makes use of *true positives (TP)* - the number of pixels predicted as object of interest pixels when they indeed are, *false positives (FP)* - the number of pixels pre-

dicted as belonging to the object of interest when they are not, *true negatives (TN)* - the number of pixels predicted as not belonging to the object of interest when they indeed are not, and *false negatives (FN)* - the number of pixels predicted as not belonging to the object of interest when they indeed are. The ROC curve is plotted using the sensitivity (also known as the recall or true positive rate (TPR)) and the 1-specificity (also known as the false positives rate (FPR)).
F1-score combines the values of the precision and recall to evaluate the performance of our segmentation model:

Metric	Definition
Overall Pixel (OP)	$\frac{TP+TN}{TP+FP+TN+FN}$
Jaccard Index (JI)	$\frac{TP}{TP+FN+FP}$
Precision (P)	$\frac{TP}{TP+FP}$
Recall or TPR	$\frac{TP}{TP+FN}$
FPR	$\frac{FP}{FP+TN}$
F-1 Score	$2 \times \frac{P \times TPR}{P+TPR}$

Table 2.3: Threshold metrics for segmentation evaluations

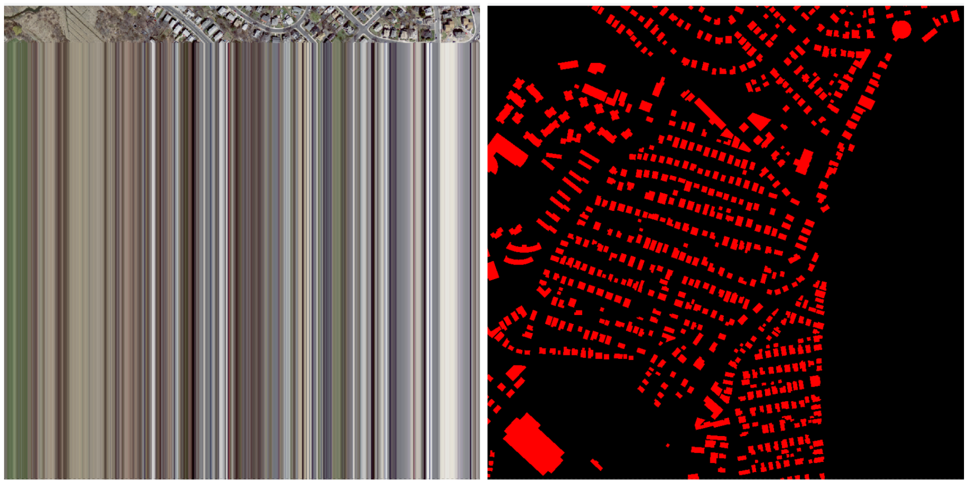
Chapter 3

Methodology

3.1 Preprocessing

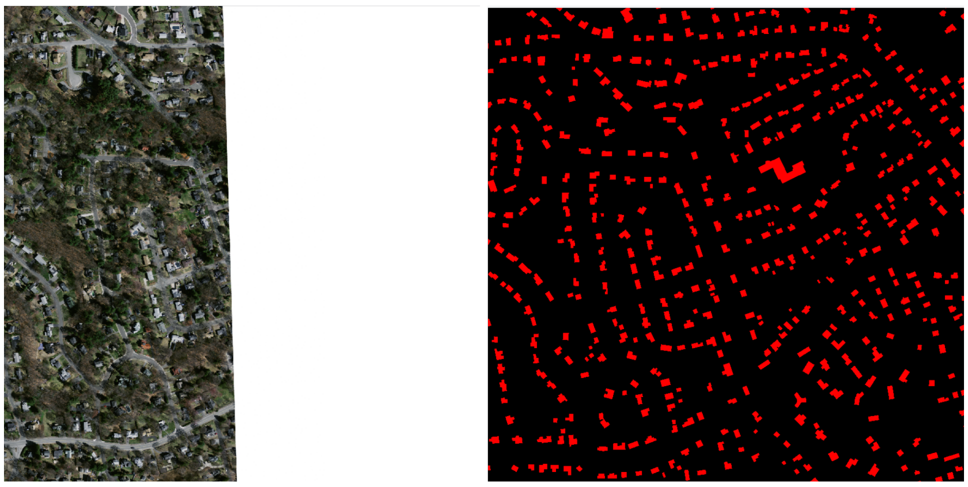
In this work, all preprocessing operations are performed on patches generated from the original aerial image datasets. Though the original image can also be referred to as a patch, in this thesis, we use patch to refer to any square image of size far less than the size of the original image. The MBD dataset has 131 images each with 1500×1500 pixels. We made patches from images that are not corrupted. Examples of corrupted and uncorrupted images are shown in Figures 3.1a, 3.2a & 3.3a. The corrupted images present two notable challenges: First, the vector values of some of them are not extractable. Second, corrupted images with accurate ground truths introduce confusion into the network as a result of wrong supervision during training. The main idea of the patch based preprocessing is to extract all $w \times w$ patches from the original $W \times W$ image size ($w << W$), usually with overlaps from the given image. In this approach, it is expected that each patch taken from the original image has similarity with neighbouring patches. The intuition behind this approach is that a combination of these patches during training may result in learning of a better representation of the input image and potentially translates to a better prediction of image class. Below, we provide a mathematical formulation of creating square patches from an original square image of a given size. Note that it is also possible to divide a non-square image into patches. Here, we focus only on square patches and square image.

Let $w \times w$ represent the width and height of a single patch $\mathcal{S}_{ij}$ from a collec-



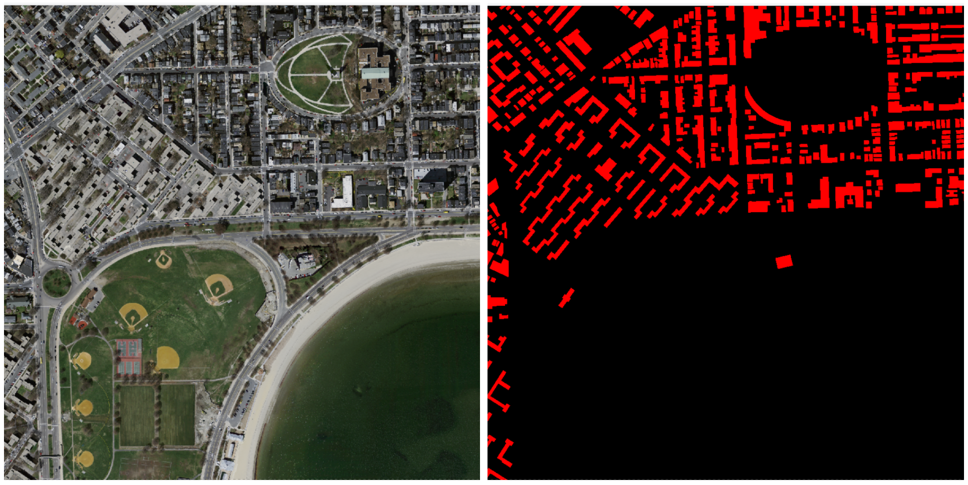
(a) A corrupted aerial image (b) Image groundtruth

Figure 3.1: An example of a corrupted image but with a valid groundtruth mask.



(a) A partially corrupted aerial image (b) Image groundtruth

Figure 3.2: An example of partially corrupted image but with a valid groundtruth mask. This image can be read without error, but the presence of labels on blank regions of the image would introduce confusion during training.



(a) An uncorrupted aerial image (b) Image groundtruth

Figure 3.3: (a) An example of an uncorrupted aerial image (b) The corresponding groundtruth for the uncorrupted aerial image. It can be seen that the groundtruth is similar to the uncorrupted aerial image

tion of patches $\tilde{\mathcal{S}}_i$ to be cut out from the original image $\mathcal{S}_i$ of size $W \times W$. In this formulation, it is expected that $w << W$. Generally, the total number of patches $n(\tilde{\mathcal{S}}_i)$ that can be produced from $\mathcal{S}_i$ is given by:

$$n(\tilde{\mathcal{S}}_i) = \left(\frac{w}{\eta} \times \frac{W_i}{w}\right)^2 \tag{3.1}$$

Where $\eta \geq 1$ is the number of $\mathcal{S}_{ij}$ pixels excluded in its nearest neighboring patches.

An immediately obvious benefit of working with image patches instead of working with the entire image is the reduction of computational demand as a result of less number of convolutional operations performed on patches in a CNN framework.

Setting η in Equation 3.1 to w is the case of producing non-overlapping patches from $\mathcal{S}_i$. A downside of this is the appearance of artifacts in the resulting patches - a single object may fall in more than one patch. This can be reduced by using overlapping patches obtained by setting $1 \leq \eta < w$. Overlapping gives objects a better chance of being fully visible in at least one patch. While overlapping can result in gain in accuracy; non-overlapping produces fewer patches, hence reduced preprocessing and training time.

3.2 CNN Architecture

Adopted from [51], the CNN architecture used in this work (Figure 3.4) accepts as input, aerial image patches each of size 64×64. The first hidden layer is a convolution layer with 64 filters of size 7×7 with stride 3. It is followed by 2×2 max pooling with stride 1. The second hidden layer is also convolutional with 112 filters of size 3×3 with stride 1, followed by a third convolutional layer with 80 filters of size 2×2 with stride 1. All four hidden layers consist of LeakyReLU activation function. The output layer is a fully-connected layer of 4096 logistic units.

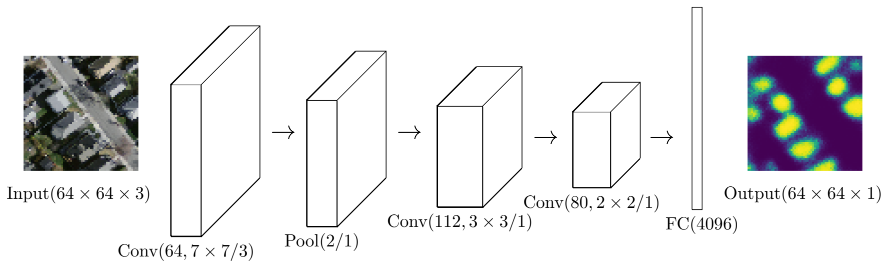

Input$(64 \times 64 \times 3)$
Conv$(64, 7 \times 7/3)$
Pool$(2/1)$
Conv$(112, 3 \times 3/1)$
Conv$(80, 2 \times 2/1)$
FC(4096)
Output$(64 \times 64 \times 1)$

Figure 3.4: CNN Architecture used in this work.

Let K' be the number of classes of interest (1 in our case representing the building class); label image $\tilde{M}$ has $K' + 1$ classes. Because it is difficult to consider all objects that can appear in aerial imagery as classes of interest, we consider the background class to represent pixels that do not belong to any of the K' classes of interest.

Let K denote the number of all classes including the background, $K = K' + 1$; 2 in our case. The goal is to train a CNN to predict a $n_h^{[l-1]} \times n_w^{[l-1]}$ sized label patch from a $n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$ sized aerial image $\mathbf{S}$ where l is the l-th layer of our architecture. The architecture used in this thesis generally takes an order 3 tensor as its inputs, specifically, an aerial image (prepared as described in Section 3.1) of width $n_w^{[l-1]}$, height $n_h^{[l-1]}$ and channel $n_c^{[l-1]}$ ($n_c^{[l-1]} = 3$ representing R,G,B colour channels). The input passes through a series of processing units in a sequential fashion. A processing unit in this single forward-pass collectively represents a **layer** in the CNN architecture. The layer could be a convolutional layer, a pooling layer, a fully connected layer or a normalization layer with non-linear activation. In the architecture investigated in this thesis, the first layer is a hidden convolutional layer. Our task is image segmentation, i.e., classify each pixel on the image and

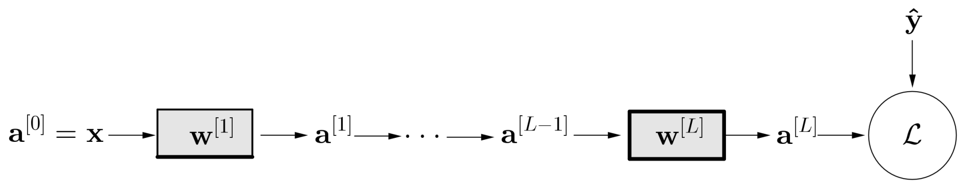

$$\begin{tikzcd}
 & & & & & & \hat{\mathbf{y}} \arrow[d] \\
\mathbf{a}^{[0]} = \mathbf{x} \arrow[r] & \boxed{\mathbf{w}^{[1]}} \arrow[r] & \mathbf{a}^{[1]} \arrow[r] & \cdots \arrow[r] & \mathbf{a}^{[L-1]} \arrow[r] & \boxed{\mathbf{w}^{[L]}} \arrow[r] & \mathbf{a}^{[L]} \arrow[r] & \mathcal{L}
\end{tikzcd}$$

Figure 3.5: The input $a^{[0]}$ is fed into the first hidden convolutional layer where $w^{[1]}$ denote the parameters associated with the layer. The resulting activation $a^{[1]}$ is fed into the next hidden convolutional layer. The process is repeated until the last layer of the network to obtain the output predictions representing the class each pixel belongs to. $\hat{y}$ represent the actual class of each pixel found in the input image and $\mathcal{L}$ denotes the loss function applied to compute the difference between the actual labels and the predicted labels.

create a map of all detected building areas on the aerial image. Basically, we want our final output to be a binary image where each pixel in the image is either a **building** pixel or **non-building** pixel. Figure (3.6) shows a typical output from a segmentation system.
Figure (3.5) illustrates the processing that takes place from one layer to the other in a CNN. The input $\mathbf{a}^{[0]}$ is fed into the first hidden convolutional layer where $w^{[1]}$ denotes the parameters associated with this layer. LeakyReLU activation is applied to the resulting output $\mathbf{a}^{[\mathbf{1}]}$. The output of this layer is max-pooled and fed into the next hidden convolutional layer. The process continues until all layers in the CNN have been visited. The final output is computed by applying a sigmoid function at every unit/neuron/node in the final layer.

The CNN computes the probability of a pixel p belonging to the building class, using as input the intensity values of an input aerial image of size $n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$. The CNN is first trained using the available training images. After training, the model is used to predict the probability of each pixel in the test image being a building pixel thus generating a map of building probabilities. By thresholding, a new binary image of similar size with the input image is obtained. To achieve better output quality, a post-

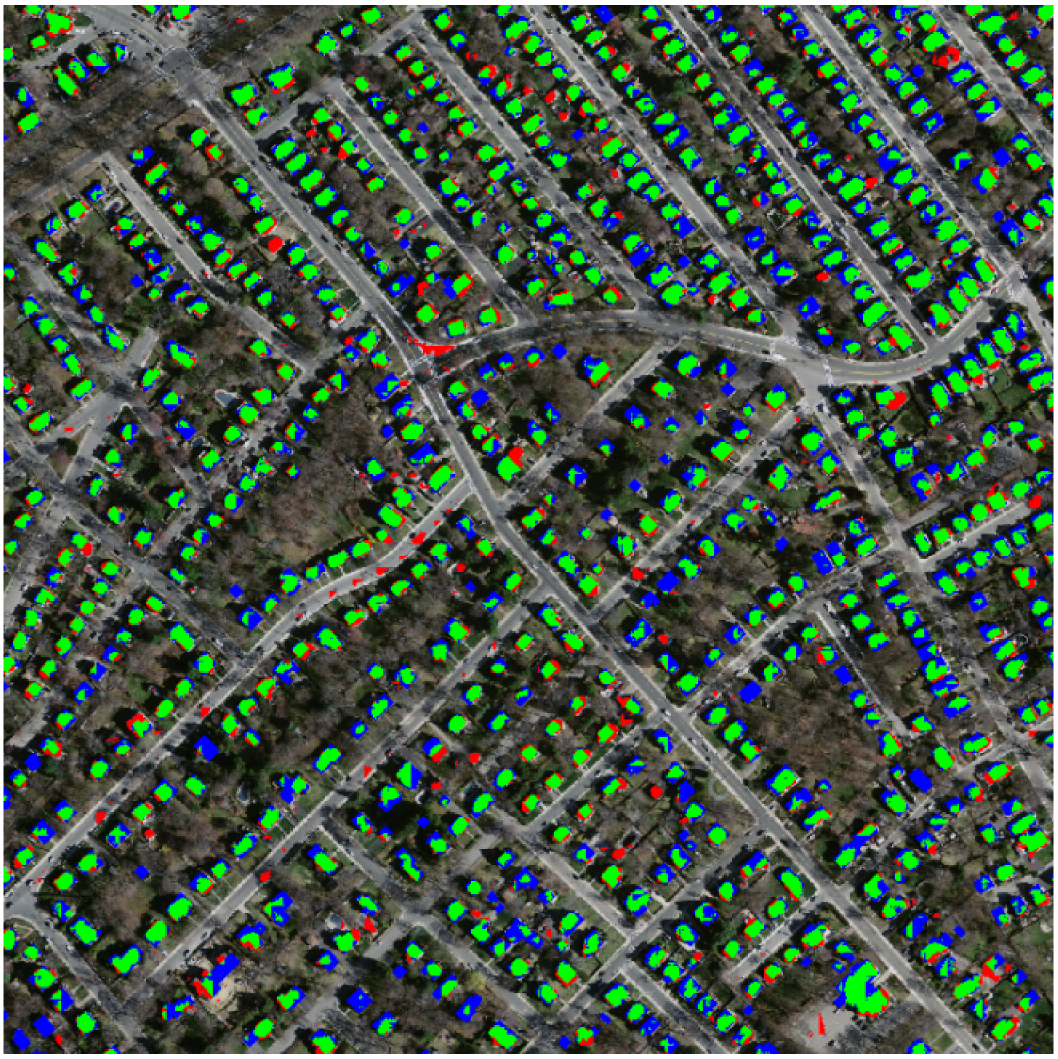


Figure 3.6: the image is the map predicted by our base model (without dropout, and post-processing) overlayed on the actual image. The green parts are true positives, the red parts are false positives, the blue parts are false negatives and the rest are true negatives.

processing technique is usually applied to the building probabilities before or after thresholding but we did not consider this in this thesis.

3.2.1 Fully Connected Deep CNN Architecture

In this thesis, deep CNN is used to extract features from one hidden layer to the other in a hierarchical fashion. The output (or last) layer of our base deep CNN architecture is a fully connected layer that infers the class of each pixel in the input aerial image.

The output layer, in our architecture, is a fully connected layer with one neuron per pixel. Using a sigmoid activation function for the last layer guarantees that each neuron's output activation can be interpreted as the probability of a particular pixel belonging to the building class. $a^{[L-1]}$ is transformed by sigmoid activation function to obtain its probability of being a building pixel $a^{[L]}$. The last layer $\mathcal{L}$ as represented in Figure 3.5 computes the distance between $a^{[L]}$ and the corresponding ground-truth value $\hat{y}$. In our case, the loss function used to compute the distance is a binary cross entropy defined as:

$$\mathcal{L}_{i,j,c} = -(\hat{y}_{i,j,c} \ln a^{[L]}_{i,j,c} + (1 - \hat{y}_{i,j,c})(1 - \ln a^{[L]}_{i,j,c})) \tag{3.2}$$

The total loss function expressed as:

$$\mathcal{C} = \sum_i \sum_j \sum_c \mathcal{L}_{i,j,c} \tag{3.3}$$

is then minimized using stochastic gradient descent.
The convolution operation applied in this work can be expressed as

$$\mathbf{z}^l_{i.j,c} = \sum_{i=0}^{H} \sum_{j=0}^{W} \sum_{c^l=0}^{C^l} \mathbf{w}_{i,j,c^l,c} \times \mathbf{a}^{[l-1]}_{i,j,c^l} \tag{3.4}$$

where $\mathbf{w}$ are filter values and $\mathbf{z}$ is the feature map.
Throughout this thesis, we omit the **bias** term usually added to $\mathbf{z}_{i,j,c}$. For a more detailed description of the convolution operation, see Chapter 2.

Activation Function

A detailed review of all activation functions used in this thesis is provided in Chapter 2 and a comparison of the use of **tanh**, **LeakyReLU** and **ReLU** non-linear activations in the hidden layers is provided in Chapter 4.

For our task, the main hidden layer activation function used is LeakyReLU. Individual element in the output obtained from convolving $w^{[l]}$ with the input $\mathbf{a}^{[l-1]}$ ($l = \{1, 2, \cdots, L-1\}$) is fed as input to the LeakyReLU layer to obtain the corresponding activation maps $\mathbf{a}^{[i]}$ as represented in Figure 3.5.

We experimented with other non-linear activation functions namely Tanh and ReLU for the purpose of comparison.

The Forward Pass

After learning the weights of the CNN model $w^{[1]}, \cdots w^{[L-1]}$, the probability of each pixel in the input image belonging to the object of interest (building in our case) is predicted using the model. This completes the first forward pass.

Later, we discuss the role of the $\mathcal{L}$ layer in our base CNN architecture.

Training

The $\mathcal{L}$ layer is useful in the training stage to learn the CNN model parameters using the given training examples. CNN model parameters are adjusted to minimize the distance between the CNN model prediction and the ground-truth.

In particular, for a single training example $a^{[0]}$, adjusting such parameters involves running the CNN network in the forward direction to compute $a^{[L]}$ which is then compared with $\hat{y}$ - the label corresponding to $a^{[0]}$. The gradient of the loss function as defined in Equation 3.2 is computed to obtain information about how to update the parameters of the network. In our task, the optimizer used is stochastic gradient descent represented in the form:

$$w^{[l]} \leftarrow w^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial w^{[l]}} \tag{3.5}$$

where $\frac{\partial \mathcal{L}}{\partial w^{[l]}}$ computes the rate of change of $\mathcal{L}$ with respect to changes in $w^{[i]}$. η, the learning rate guides how the parameters are updated. Popular choices for η are 0.001, 0.005, 0.05. In our case, we used 0.05 as default learning rate as it appears to give the best performance. The parameter update is performed at the end of each batch to minimize the overall loss. A single epoch is completed once all training examples have been seen and used to update the parameters. The number of steps or iteration in a single epoch is typically equal to the number of dataset samples divided by the batch size but in our case, we set batch size to 32 and with 64000 training examples we iterate 2000 times per epoch.

Gradient Descent (GD)

GD is an approach for training Deep Neural Networks (DNN). To further establish GD as the state-of-the-art for training DNN, different variants of it have been introduced over the years [67, 29, 17, 34]. The primary objective of GD is to optimize the parameters θ of DNN by minimizing the objective function which (in essence) is the average-sum of discrepancies between the DNN predictions and the actual input labels.

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\mathbf{a}^{[L]}, \hat{y}) \tag{3.6}$$

Variants of GD include:
online training or Stochastic Gradient Descent (SGD) - designed to see exactly one training example at a time before parameters update are performed. While this strategy potentially reduces premature convergence due to noisy updates, one downside is that it is computational expensive to apply because it requires parameters update to be done after each training example is seen by the network.
Batch Gradient Descent (BGD) - designed to perform parameters updates only after all training examples have been treated. BGD tackles the downside of SGD but the DNN stands the risk of converging to a sub-optimal set of parameters due to less noisy error gradients. Another drawback of BGD is that it requires all training examples to be loaded all at once to memory making it less memory efficient.
Mini-batch Gradient Descent (MGD) attempts to combine the model robustness of SGD with the computational efficiency of BGD by exposing the algorithm to randomly chosen subsets (mini-batches) of the entire dataset. Parameters update are performed after each mini-batch is evaluated. A downside is that MBG introduces an additional hyper-parameter (mini-batch size) to the algorithm.

Optimizing Gradient Descent

MGD is a widely used DNN optimization technique but its vanilla form does not guarantee good convergence due to the difficulties in choosing proper learning rate as well as adjusting the learning rate during training and the issue of getting stuck in their numerous local sub-optimal minima as a result of the use of non-convex objective function popularly used in DNN. Here, we briefly describe some of the optimization algorithms that are considered in this thesis to improve the performance of the vanilla MGD.

Momentum [67]: In essence, momentum helps to accelerate GD in relevant direction towards global minimum by avoiding getting trapped in local sub-optimal minima. In our experiment we use a momentum value of 0.9 - [51] demonstrated that using other values does yield an improvement in the performance of the network.

$$v_{t+1} = \mu v_t - \eta J(\theta_t) \tag{3.7}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3.8}$$

where $\eta > 0$ is the learning rate, $\mu \in [0, 1]$ is the momentum coefficient, v is the update vector, and $\nabla J\left(\theta_t\right)$ is the gradient of the objective function J at θ_t.

Batch Normalization [29]: As the parameters of all preceding layers of DNN adjust during training, they in-turn affect the distribution of input u fed into successive layers. This effect is popularly referred to as the internal covariate shift [29]. Batch normalization helps to ensure that the parameters to be optimized during training do not have to re-adjust to account for the change in the distribution of u. This is achieved by linearly transforming each scalar feature in layer inputs to have zero mean and unit variance, and decorrelated [29]. Formally, for a layer with n-dimensional input $x = \left(x^1, \cdots, x^n\right)$, the normalized ith feature is given by:

$$\hat{x}^i = \frac{x^i - E\left[x^i\right]}{\sqrt{Var\left[x^i\right]}} \tag{3.9}$$

where the expectation $E\left[x^i\right]$, and variance $Var\left[x^i\right]$ are computed over the training data set.

To avoid reducing the representional power of each layer as a result of the normalization, a pair of learnable parameters γ^i and β^i are applied to scale and shift $\hat{x}$ as follows:

$$y^i = \gamma^i \hat{x}^i + \beta^i \tag{3.10}$$

Backward Pass

Stochastic gradient descent involves adjusting the parameters of the network using gradients obtained from each batch. This approach is known as minibatch SGD when a minibatch training examples is used. Popular choices for mini batch training examples are 32 and 64.

The backward pass starts by computing how much the prediction $\mathbf{a}^{[L]}$ differs from the actual label $\hat{\mathbf{y}}$ using the loss function $\mathcal{L}$ as given in Equation 3.2.

For each layer of the CNN architecture, the partial derivatives of the loss function with respect to $z^{[l]}$ (result of the convolution operation performed on layer $l-1$) $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l]}}$ and that layer's parameters $\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{[l]}}$ are computed using chain rule. $\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{[l]}}$ guides the update of the current i-th layer parameters and $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l]}}$ provides information about how $z^{[l]}$ should be changed to reduce the loss function. Equation 3.5 gives the update rule for the parameters of each layer.

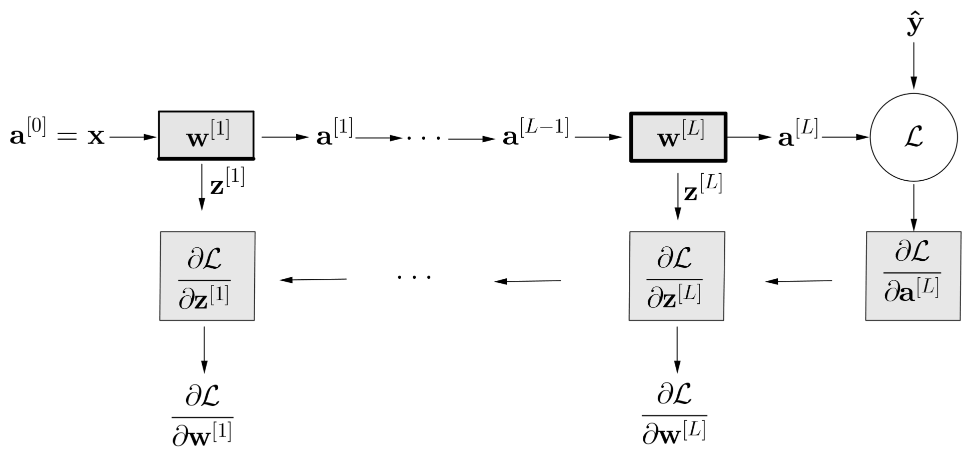
Formally, $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{[L]}}$ are computed as follows:
From Equation 3.2, the gradient of the loss function with respect to the i-th activation of L-th layer is given by:

$$\frac{\partial \mathcal{L}_{i,j,c}}{\partial \mathbf{a}^{[L]}_{i,j,c}} = -\frac{\hat{\mathbf{y}}_i}{\mathbf{a}^{[L]}_{i,j,c}} + \frac{1-\hat{\mathbf{y}}_{i,j,c}}{1-\mathbf{a}^{[L]}_{i,j,c}} \tag{3.11}$$

Using chain rule and Equation 3.11

$$\frac{\partial \mathcal{L}_{i,j,c}}{\partial \mathbf{z}^{[L]}_{i}} = \mathbf{a}^{[L]}_{i,j,c} - \hat{\mathbf{y}}_{i,j,c} \tag{3.12}$$

$$\frac{\partial \mathcal{L}_{i,j,c}}{\partial \mathbf{w}^{[L]}_{i,j,c}} = (\mathbf{a}^{[L]}_{i,j,c} - \hat{\mathbf{y}}_{i,j,c}) \cdot \mathbf{z}^{[L]}_{i,j,c} \tag{3.13}$$



$\hat{\mathbf{y}}$
$\mathbf{a}^{[0]} = \mathbf{x}$
$\mathbf{w}^{[1]}$
$\mathbf{a}^{[1]}$
$\mathbf{a}^{[L-1]}$
$\mathbf{w}^{[L]}$
$\mathbf{a}^{[L]}$
$\mathcal{L}$
$\mathbf{z}^{[1]}$
$\mathbf{z}^{[L]}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{[1]}}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{w}^{[L]}}$

Figure 3.7: Forward and backward pass of CNN architecture

3.3 Experimental Setup

Keras

Building deep neural networks from scratch, particularly deep CNN, is a daunting task involving complex computational graphs with atleast a few million of tunable parameters. Hence, open source frameworks that abstract most of the computations away from the user are generally employed. A number of such frameworks exist but in this work, we make use of *Keras* - a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, and Theano. Keras allows for fast experimentation, supports CNNs, runs seamlessly on CPU and GPU, treats individual components of a neural network architecture (neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes) as standalone modules that can be combined to create new models, and easy addition of customized modules [9]. In this work, we use TensorFlow [1] as backend for Keras.

Hyper-parameter Initializations

Here, we discuss how we set the initial values of the hyper-parameters. For the most part, we follow approaches proposed in the literature.

- **Filters Size**: In this study, we use small filter size (7×7) in the first hidden convolutional layer since they are better suited for learning low level features and computationally more efficient than larger ones (a comparison of different choices of filter size is presented in Chapter 4).

- **Weight Initialization**: We initialized the weights of the CNNs using Glorot uniform [20] to make the variance of the output of a layer to be equal to the variance of its inputs.
- **Number of epochs versus iterations-per-epoch**: we favoured the use of large number of epochs over the use of large number of iterations-per-epoch.
- **Learning Rate**: We use a default learning rate of 0.05.
- **Batch Normalization (BN)**: To investigate the effect of BN on the performance of our model, we performed three experiments - 1). applying batch normalization to the feature maps after non-linear activation (BNA), 2). applying batch normalization to the feature maps before non-linear activation (BNC), and 3). No batch normalization (NBN). The results of these experiments are discussed in Chapter 4.

Learning

We used SGD and momentum and trained 100 epochs with a learning rate of 0.05 and set momentum to 0.9. Each epoch was trained on 2000 batches. Each batch consists 32 image patches. Each batch was generated randomly by cropping 32 64×64 patches from the original images. We learn the parameters of the neural network by minimizing the negative log likelihood of the training data. The negative log likelihood takes the form of a cross entropy between the patch $\tilde{\mathbf{m}}$ derived from the given map and the predicted patch $\hat{\mathbf{m}}$. Experiments with different activation functions (ReLU-family, tanh) show that LeakyReLU leads to a network that performs well in labelling buildings from aerial images. The model is trained on MBD and evaluated on held-out samples of MBD. To understand the generalization capability of the model, we further evaluated its performance on INRIA dataset.

3.4 Evaluation Measures

In this thesis, we evaluated the performance of our models using *precision-recall* curve, and the F-1 score. The *break-even point* corresponding to a single value in the precision-recall curve (where precision is equal to recall) is also used in order to compare with other works in this domain ([50]. We reviewed these metrices in Chapter 2.

Chapter 4

Results and Discussion

In this chapter we present and discuss the main results as well as insights gained from our experiments to determine the effect of hyperparameter choices in our analysis pipeline.

Overlapping patches

Our input pipeline involves making a cutout of smaller patches from larger images. This method has, however, an undesirable effect of splitting objects at the boundary between two separate patches - which may confuse CNN during training. A non-overlapping cutout strategy also reduces the total number of training samples. To mitigate this issue we generated training samples by letting neighbouring patches overlap as described in Chapter 3.

In Figure 4.1, we show the precision-recall curve of a CNN model trained with examples produced using sliding windows 16 (blue curve), 32 (red curve) and 64 (green curve). The amount of the sliding window size controls the degree of overlapping. Given our choice of an input image size 64×64, sliding window size of 64 means no overlapping between neighbouring patches. The performance of our model significantly improves as we increase the amount of overlap between neighbouring patches.

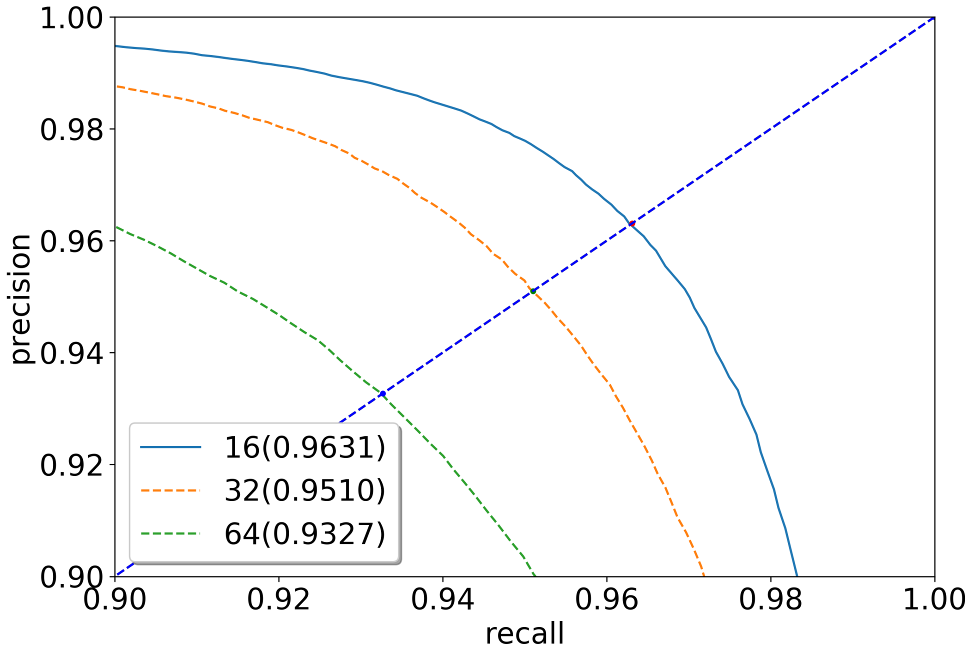


Figure 4.1: **Effect of overlapping patches:** a precision-recall curve of two hidden layer model trained on samples cutout from an aerial image using different window strides - 16, 32 and 64. The numbers inside the parenthesis give the break-even point values, where precision is equal to recall. The model trained on samples generated by using stride = 16 performed best while the model trained on samples with no correlation between neighbouring patches (stride = 64) has the worst performance. Each patch contains at least 10% building pixels.

Number of iterations

It is well known that deep CNNs require a large number of iterations with large number of data to learn robust features. Commonly a single epoch is defined by the number of iteration it takes to train the model by all available data. The number of weight updates per epoch is determined by the ratio of the total number of training sampled and the batch size. In our case we have about 1.2 millions patches and a batch size of 32. As a compromise between computational limitations and a high correlation in our training samples, as a result of large overlapping fraction, we set an epoch to be the number of iteration it takes to process 64000 randomly selected images. A batch size of

32 means, the number of weight updates per epoch is 2000.

Our experiment shows that increasing the number of epochs to about 100 yields a better result - see Figure 4.2. Increasing further, however, did not improve our result. We used 100 epochs to train our main model but due to the computational burden we limited the number of epochs to 20 for all experimental runs used to study the effect of hyperparameters.

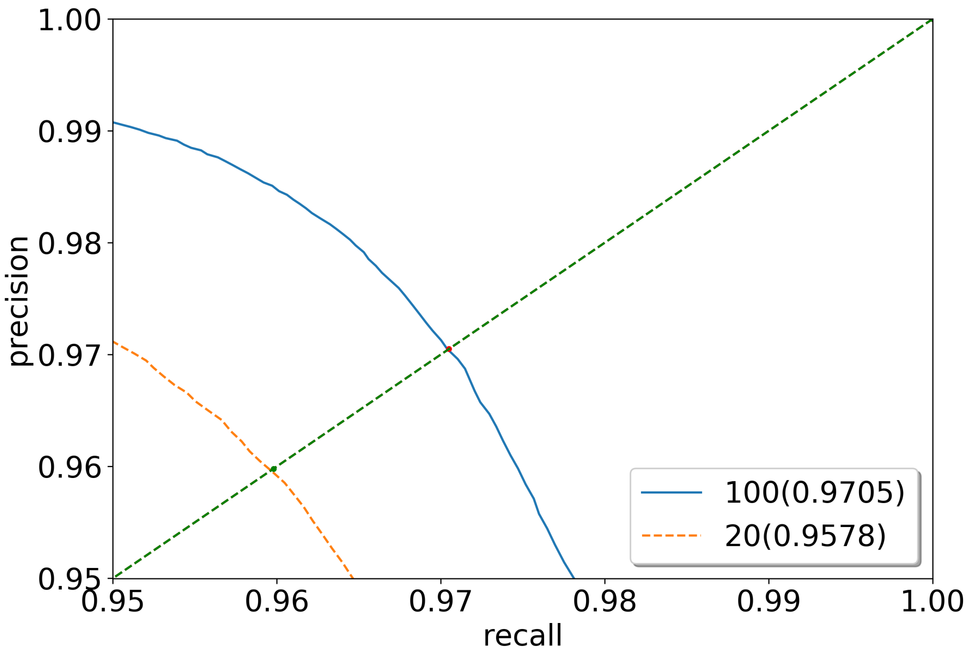


Figure 4.2: **Effect of total number of iterations:** precision-recall curve comparing models trained for 20 and 100 epochs. The red and green dots represent the break-even points for 100 and 20 epoch models respectively, shown in the legend inside the parenthesis.

Effect of Activation Function

Details about different types of activation functions, their pros and cons as well as their use cases are presented in Chapter 2. In this section, we discuss how **ReLU, Tanh, LeakyReLU** activation functions impact the performance of CNN model trained on aerial images.
The difference between using ReLU, tanh and LeakyReLU functions in the hidden layer is apparent in the first hidden convolutional layer (See Fig-

Model type	Accuracy	Break-even point	F1-Score
BNA	0.9019	0.9594	0.9574
BNC	0.8951	0.9578	0.9585
NBN	0.9072	0.9598	0.9505

Table 4.1: **Effect of batch-normalization:** no batch-normalization (NBN) compared with batch normalization after convolution but before activation layer (BNC) and after activation layer (BNA).

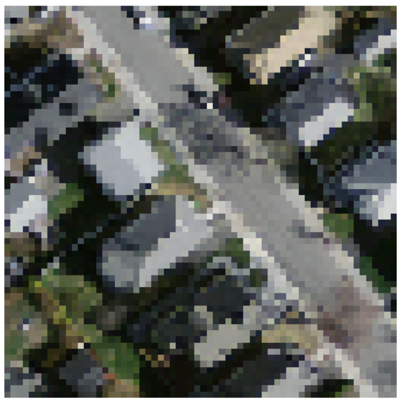
ures 4.3. In ReLU, filters with non-positive values are not activated, making the network sparse. This sparsity results in efficiency and easy computation. One of the caveats of using ReLU, however, is that during back-propagation, the weights associated to non-positive filters are not updated which can result in dead filters which never get activated. The impact of sparsity caused by ReLU on CNN performance in labelling aerial images is compared to less sparse activation functions (Tanh and LeakyReLU) in Figure 4.4. We observe that LeakyReLU activation leads to a slightly better precision-recall performance.

Effect of Batch Normalization

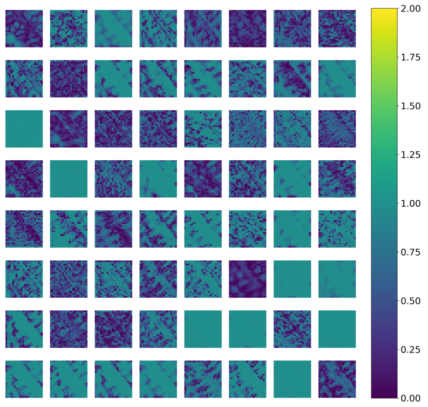
We carried out three experiments to understand the effect of batch normalization: 1) NBN - no batch-normalization layer is added 2) BNC - adding a batch-normalization layer after a convolution but before an activation layer 3) BNA - adding a batch-normalization map after an activation layer. As can be seen from Table 4.3, the best (F1-score) accuracy is achieved by BNA. The same table and Figures 4.5, on the other, suggest that when it comes to visual inspection and the break-even point value, NBN performs better than the others.

Effect of Filter Size

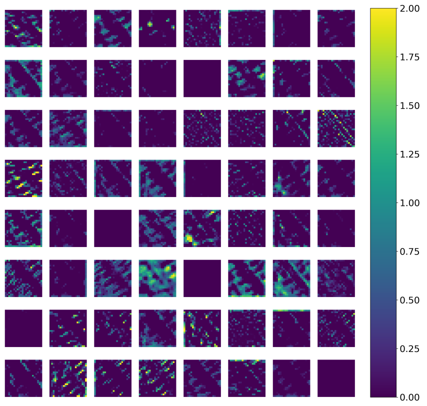
In this section, we present a discussion on the results obtained from exploring the hyper-parameter space (such as filter size, learning rate, and weight



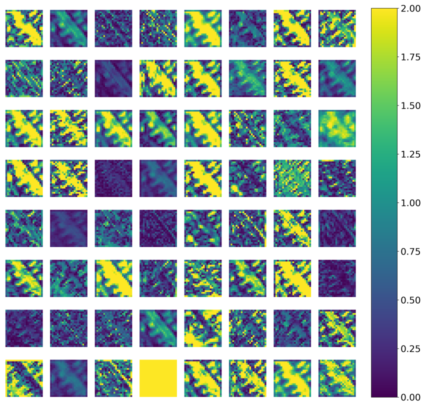
(a) A patch from the test set.



(b) Tanh activation maps



(c) ReLU activation maps



(d) LeakyReLU activation maps

Figure 4.3: **Effect of activation functions:** the image in the top left is used to produce the activation maps at the first hidden convolutional layer shown in the other figures. The model with the ReLU (bottom left) activation function leads to sparse features while Tanh results in dense features. LeakyReLU is in between.

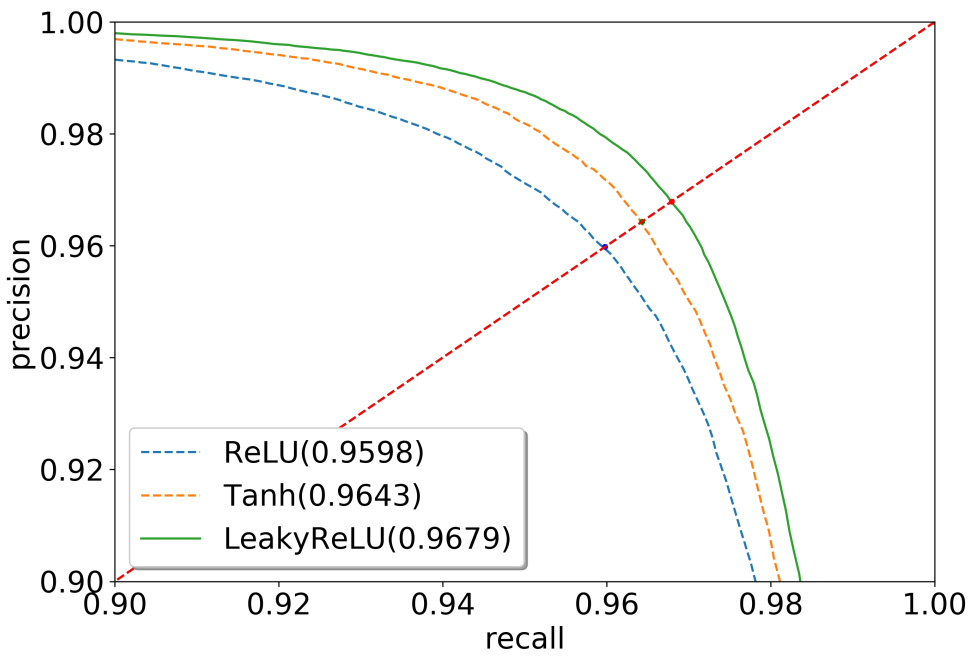


Figure 4.4: **Effect of activation functions:** Comparison of Precision-Recall (PR) curve from ReLU, LeakyReLU and Tanh activation functions. The LeakyReLU has a slightly better PR performance. The red, blue and green dots represent the break-even points,and mentioned in the legend, for LeakyReLU, Tanh and ReLU models respectively.

decay) different from the ones used in [50].

We compare the classification output obtained from using convolutional filter size as presented in [50] with other filter size choices.

Table 4.3 shows the precision-recall break-even point values and the overall accuracy achieved by CNN models with different filter sizes. The second row compares the result obtained using [50] filter sizes. This clearly illustrates the well known fact that large convolutions can be replaced by smaller ones without loosing accuracy. Aside from a model performance gain, small convolution filter sizes are also better suited for smoothing, edge detection and shifting which ultimately contribute to a better model performance.

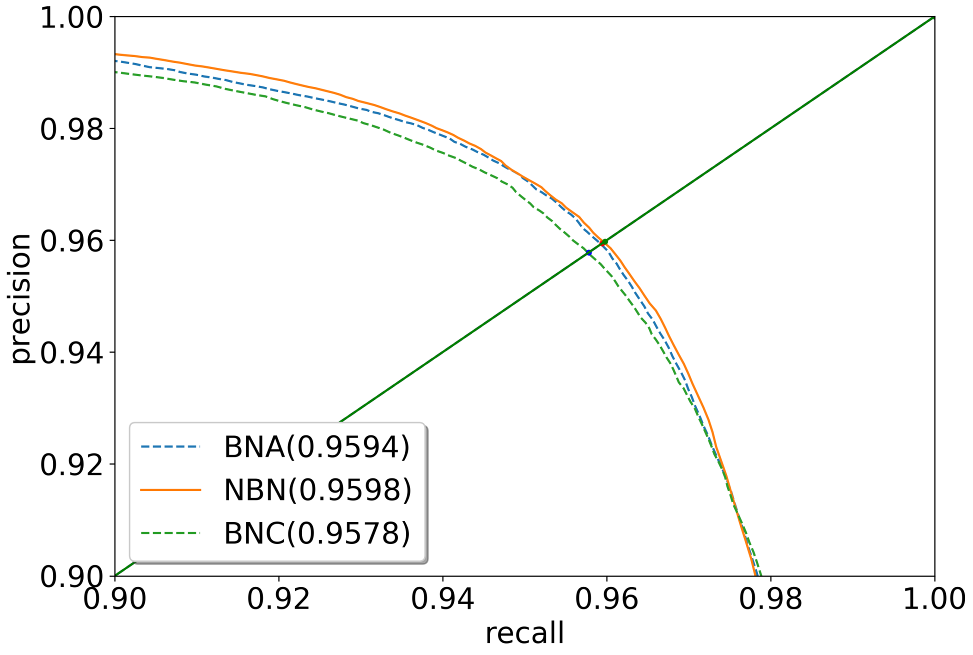


Figure 4.5: **Effect of batch-normalization:** Precision-recall curve for three models as named in the legend. The values in the parenthesis are the break-even point, which is where precision and recall are equal.

Ref.	Year	Break-even-point*
Volodymyr [50]	2013	0.9211
Shunta & Yoshimitsu [61]	2015	0.9230
Marcu [46]	2016	0.9423
This work	2018	0.9568

Table 4.2: Results on MBD from 2013 to 2018.

Conv 1	Conv 2	Conv 3	Break-even point	F1-Score
16	4	3	0.9211	-
16	3	2	0.9686	0.9566
9	3	2	0.9715	0.9611
7	3	2	0.9737	0.9626

Table 4.3: **Effect of filter size:** comparison between different filter sizes used in this work. The break-even point is computed from the precision-recall curve. The second row entry displays the filter sizes used in [50] work.

Unless stated otherwise, we use relaxed precision and recall scores to compare with previous works ([50, 61]). The relaxed precision is defined as the fraction of detected pixels that are within ρ pixels of a true pixel, while the relaxed recall is defined as the fraction of true pixels that are within ρ pixels of a detected pixel [61].

4.1 Generalisability of the model

Training models to find buildings on an aerial image is an expensive task. The number of satellites and airborne devices equipped with different sensors capturing images from different parts of the world under continuously changing lighting conditions introduces more complications for coming up with a unified model. In this section, we investigate how well our model (trained on MBD) generalises to the task of finding buildings on the Inria Aerial Image Labeling Dataset (Inria) - See Chapter 2 for the description of this dataset.

Figure 4.6 shows the precision-recall curve obtained using CNN model trained on MBD dataset applied to a sample of Inria training maps. We applied the MBD trained model to a variety of city maps included in the Inria data. For comparisons in the same figure we show the curve from the MBD test data. From this figure, we see that the MBD trained model performs poorly on all Inria maps - illustrating the difficulty of generalization in CNN networks.

In Figures 4.7 & 4.8, we show the predicted masks for a sample of Inria maps

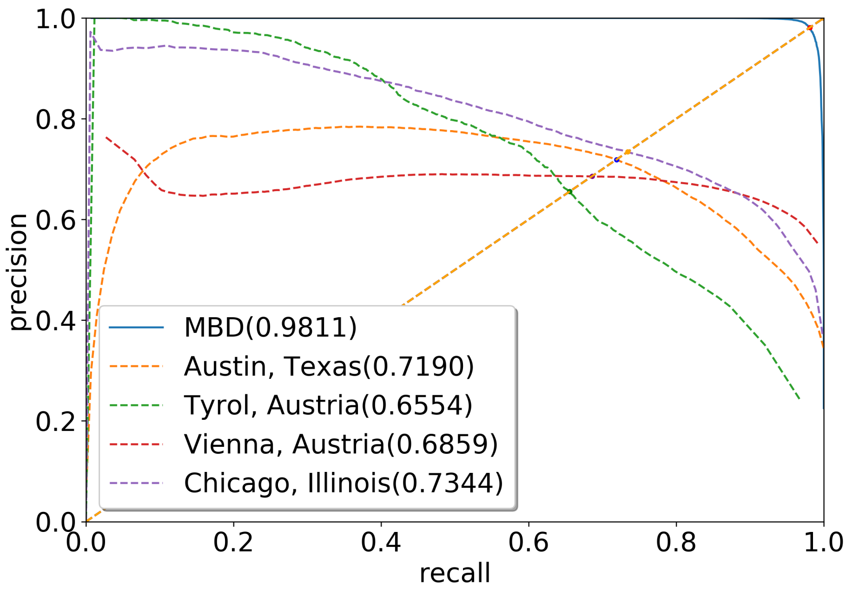


Figure 4.6: Precision-Recall curve obtained by applying MBD-trained model on a sample of Inria dataset. We applied the MBD trained model on a variety of cities included in the Inria data. The model generalises poorly to the Inria dataset. For comparison we show the curve from the MBD test data. Break-even points, where precision and recall values are equal, are mentioned in the legend.

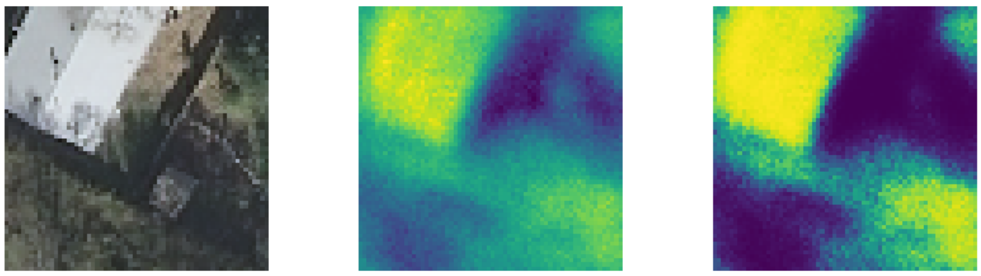


Figure 4.7: Image on the left is a patch from Inria. Image at the middle is maps predicted by the last (fully connected) layer of one of our models before the sigmoid function is applied. The image on the right is same as the middle figure but after sigmoid activation function is applied. Clearly, our model is able to learn some structure of the Inria dataset.

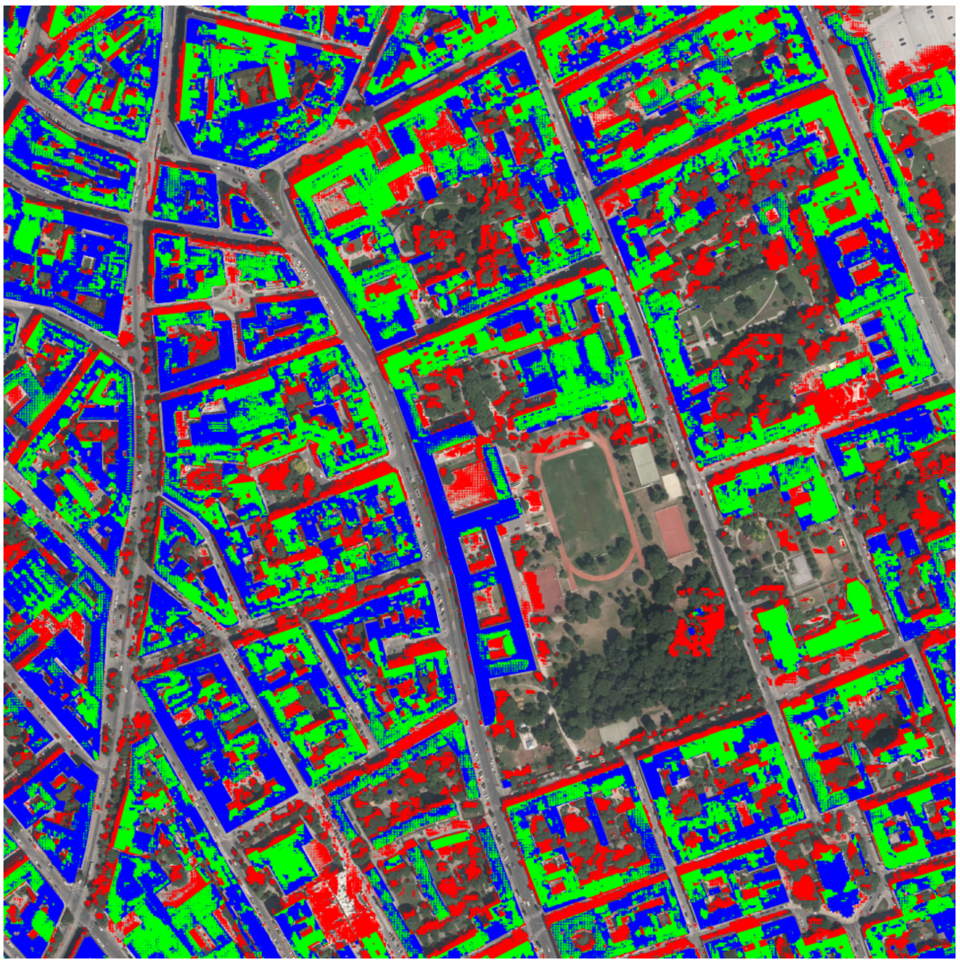


Figure 4.8: The predictions made by our MBD trained model (green parts in the image) are overlayed on one of the Inria dataset. The red parts are the false positives and the blue parts are the false negatives.

- visually illustrating the weak performance of the MBD trained model on the Inria dataset.

The poor generalization may largely be a result of difference between the structure of buildings found in the training set (MBD) and the Inria dataset.

Chapter 5

Conclusions and Future Work

In this thesis we presented the following:

- A review of aerial datasets and existing methods for labelling from them, paying special attention to the algorithms' performance.
- Drawing inspiration from [50], we investigated the impact of different hyper-parameter choices on our CNN architecture using the Masachusset Building Dataset.
- We studied the generalisation potential of MBD trained model.

In Chapter 3, we looked at different choices of feeding data into our models in terms of the amount of training, validation and test sets and presented our findings in Chapter 4. For all our analysis on overlapping patches we defined an epoch being 64,000 iterations of randomly selected patches out of more than a million possible ones. We used 20 epochs while testing for optimal hyper-parameter values. We used 100 epochs to train the main model as we observed a significant improvement in the precision-recall metric when increasing the total number of iterations - see Figure 4.2. We checked that increasing the number of epochs further does not improve our results.
In [50] it is shown that for L2 regularization the effect of the weight decay parameter is negligible, hence in our analysis we did not use weight decay.

Similarly, like [50] we used max-pooling sparingly. An attempt to do otherwise resulted in a model with inferior performance. Thus, we restricted its use to only the first convolution layer.

In Chapter 4, we presented the results obtained from training CNN using dfferent filter sizes. The use of small filter size in our network outperforms the larger ones used in [50]. Aside from the reduction in computation time, small filter size also improves the accuracy of the model in labelling buildings and non-building pixels. Using the optimal hyperparameters found from the different experiments we carried out, we achieved high accuracy on MBD testing dataset than presented in [50]. Our break-even point on the precision-recall curve is 95.68 compared to 92.11 for [50].

We also studied the generalisability of our model which is trained on MBD data, and applied it to other data sets in particular the Inria data set. The MBD and Inria datasets have a completely different geographic conditions and city structure, and, expectedly, our model performed poorly on Inria.

Possible ways to extend our work include:

- Refining the model predictions [13] - this is possibly the most cost-effective approach as it does not incur training cost.
- Fine-tuning the model's weights on a more diverse aerial dataset.
- Labelling African aerial dataset. A possible way to achieve this, in addition to the afore-mentioned, is to establish a good procedure for fitting our main model to the dataset as suggested in [50] - instead of the common procedure of using SGD on the negative log likelihood, the authors suggested searching for a better loss function such as the area under the precision-recall curve.
- While semantic labelling of building is already a useful application, it can further be used to facilitate urban management by counting the number of buildings in a given building map. A possible future path is to integrate the building counting operation into the training process.

List of References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning.

[2] R. Bajcsy and M. Tavakoli. Computer recognition of roads from satellite pictures. *IEEE Transactions on Systems, Man, and Cybernetics*, 1976.

[3] H. Bischof, W. Schneider, and A. J. Pinz. Multispectral classification of landsat-images using neural networks. *IEEE transactions on Geoscience and Remote Sensing*, 1992.

[4] T. Blaschke. What's wrong with pixels? some recent developments interfacing remote sensing and gis. *GeoBIT/GIS*, 2001.

[5] T. Blaschke. Object-based contextual image classification built on image segmentation. In *Advances in Techniques for Analysis of Remotely Sensed Data, 2003 IEEE Workshop on*. IEEE, 2003.

[6] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva. Land use classification in remote sensing images by convolutional neural networks. *CoRR*, 2015.

[7] G. Cheng, L. Guo, T. Zhao, J. Han, H. Li, and J. Fang. Automatic landslide detection from remote-sensing imagery using a scene classification method based on bovw and plsa. *International Journal of Remote Sensing*, 2013.

[8] G. Cheng, J. Han, and X. Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 2017.

[9] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[10] S. Cui and M. Datcu. Comparison of kullback-leibler divergence approximation methods between gaussian mixture models for satellite image retrieval. In *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*. IEEE, 2015.

[11] D. Dai and W. Yang. Satellite image classification via two-layer sparse coding with biased image representation. *IEEE Geoscience and Remote Sensing Letters*, 2011.

[12] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, 2016.

[13] J. W. Davis, C. Menart, M. Akbar, and R. Ilin. A classification refinement strategy for semantic segmentation. *arXiv preprint arXiv:1801.07674*, 2018.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[15] DigitalGlobe. https://www.digitalglobe.com.

[16] L. Drăguţ and T. Blaschke. Automated classification of landform elements using object-based image analysis. *Geomorphology*, 2006.

[17] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[18] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 2010.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

[20] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[22] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[23] J. Han, D. Zhang, G. Cheng, L. Guo, and J. Ren. Object detection in optical remote sensing images based on weakly supervised learning and high-level feature learning. *IEEE Transactions on Geoscience and Remote Sensing*, 2015.

[24] M. C. Han J. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *Lecture Notes in Computer Science, vol 930.* Springer, 1995.

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[26] F. Hu, G.-S. Xia, J. Hu, and L. Zhang. Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery. *Remote Sensing*, 2015.

[27] L. Huang, C. Chen, W. Li, and Q. Du. Remote sensing image scene classification using multi-scale completed local binary patterns and fisher vectors. *Remote Sensing*, 2016.

[28] J. Idelsohn. A learning system for terrain recognition. *Pattern recognition*, 1970.

[29] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[30] ISPRS. Web site of the isprs test project on urban classification and 3d building reconstruction. `http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html`, 2012.

[31] N. Jean, M. Burke, M. Xie, W. M. Davis, D. B. Lobell, and S. Ermon. Combining satellite imagery and machine learning to predict poverty. *Science*, 2016.

[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014.

[33] R. L. Kettig and D. Landgrebe. Classification of multispectral image data by extraction and classification of homogeneous objects. *IEEE Transactions on geoscience Electronics*, 1976.

[34] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] P. Kohli, P. H. Torr, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 2009.

[36] X. Li and G. Shao. Object-based urban vegetation mapping with high-resolution aerial photography as a single data source. *International journal of remote sensing*, 2013.

[37] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, 2014.

[38] L. Liu, Z. Pan, and B. Lei. Learning a rotation invariant detector with rotatable bounding box. *CoRR*, abs/1711.09405, 2017.

[39] L. Liu, Z. Pan, and B. Lei. Learning a rotation invariant detector with rotatable bounding box. *CoRR*, abs/1711.09405, 2017.

[40] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, 2015.

[41] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[42] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[43] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, 2013.

[44] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez. High-resolution semantic labeling with convolutional neural networks. *arXiv preprint arXiv:1611.01962*, 2016.

[45] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2017.

[46] A. Marcu. A local-global approach to semantic segmentation in aerial images. *CoRR*, abs/1607.05620, 2016.

[47] T. R. Martha, N. Kerle, C. J. van Westen, V. Jetten, and K. V. Kumar. Segment optimization and data-driven thresholding for knowledge-based landslide detection by object-based image analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 2011.

[48] M. L. Mekhalfi, F. Melgani, Y. Bazi, and N. Alajlan. Land-use classification with compressive sensing multifeature fusion. *IEEE Geoscience and Remote Sensing Letters*, 2015.

[49] N. B. Mishra and K. A. Crews. Mapping vegetation morphology types in a dry savanna ecosystem: integrating hierarchical object-based image analysis with random forest. *International journal of remote sensing*, 2014.

[50] V. Mnih. *Machine learning for aerial image labeling*. PhD thesis, University of Toronto (Canada), 2013.

[51] V. Mnih and G. E. Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 567–574, 2012.

[52] T. N. Mundhenk, G. Konjevod, W. A. Sakla, and K. Boakye. A large contextual dataset for classification, detection and counting of cars with deep learning. *CoRR*, 2016.

[53] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010.

[54] K. Nogueira, O. A. Penatti, and J. A. dos Santos. Towards better exploiting convolutional neural networks for remote sensing scene classification. *Pattern Recognition*, 2017.

[55] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 2002.

[56] O. A. Penatti, K. Nogueira, and J. A. dos Santos. Do deep features generalize from everyday objects to remote sensing and aerial scenes domains? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015.

[57] S. R. Phinn, C. M. Roelfsema, and P. J. Mumby. Multi-scale, object-based image analysis for mapping geomorphic and ecological zones on coral reefs. *International Journal of Remote Sensing*, 2012.

[58] K. Qi, H. Wu, C. Shen, and J. Gong. Land-use scene classification in high-resolution remote sensing images using improved correlatons. *IEEE Geoscience and Remote Sensing Letters*, 2015.

[59] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, 2015.

[60] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, 2015.

[61] S. Saito and Y. Aoki. Building and road detection from large aerial imagery. In *SPIE/IS&T Electronic Imaging*, pages 94050K–94050K. International Society for Optics and Photonics, 2015.

[62] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[63] G. Sheng, W. Yang, T. Xu, and H. Sun. High-resolution satellite scene classification using a sparse coding based multiple feature combination. *International journal of remote sensing*, 2012.

[64] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European conference on computer vision*. Springer, 2006.

[65] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[66] A. Stumpf and N. Kerle. Object-oriented mapping of landslides using random forests. *Remote Sensing of Environment*, 2011.

[67] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[68] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, 2014.

[69] J. Uhrig, M. Cordts, U. Franke, and T. Brox. Pixel-level encoding and depth layering for instance-level semantic labeling. *CoRR*, 2016.

[70] J. S. Walker and J. M. Briggs. An object-oriented classification of an arid urban forest with true-color aerial photography. Citeseer.

[71] J. S. Walker and J. M. Briggs. An object-oriented approach to urban forest mapping in phoenix. *Photogrammetric Engineering & Remote Sensing*, 2007.

[72] H. Wu, B. Liu, W. Su, W. Zhang, and J. Sun. Hierarchical coding vectors for scene level land-use classification. *Remote Sensing*, 2016.

[73] G. Xia, J. Hu, F. Hu, B. Shi, X. Bai, Y. Zhong, and L. Zhang. AID: A benchmark dataset for performance evaluation of aerial scene classification. *CoRR*, 2016.

[74] G.-S. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang. Dota: A large-scale dataset for object detection in aerial images. In *IEEE CVPR*, 2018.

[75] G.-S. Xia, Z. Wang, C. Xiong, and L. Zhang. Accurate annotation of remote sensing images via active spectral clustering with little expert knowledge. *Remote Sensing*, 2015.

[76] M. Xie, N. Jean, M. Burke, D. B. Lobell, and S. Ermon. Transfer learning from deep features for remote sensing and poverty mapping. *CoRR*, abs/1510.00098, 2015.

[77] Y. Yang and S. Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*. ACM, 2010.

[78] B. Zhao, Y. Zhong, G.-S. Xia, and L. Zhang. Dirichlet-derived multiple topic scene classification model for high spatial resolution remote sensing imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 2016.

[79] B. Zhao, Y. Zhong, L. Zhang, and B. Huang. The fisher kernel coding framework for high spatial resolution scene classification. *Remote Sensing*, 2016.

[80] L. Zhao, P. Tang, and L. Huo. Feature significance-based multibag-of-visual-words model for remote sensing image scene classification. *Journal of Applied Remote Sensing*, 2016.

[81] Q. Zhu, Y. Zhong, B. Zhao, G.-S. Xia, and L. Zhang. Bag-of-visual-words scene classifier with local and global features for high spatial resolution remote sensing imagery. *IEEE Geoscience and Remote Sensing Letters*, 2016.

[82] J. Zou, W. Li, C. Chen, and Q. Du. Scene classification using local and global features with collaborative representation fusion. *Information Sciences*, 2016.

[83] Q. Zou, L. Ni, T. Zhang, and Q. Wang. Deep learning based feature selection for remote sensing scene classification. *IEEE Geoscience and Remote Sensing Letters*, 2015.