Masters Program in Geospatial Technologies

IMPROVED FULLY CONVOLUTIONAL NETWORK WITH CONDITIONAL RANDOM FIELD FOR BUILDING EXTRACTION

Sanjeevan Shrestha

Dissertation submitted in partial fulfilment of the requirements
for the Degree of *Master of Science in Geospatial Technologies*

NOVA IMS Information Management School
Universitat Jaume I
Westfälische Wilhelms-Universität Münster

IMPROVED FULLY CONVOLUTIONAL NETWORK WITH CONDITIONAL RANDOM FIELD FOR BUILDING EXTRACTION

Dissertation supervised by:
Leonardo VANNESCHI , PhD
Instituto Superior de Estatística e
Gestão de Informação,
Universidade Nova de Lisboa,
Lisbon, Portugal

Co-supervised by:
Florian HILLEN , PhD
IP SYSCON GmbH,
Hannover, Germany

Co-supervised by:
Lledó MUSEROS , PhD
Universitat Jaume I,
Castellon de la Plana, Spain

February 28, 2018

ACKNOWLEDGEMENTS

It's with profound gratitude that I express my due regards to **Erasmus Mundus Program** for giving me an opportunity and funding my studies. I want to express my deepest gratitude to my main supervisor **Dr. Leonardo Vanneschi** for continuous encouragement and criticism in inspiring to work to the fullest level with proper guidance and support. Additionally, I would like to express heartfelt gratitude to the co-supervisors, **Dr. Florian Hillen** and **Dr. Lledó Museros** for advice and critical comments. I am equally thankful to **Prof. Dr. Marco Painho** for constant monitoring, critical analysis, feedbacks, guidance, suggestion, and encouragement throughout the thesis period. Their ideas and suggestions were always guidance to me during thesis period.

I also want to take this opportunity to convey a deep sense of gratitude to **Mr. Luis Alves** and **Mr. Mario David**, from **LIP - Laboratório de Instrumentação e Física Experimental de Partículas**, for their untiring effort and help during accessing server for simulation in my thesis.

I would always be grateful to **my parents**, my brother **Jeevan**, **Tina** and **Bishruti** for being there for me through all my thick and thin. Without your love, generosity, and support, this would not have been possible. Thank you for having faith and hope in me. I am also thankful to **Rojika**, **Binaya** and **Megha** for proof-reading the thesis. Thankful to all my friends, especially **Stavros**, **Nikolina** and **Mulu** for their support in every aspect to keep me moving forward. Finally, I would like to thank my family back home and my extended family here, for their contribution and encouragement throughout the study.

IMPROVED FULLY CONVOLUTIONAL NETWORK WITH CONDITIONAL RANDOM FIELD FOR BUILDING EXTRACTION

ABSTRACT

Building extraction from remotely sensed imagery plays an important role in urban planning, disaster management, navigation, updating geographic databases and several other geospatial applications. Several published contributions are dedicated to the applications of Deep Convolutional Neural Network (DCNN) for building extraction using aerial/satellite imagery exists; however, in all these contributions a good accuracy is always paid at the price of extremely complex and large network architectures. In this paper, we present an enhanced Fully Convolutional Network (FCN) framework especially molded for building extraction of remotely sensed images by applying Conditional Random Field (CRF). The main purpose here is to propose a framework which balances maximum accuracy with less network complexity. The modern activation function called Exponential Linear Unit (ELU) is applied to improve the performance of the Fully Convolutional Network (FCN), resulting in more, yet accurate building prediction. To further reduce the noise (false classified buildings) and to sharpen the boundary of the buildings, a post processing CRF is added at the end of the adopted Convolutional Neural Network (CNN) framework. The experiments were conducted on Massachusetts building aerial imagery. The results show that our proposed framework outperformed FCN baseline, which is the existing baseline framework for semantic segmentation, in term of performance measure, the F1-score and Intersection Over Union (IoU) measure. Additionally, the proposed method stood superior to the pre-existing classifier for building extraction using the same dataset in terms of performance measure and network complexity at once.

KEYWORDS

Building Extraction

High Resolution Aerial Imagery

Deep Learning

Deep Convolutional Neural Network

Fully Convolutional Network

Conditional Random Field

ACRONYMS

ANN	Artificial Neural Network.
CIS	Channel-wise Inhibited Softmax.
CNN	Convolutional Neural Network.
CRF	Conditional Random Field.
DCNN	Deep Convolutional Neural Network.
ELU	Exponential Linear Unit.
FAA	US Federal Aviation Administration.
FCN	Fully Convolutional Network.
FCRF	Fully Connected Conditional Random Field.
FN	False Negative.
FP	False Positive.
GAP	Global Average Pooling.
GTLabel	Ground Truth Label.
ILSVRC	ImagNet Large-Scale Visual Recognition Challenges.
IoU	Intersection Over Union.
LIDAR	Light Detection and Ranging.
LReLU	Leaky Rectified Linear Unit.
MRF	Markov Random Field.
NRG	Near-Infrared, Red, Blue.

OSM OpenStreetMap.

PASCAL VOC PASCAL Visual Object Classes.

PCA Principal Component Analysis.

RBM Restricted Boltzmann Machine.

ReLU Rectified Linear Unit.

RGB Red, Blue, Green.

SLIC Simple Linear Iterative Clustering.

SVM Support Vector Machine.

TN True Negative.

TP True Positive.

UAS Unmanned Aerial System.

UAV Unmanned Aerial Vehicle.

VGI Volunteered Geographic Information.

INDEX OF THE TEXT

INDEX OF TABLES

INDEX OF FIGURES

1 INTRODUCTION

1.1 Thesis Context

Over the past decade, with the rapid growth in the remote sensing/satellite imaging technology, high resolution satellite imageries are readily available. With established companies such as Digital Globe and IMAO already offering frequent data at high spatial resolution (up to 31 cm for worldview-3), the use of satellite imagery for scenery extraction has been revamped [Globe (2017)]. In the meantime, the concept of open data for disaster recovery by providing image freely to immediately support disaster relief [Globe (2017)] has been proposed and already being implemented. New companies such as Planet Labs are also joining in this race by launching 20^{th} satellite in the constellation of micro-satellites in the orbit with the goal of collecting high-resolution imagery on daily basis [Planet (2017)]. Between 2012 and 2016, 364 remote sensing satellites were delivered worth $8.79 billion and projected to deliver 951 satellites between 2017 and 2022, worth $16.52 billion. On top of that, the imaging capability is further boosted to sub-meter spatial resolution by merging Google based SkySat earth imaging satellite to Planet Labs [Planet (2017)]. To encourage the advancement, the US government has lifted restriction on trade of 25cm satellite imagery [News (2017)] and allowed commercial use of Unmanned Aerial Vehicle (UAV)s [FAA (2017)]. US Federal Aviation Administration (FAA) is encouraging partnership between government bodies and private sector for advanced Unmanned Aerial System (UAS) operation [FAA (2017)]. Hence, it is plausible that the rapidly growing abundance and sophistication of satellite imagery and remote sensing data can answer big questions.

Past studies show that an affordable access to massive amount of high-resolution aerial/satellite imagery with high revisit time is plausible over the coming decades. This could enable object extraction from the image of earth surface with a high degree of accuracy to provide reliable information for real field applications. One of the potential applications can be the reliable extraction of a small ground feature (i.e. buildings) with only sub-decimeter coverage. Extracting building image from satellite imagery will certainly benefit urban planning, disaster management, navigation, updating the geographic database and several other geospatial applications [Mayer (1999), Krizhevsky et al. (2012)]. To enable such quantification and analysis using geographic information systems, raw image should be transferred into tangible information [Shu (2014)]. This transformation often comes with labor intensive and time-consuming process of digitization or interpretation of information contained within the image. Although the introduction of Volunteered Geographic Information (VGI) technique has emerged as an alternative source [Yuan (2016)], the usability of VGI is somehow limited due to variation in completeness and positional accuracy. The main reason could be 'participation inequality' in terms of varying judgement, cultural difference and impression. So, this limits the availability of up-to-date and reliable building map, and the information contained in new image data to those who really need it the most.

Developing reliable methods to automatically extract object (i.e. building) from HSR imagery is essential to support preparation of building map. Despite a decade of research in this area, a promising method is not yet developed for reliable and automatic extraction of individual buildings using an aerial/satellite image [Yuan (2016), Marcu and Leordeanu (2016)]. Large variations of building appearances in an image due to different characteristics of buildings like different roof material, different structure, different illuminating condition, occlusion and shadows cast by buildings are major factors that make this process challenging [Yuan and Cheriyadat (2014)].

Recent works have shown that feature based deep learning approaches such as CNN could be a promising state of art technique for semantic classification for both satellite imagery [Krizhevsky et al. (2012), Alshehhi et al. (2017), Yu et al. (2016), Marcu and Leordeanu (2016), Mnih (2013), Bittner et al. (2017), Vakalopoulou et al. (2015)] as well as computer vision [Lin et al. (2013), Szegedy et al. (2015), He et al. (2015)]. Deep Convolutional Neural Network (DCNN) architecture has become a notable method due to its capability to effectively combine spectral and spatial information based on the input image without preprocessing [Alshehhi et al. (2017)]. The following major characteristics of CNN are the reasons that are making CNN superior to other classical classification techniques.

- They are made up of a set of adaptive filters which can learn their weight directly from raw input without any preprocessing; making them more efficient and automatic in nature [Krizhevsky et al. (2012)].
- They have great ability for automatic feature extraction and efficient high feature learning because they are non-parametric in nature unlike traditional classification methods which rely on manual feature extraction [O'Shea and Nash (2015)].
- They bear ability to process large amount of data as well as learn from difficult data [Krizhevsky et al. (2012)].

1.2 Research Gap

The increasing development in remote sensing community described above is also associated with different problems. The problem associated with this data era (preferably termed as "big data") is allied with two of three dimensions (3V's) of the remote sensing imagery namely; volume, velocity, and variety [Casado and Younas (2015)]. Data volume is the first dimension which is becoming problematic with increasing number of high resolution images. Secondly, data velocity problem is amplified with the increasing number of platforms offering the high revisit time such as Planet Lab and UAV.

Additionally, although the number of publications show that there is an improvement in the field of building extraction using CNN architectures, it seems that every research is dedicated only towards an increase in accuracy of classification. They put only little concentration on lessening of network complexity in conjunction with a gain in performance. This plays a huge impact in some scenarios such as real time semantic segmentation, which demands low computational complexity.

Considering these scenarios, the following key issues can be identified in building extraction using remote sensing images.

- Streamlining the classification pipeline, possibly eliminating inefficient steps and automating every component
- Learning from difficult, high volume and high velocity data source
- Increasing accuracy of classifier, not compromising the computational efficiency

With growing amount of big data in remote sensing application, it is imperative to have robust algorithms to deal with these issues for reliable feature extraction. So, the main goal of the thesis is to develop a deep feature learning approach addressing these problems in building extraction using high-resolution aerial images.

1.3 Research Objectives

The research primarily aims to formulate a method for building extraction from high-resolution remote sensing imagery exploiting deep learning approach-based on convolution neural network.

To fulfill the aim, the specific objectives are:

- To review and evaluate the potential of state-of-art deep learning algorithms for automatic building extraction using high-resolution aerial/satellite imagery.
- To design, implement and analyze the performance of a working streamlined classifier based on the chosen CNN based deep learning algorithm.
- To compare the performance of the proposed deep learning classifier against alternative classification methods.

1.4 Innovation

Only limited research has been done with deep feature learning in the context of remote sensing as most of its popular applications are in the field of computer vision and natural language processing. The use of deep learning algorithms, especially for building extraction using high-resolution aerial/remote sensing imagery is further confined. Among them, most of the related researchers are always concentrated towards getting higher performance in term of accuracy. Moreover, specific details such as effect of variations in hyper-parameters used in deep learning algorithms network were not thoroughly investigated, especially in the case of building extraction problem using aerial/satellite imagery. Up to the author's knowledge, the effect of different activation functions on the performance of the network, focusing on building extraction is not explored yet.

The thesis exploits the use of deep learning to prototype a classifier for building extraction from high-resolution aerial images. Ideally, this classifier will be able to extract building from high-resolution aerial imagery in an automated manner. A FCN is implemented for building extraction considering accuracy and complexity of the network as optimization parameter. The FCN uses ELU activation function in place of Rectified Linear Unit (ReLU) activation function along with post processing CRF at the end with an expectation to increase the performance of the network. To validate this, the performance of the proposed method is evaluated against its variations and benchmarks. Several attempts have been made to extract building using the same dataset using deep learning algorithms but better classification results in terms of performance measures as well as in terms of network complexity are expected from the proposed classifier. However, there is

no literature which compares FCN with pre-existing classifiers for building extraction.Thus, its performance is evaluated against other pre-existing deep learning-based classification methods.

1.5 Assumptions

The thesis adopts existing FCN network of semantic segmentation and tailors it for building extraction. The existing FCN network for semantic segmentation is based on ReLU activation function whereas the thesis adopted ELU activation function in this network. The preliminary assumption is that it will improve the accuracy of network as well as lower computational complexity. Moreover, the use of the CRF technique as post processing technique will enhance the boundary of output map as well as the accuracy of the output. The thesis hereby propose robust classifier for building extraction for improved accuracy and lower computational complexity.

1.6 General Methodology

The thesis is divided into three stages namely; i) review and evaluation; ii) design, implementation, and analysis; iii) performance comparison. The detailed structure of overall methodology is shown in Figure 1.1.

In the first stage, several existing CNN architectures of deep learning algorithms for semantic segmentation were reviewed. The focus of the review was to discuss the applicability of these algorithms for extracting building footprint using high resolution satellite imagery. Existing literatures and result of benchmark algorithms were analyzed for comparison and evaluation. The suitable algorithm was chosen for the research. Similarly, boundary enhancement techniques as well as activation functions used in the CNN architecture were reviewed and the most applicable one was chosen.

The second stage deals with design, implementation and analysis of a classifier based on chosen deep learning algorithm. Initial architecture was designed using the concept of FCN network framework. Several experimental designs were conducted for selecting the optimal value of hyper-parameters. The purpose of this sensitivity analysis is to understand the effect of the hyper-parameters on the performance. The sensitivity of classifier to this change in parameters on performance metrics and computational time was studied. For this, benchmark Massachusetts building dataset prepared by [Mnih (2013)] was used as an input. To obtain final building binary mask, the fully connected CRF based boundary enhancing algorithm is added in conjunction with this classifier. Similar nature of experiments were run to achieve optimal value of parameters for CRF algorithm. Finally, we finalized the classifier based on the knowledge gained with the change in parameters.

In the third stage, we compared the performance of the modified deep learning approach with its alternative approaches using qualitative and quantitative approach. For quantitative approach, several metrics such as overall accuracy, precision, recall, F1-score, intersection over union (IoU) as well as computational time and complexity were used as a measure to evaluate the performance of each classification approach. Additionally, visual inspection and comparison of the classification output is performed for qualitative evaluation. This process evaluates the assumptions made in the thesis and determines the best algorithm which is used in rest of the evaluation process. Finally, to demonstrate the performance of our model, the result from the chosen model is compared with

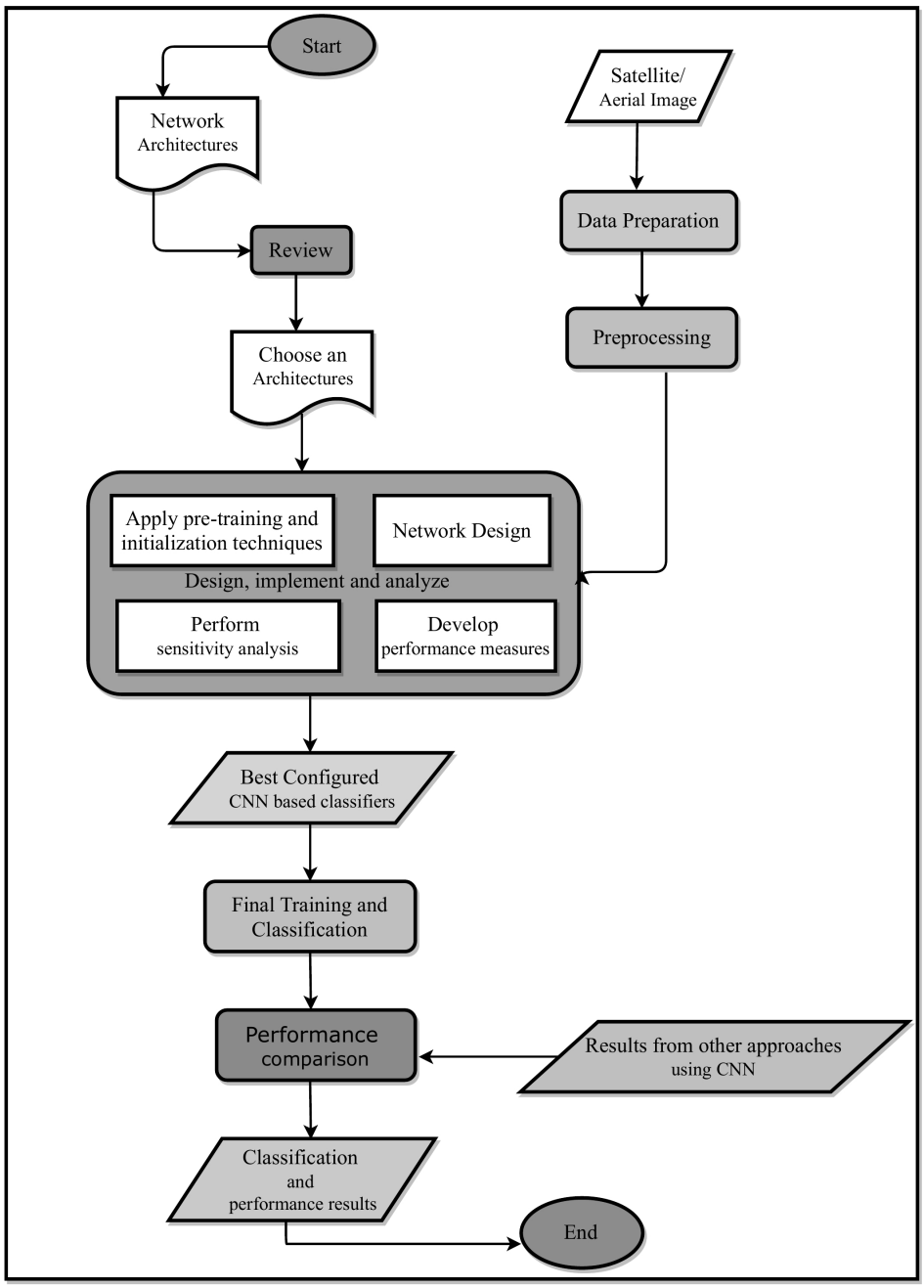


Figure 1.1: Overall Methodology of the thesis. The dark shaded box represents tasks, directly corresponds with objectives; other fair dark box represents tasks which support to fulfill the objectives.

the result from existing literatures based on the same dataset. The comparison is based on the performance measure and network complexity adopted by the existing literature.

1.7 Thesis Organization

The thesis is divided into 8 chapters. Chapter 1 introduces the thesis context, research problems and research objectives considered in the thesis. Chapter 2 reviews the development of the existing methods for building extraction using aerial/remote sensing imagery. Additionally, this chapter presents several potential deep learning algorithms for building extraction (i.e. semantic segmentation) as well as activation functions and boundary enhancing algorithms. The end of the Chapter 2 presents chosen CNN algorithms to be implemented as the classifier. Chapter 3 presents the theoretical background of deep convolutional neural network architecture and training. Additionally, this chapter elaborates the concept of transfer learning and post processing classification technique. Finally, this chapter review pre-existing classifiers which are later compared with the proposed technique in this thesis. Chapter 4 elaborates the formulation of the chosen CNN-based classifier and the CRF algorithm for post classification processing. Chapter 5 deals with the description of the datasets used along with data preparation, preprocessing, CNN hyper-parameter sensitivity experiments and CRF parameter tuning experiments. Chapter 6 includes a description of the final configuration of variations of the selected CNN classifier, several approaches used for comparison. Chapter 7 summarizes the finding from the implementation, performance analysis and comparison experiments conducted. Finally, Chapter 8 deals with conclusions and recommendations for the future work based on the findings.

2 LITERATURE REVIEW

This chapter presents a brief review of existing deep learning algorithms in the context of building extraction from remote sensing imagery to select a classifier. This first section deals with the evolution of building extraction using aerial/satellite imagery. This is further sub-divided into classical and deep learning approach for building extraction followed by a sub-section which reviews deep learning approach for semantic segmentation. This chapter also presents the recent improvement in activation function, a part of CNN architecture as well as post processing algorithms. Final sub-section presents chosen CNN algorithms to be implemented as the classifier.

2.1 Building Extraction using Remote Sensing Imagery

The building extraction techniques using aerial photos or high-resolution satellite imagery can be broadly divided into two categories i.e. classical approach and deep learning approach. The description of each category is detailed below.

2.1.1 Classical Approach for Building Extraction

The traditional approaches use handcrafted features as a key feature for building extraction [Huertas and Nevatia (1988), Peng and Liu (2005), Karantzalos and Paragios (2009), Kim and Muller (1999), Levitt and Aghdasi (1998)]. The pioneering approach used spectral information to detect and extract edge, borderline and corners of building extraction [Huertas and Nevatia (1988)]. In addition to this, some research utilized shadow information as a low-level feature [Peng and Liu (2005)]. Kim and Muller (1999) applied a similar methodology based on line analysis but by using graph-based approach for automatic building extraction. Many researchers used the spatial feature information (e.g. texture, structure and context) such as combination of morphological building index with shadow index [Huang et al. (2016)], wavelet texture [Levitt and Aghdasi (1998)] and the gray level co-occurrence matrix (GLCM) [Myint et al. (2004)] to extract building footprint. Further research [Karantzalos and Paragios (2009)] utilized the prior knowledge of the geometrical structure of building during detection phase for building extraction. Similarly, multi-spectral properties of remote sensing imagery is applied for building detection and classification using support vector machine in [Inglada (2007)] and genetic algorithms in [Sumer and Turker (2013)]. Beside these, height information of building using DSM (extracted from stereo optical imagery or Light Detection and Ranging (LIDAR)) alone [Lafarge et al. (2010)] or in combination with remote sensing imagery [Gerke et al. (2001)] is used for delineation and extraction of the building boundary. The performance of these approaches relies on the extraction of low-level hand engineered local features e.g. local structure (edge, line, corners), color histogram and texture features [Hu et al. (2015)]. This limits representative ability, thereby restricting their performance. Therefore, extraction and use of more representative high-level features are desirable which can substantially discriminate the feature and play a dominant role in image segmentation.

2.1.2 Deep Learning for Building Extraction

With tremendous improvement in CNN as a state of art technology in object recognition field, there is a significant improvement in research in the field of semantic pixel-based classification for building extraction. The ability of deep learning algorithms to learn hierarchical feature corresponding to the different levels of abstraction makes it dominant in the field of building extraction. In this section, some promising CNN approaches for extracting building and road from aerial or satellite imagery are discussed, and their main contributions are highlighted.

Mnih (2013) proposed patch-based CNN approach for building and road extraction separately, which is considered as pioneering work in the field of CNN for building extraction. The aerial imagery of Massachusetts dataset of spatial resolution of 1 m divided into image patches of $64 \times 64m$ dimension is used as input. The CNN input is extracted from the Principal Component Analysis (PCA) to reduce a dimensionality of an original image. These PCA vectors are used for fine-tuning Restricted Boltzmann Machine (RBM) to extract buildings and road network. CRF is used as post-processing technique to refine the previous output for final building layer [Mnih (2013)]. Shu (2014) experimented the performance of object-based segmentation CNN method on final result rather than patch-based CNN using same architecture, but with orthorectified Red, Blue, Green (RGB) imagery of spatial resolution of $12cm$. A bottom-up DCNN with top-down object modeling is proposed for building extraction in the research [Shu (2014)]. Saito and Aoki (2015) and Saito et al. (2016) used a single CNN architecture to extract road and building simultaneously using Mhin imagery dataset. The capability of CNN for multi-channel semantic segmentation (of building, roads, and background simultaneously) using single patch-based CNN is demonstrated in the research [Saito and Aoki (2015), Saito et al. (2016)]. Former research [Saito and Aoki (2015)] used an additional MAXOUT layer with dropout optimization instead of ReLU to increase the performance of CNN while latter research [Saito et al. (2016)] used model averaging with spatial displacement technique for semantic segmentation and Channel-wise Inhibited Softmax (CIS) function to suppress the effect of the background.

Alshehhi et al. (2017) also applied single deeper patch-based CNN architecture for extraction of road and building simultaneously. Global Average Pooling (GAP) layer is used instead of fully connected layer realizing the issue of fully connected layer diminishing localization ability. New post-processing method based on low-level spatial feature (adjacent Simple Linear Iterative Clustering (SLIC) regions) is used to enhance CNN output. All of the results showed an excellent outcome in extracting building using aerial or high-resolution satellite imagery. However, the patch-based CNN network works good at extracting individual houses but does not perform well on larger and complex building which is imperative in the case of urban scene [Huang et al. (2016)]. Moreover, patch-based network shows the existence of discontinuities border of output probability patches. This shows the patch-based network is incapable of learning to classify pixel independent of their location inside patches [Maggiori et al. (2016)]. The use of ultra-high-resolution imagery (i.e. sub-decimeter resolution) in this approach is problematic because small patches tend to cover fragmented building and thus, fails to capture complete information of individual buildings [Yuan (2016)].

Realizing the issue of patch based CNN network, Vakalopoulou et al. (2015) demonstrated supervised building extraction procedure based on the ImageNet[1] framework [Krizhevsky et al. (2012)].

[1] **ImageNet:** Largest Inventory of images for visual object recognition research

Spectral information is integrated by employing multispectral band combination into the training procedure. Building detection was addressed through a binary classification procedure based on Support Vector Machine (SVM) classifier and refined by solving Markov Random Field (MRF) problem. However, the patch-based sliding window is still applied for testing and training purpose which is time consuming. Additionally, use of fully connected layer at end discard spatial information at a finer resolution which is crucial for dense prediction [Huang et al. (2016)]. Maggiori et al. (2016) uses the similar architecture suggested by Shu (2014) but used pixel-based FCN to produce a dense prediction. They added a deconvolutional layer, which learns filters for up-sampling into the original resolution of input image to increase the resolution of an output map for dense pixel-based classification. For robust training, use of possibly inaccurate reference data to train initially and refinement on a small amount of manually labeled data is applied. This process eliminates the discontinuity issue as well as improves accuracy due to a simplified learning process and lower execution time. Yuan and Cheriyadat (2014) used the similar approach as that of Maggiori et al. (2016) for extraction of the buildings. Convolutional Network (ConvNet) framework is employed integrating multistage feature map using upsampling techniques. Huang et al. (2016) used multisource remote sensing imagery provided by IEEE GRASS data fusion contest with ground truth from OpenStreetMap (OSM). Supervised extraction of buildings is obtained by using the deconvolutional neural network with decoder and encoder architecture. Pre-training of the Deep Deconvolutional Neural Network (DeCNN) by using public large-scale Massachusetts building dataset and further fine-tuned by using two band combination (RGB and Near-Infrared, Red, Blue (NRG)), and fused together to accurately extract building. Marcu and Leordeanu (2016) proposed dual stream deep network model to extract buildings using two independent pathways, one for local and another for global context and later combined in final layer processing. VGG-net [Simonyan and Zisserman (2014)] is used due to its capability to detect local and object level information due to smaller filter size while Alex-net [Krizhevsky et al. (2012)] is used as it considers information from large area around object of interest due to large filter size and later combined, which composed of three fully connected layers. Bittner et al. (2017) used a bit different dataset and proposed DSM based building extraction technique using FCN. Fine tuning is done on FCNs, proposed by Long et al. (2015) constructed based on VGG-16 networks [Simonyan and Zisserman (2014)]. Finally, binary building mask is obtained by using CRF technique using Fully Connected Conditional Random Field (FCRF) software.

2.2 Deep Learning for Semantic Segmentation

Building extraction can be considered as one of the semantic segmentation problems. So, it is worth discussing recent trend in semantic segmentation using deep learning.

Semantic segmentation algorithms are parts of computer vision community to deal with pixel-wise labeling problem. With an evolution of Deep Convolutional Neural Network (DCNN), this has now become state of art technology for modeling and extracting the feature hierarchy. A major breakthrough came when Long et al. (2015) proposed FCN network for semantic segmentation. They adapt existing contemporary classification networks namely AlexNet [Krizhevsky et al. (2012)], VGGNet [Simonyan and Zisserman (2014)] and GoogleNet [Szegedy et al. (2015)] into the fully DCNN and transfer their learned representations by fine tuning to the segmentation task. The inherent tension between semantics and location that was hovering around semantic segmentation is solved by jointly encoding them in a local-to-global pyramid. In this method, skip architecture is used skipping 3 layers, namely layer 3 (FCN – 8s), layer 4 (FCN – 16s) and layer 5 (FCN-32s). This

architecture reduces overfitting and improves performance up to 20% reaching 62.2% in experiments in PASCAL Visual Object Classes (PASCAL VOC) 2012 dataset [Liu et al. (2017)]. However, low performance as well as loss of the detailed structure of the object or smoothed is persistent in this model. To mitigate this limitation, Noh et al. (2015) proposed novel semantic segmentation algorithm by learning a deconvolution network on the top of the convolutional layers adopted from the VGG-16 network [Simonyan and Zisserman (2014)]. The deconvolution network comprises of deconvolution and unpooling layers to solve pixel-wise labeling problem as well as segmentation task. This proposed method showed outstanding performance achieving 72.55% accuracy in PASCAL VOC 2012 dataset. New state of art technology at the PASCAL VOC 2012[2] semantic image segmentation task is set by Chen et al. (2016), reaching 79.7% accuracy. Chen et al. (2016) proposed DeCNN based on either VGG-16 [Simonyan and Zisserman (2014)] or ResNet-101 model [He et al. (2015)]. Existing trained model for image classification is re-investigated to the task of image segmentation by transforming all fully connected layer to convolutional layers increasing feature resolution through Atreus[3] convolutional layers and using fully connected CRF to refine segmentation results. However, this proposed model also failed to capture the delicate boundaries of the object and even could not recover by the CRF post-processing. Badrinarayanan et al. (2017) put forward the novel and practical deep fully CNN architecture, consisting encoder network and corresponding decoder network followed by pixel-wise classification layer for semantic pixel-wise segmentation. The topology of this model is similar to the 13 convolutional layers in the VGG-16 network. Urban scene benchmark dataset such as CamVid [4] is used for road scene and indoor scene segmentation and stood superior outperforming all existing techniques of semantic segmentation.

2.3 Improvements in Deep learning

The recent improvement in some components of DCNN also contributed to improving the accuracy of DCNN output. Some of them which are relevant to our research are discussed below.

2.3.1 Activation Function in Deep Learning

The activation function is the important factor for DCNN considering its role to improve the performance of a network. ReLU is considered as popular activation function and used in most of the DCNN [Audebert et al. (2016), Badrinarayanan et al. (2017), Chen et al. (2016), He et al. (2016), Krizhevsky et al. (2012),Liu et al. (2017), Long et al. (2015), Noh et al. (2015), Ronneberger et al. (2015), Simonyan and Zisserman (2014), Szegedy et al. (2015)] for its capability to alleviate vanishing gradient problem [He et al. (2015)]. But, sometimes due to the characteristics of ReLU having a mean activation larger than zero may cause bias shift. ReLU has one drawback as it may permanently kill the neuron and never be activated if an activation is below zero. Clevert et al. (2015) devised the ELU for faster and more precise learning in DCNN leading to higher classification accuracies. Experiments in various benchmark dataset show that ELUs significantly outperform ReLU and the Leaky Rectified Linear Unit (LReLU) in faster learning and generalization performance as well as substantially increases the learning time using ImageNet. Shah et al. (2016) also proposed ELU in the residual network which learns faster and showed superior performance than original residual networks.

[2] **PASCAL VOC 2012:** Competition for visual object classification in natural scene

[3] **Atreus convolutional layer:** Devised by Chen et al. (2016) which is one type of convolutional layer for generating dense feature map using principle of wavelet transform.

[4] **CamVid:** collection of video datasets with object class semantic labels, complete with metadata

2.3.2 Post Processing Technique of Deep Learning

Recently, there is increasing trend of extending deep neural networks to increase performance combining it with another classifier as a post-processing step. Generally, DCNN architectures produce typically global smooth classification results, which decrease performance especially in case of image segmentation [Chen et al. (2016)]. Performance of DCNN in image segmentation especially in case of building segmentation is increased by enhancing local structure (object boundaries). In this case, CRF approach [Krähenbühl and Koltun (2011)] has been reported successful in increasing accuracy of DCNN by enhancing object boundaries and leads to substantial improvement over unstructured post processing neural networks [Bittner et al. (2017), Chen et al. (2016), Krähenbühl and Koltun (2011), Mnih (2013)].

2.4 Choice of Network Architecture

Exploring the trend of building extraction algorithm explained in section 2.1, it is confirmed that CNN can perform better than other algorithms for automatic building extraction from remote sensing imagery. So, CNN was chosen as a core algorithm for the thesis. In addition, recent research in semantic segmentation gives rise to several algorithms which solely depends on the CNN architectures. The FCN algorithm based on CNN network was selected for the thesis as it is based on dense image prediction. Compared to patch-based CNN that has been used in building extraction, the advantages are obvious for easy implementation, higher accuracy and less expensive computation. The FCN architecture is designed in such a way that enables to take any arbitrarily sized image as inputs. This enables training entire image rather than patch cropping, which reduces extra effort to rearrange output label together for label prediction, thus reducing implementation complexity. Moreover, only intra-patch information is taken into consideration rather than inter-patch information leading gap between the patches. So, the prediction at the edge of each patch is of low accuracy. Unlike patch-based CNN, classification on FCN is done in single loop manner increasing high quality prediction. Lastly, there is redundant computational work in patch based CNN as they use overlapped patches for dense pixel prediction increasing computational work [Fu et al. (2017)].

Similar algorithms baseed on dense image prediction which gave better results in PASCAL VOC 2012 semantic image segmentation task, the FCN architecture is chosen because of its simplicity and flexibility for computation. Other algorithms are based on the encoder decoder architecture (SegNet [Badrinarayanan et al. (2017)], DeconvNet [Noh et al. (2015)]), which is twice deep containing a lot of associated parameters, resulting in a harder optimization problem. Additionally, they contain deconvolution layers for unpooling which requires the pooled location of the pooling operations to be stored. This is computationally intensive increasing memory requirements. The DeconvNet [Noh et al. (2015)] also returns noisy prediction when the label data is not perfectly aligned with training image which is likely to occur in the used dataset. In conjunction with FCN, CRF post-classification processing algorithm is used. The CRF is believed to increase the accuracy of FCN by enhancing the boundaries of classified building boundaries.

3 THEORITICAL BACKGROUND

This chapter presents background concepts of CNN in detail by putting some limelight on deep learning and artificial neural networks. The first section contains the theoretical background of general network architecture and training approach. The second section deals with a brief explanation of hyper-parameters which are tuned for optimal network performance. The third section reviews the concept of transfer learning used in the deep learning method. The fourth section discusses the concept of working principle of CRFs. Finally, the last section presents a brief explanation of network architecture of pre-existing classifier which will be compared with proposed algorithm.

3.1 Deep Learning and ANN

Artificial Neural Network (ANN) and deep learning are the state of art technologies that is providing the best solution to existing machine learning problems such as image recognition, natural language processing, voice recognition, digit recognition and so on. An artificial neural network is the mathematical computer model mimicking the structure and functionality of biological nervous system whereas deep learning deals with the approach and technique of learning in neural networks [O'Shea and Nash (2015)].

Artificial Neural Networks is the system of interconnected neurons that work in distributed fashion to learn from input in order to optimize its final output [O'Shea and Nash (2015)]. The neurons in ANN are comparable to its biological analogous where neuron is the building block of a human nervous system. The functions of the neurons are to transmit the signals from receptors to the brain and vice versa. The major components of a neuron are dendrites, cell body (soma) and axon. Also in the case of ANN, the input layer on the ANN corresponds to the dendrites, whereas the axon corresponds to the output layer. In between the input and output layer are some hidden layers, which is analogous to soma.

The basic structure of the working mechanism of ANN can be illustrated in Figure 3.1. The input is loaded in form of a multidimensional vector, which are subsequently distributed to the hidden layers. Hidden layers make the decision from the previous layer and weigh up in such a way that they make finer improvement in final output. Multiple hidden layers stacked with each other is commonly referred as deep learning.

This makes computers learn complex concepts from the experiences and make it capable of doing the predictions based on such experiences. These experiences are fed to the machine as complex algorithms in the form of training data, by virtue of which the machine is able to use this knowledge for future predictions.

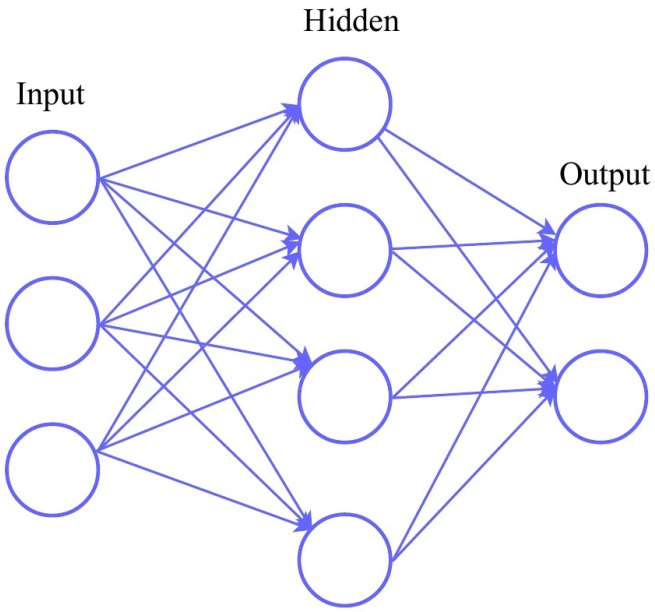


Figure 3.1: Illustration of typical Artificial Neural Network

3.2 Deep Convolutional Neural Network

CNN has been state of art technology in the field of computer vision such as character recognition [LeCun et al. (1998)], object recognition [Simonyan and Zisserman (2014)], semantic segmentation [Noh et al. (2015)] as well as object detection [He et al. (2015)]. This is like multi-layer perceptron but varies slightly to make it closer to biological mechanism of a brain [Saito et al. (2016)]. The basic idea of CNN is stacking convolution layers and pooling layer alternately [Fukushima and Miyake (1982)], as well as the use of receptive field and hierarchical structure [Hubel and Wiesel (1962)]. This forms the basis for originating the components and architecture of CNN basic architecture. One of the pioneering successful implementations of CNN goes to [LeCun et al. (1998)] for its application in hand-written digit recognition system introducing classical gradient-based optimization method, termed as back-propagation for optimizing parameters of a multilayer perceptron.

This network simply presented as a system of interconnected processing units (neurons) works in agreement to solve a specific problem such as classification, pattern recognition etc. using learning process. The CNN network possesses several advantages such as i) automatic learn local feature extractors, ii) invariant to small translation and rotation in input pattern and iii) based on the principle of weight sharing increasing generalization capability [Nogueira et al. (2017)]. The basic concept employed in CNN is presented in following subsections.

3.2.1 Basic Architecture

The basic architecture of CNN has alternatively stacked convolution layers and pooling layers followed by one or more fully connected layers or convolutional layers. The convolution layer is the main building block of the CNN. The convolution layer generally consists of two operators namely convolution and pooling. The convolution layer outputs feature map corresponding to each receptive field and weight kernels. The non-linear activation function is then applied to

these feature maps. The pooling layer then after performs down-sampling operation along spatial dimension of feature map by computing maximum on the local region. The fully connected layer or equivalent convolutional layer comes at the end after series of convolution and pooling layer which gives the class score for each pixel through the network in a feed-forward manner. The detailed explanation is presented in following section.

Convolution Layer

The convolution layers are responsible for capturing the features of the images; first layers usually get low-level features (like edge, lines, and corners) and other layers get high-level features (like structures, objects, and shapes). In this step, new images called feature maps are formed from an input image, each element of which is obtained by computing dot product between the region of interest formed by fixed size window which runs over the image, with some stride and a set of weight (called filter or kernel). This output new image (feature map) is generally smaller than the original one, containing extracted visual features [Alshehhi et al. (2017), Hu et al. (2015), Nogueira et al. (2017)].

A convolution layer takes $L \times B$ image patch with D channels centered at $x(i,j)$ and two dimensional filter kernel $l \times b$ with K number of filters as an input and output feature map of $(L-l+1) \times (B-b+1)$ spatial dimension with K channels. Each channel of this output image is called a filter site.

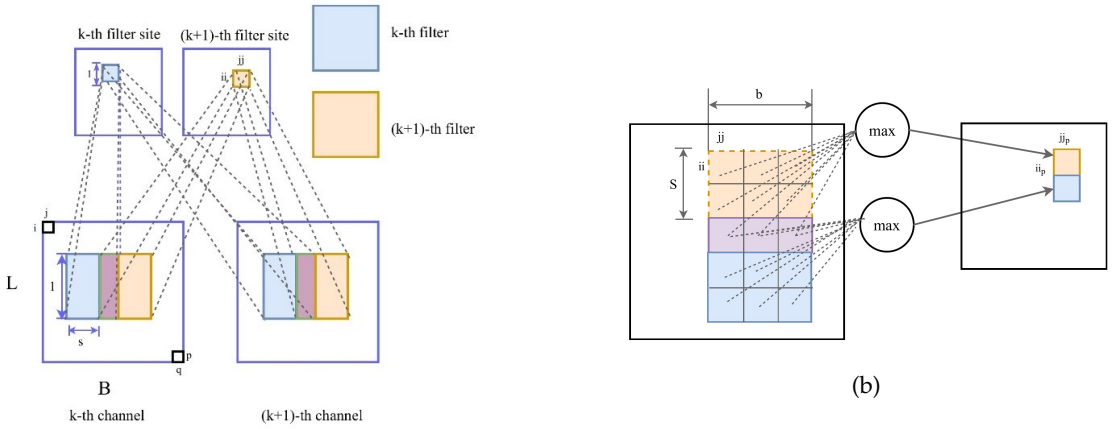
Let $x_d(i,j)$ be a pixel value at (i,j) in d^{th} channel of an input image, $x_k(ii,jj)$ be pixel value at (ii,jj) of an output feature map at k^{th} kernel size and $h_k(p,q)$ be the weight value at (p,q) at k^{th} kernel size. Then, mathematically, the convolution process is defined as,

$$x_k(i,j) = \sum_{n=1}^{D} \{ \sum_{p=0}^{l-1} \sum_{q=0}^{b-1} x_d(i.s+p, j.s+q).h_k(p,q) \} + b_k \tag{3.1}$$

where, b_k is the bias parameter of k^{th} filter shared at all location of (p,q) so that $b_k(p,q) = b_k$; and s is the stride parameter for convolution filter with an interval, generally represents distance required to slide convolution process in an input image or feature map. If $s > 1$, filters are convoluted at an interval of s horizontally and vertically, so that the size of output feature map is decreased to $((L-l)/s+1) \times (B-b)/s+1))$. The Figure 3.2a illustrates the main concept of convolution layer.

Sometimes it is necessary that input image or feature map and output feature have same spatial extent. This is where padding plays a role. Padding p_l, p_b refers to the pixels added at the outer edge of the input. For this, for every channel of the input, we can pad $[(l-1)/2]$ rows above the first row and $[l/2]$ rows below the last row, and pad $[(b-1)/2]$ columns to the left of the first column and $[b/2]$ columns to the right of the last column [Alshehhi et al. (2017), Hu et al. (2015), Nogueira et al. (2017), Saito and Aoki (2015)].

Then, element-wise nonlinear activation functions follow convolution operator. Non- linearity functions are generally used to model the activation of the specific neurons in the network and hence called activation functions [Muruganandham (2016)]. Let us assume $x_k(ii,jj)$ as an input to the activation function of neural network, w is the weight vector and b as bias vector. The activation



(a) (b)

Figure 3.2: a) An overview of convolution operator b) An overview of max-pooling operator (modified after [Saito and Aoki (2015)]

function can be expressed as:

$$Z(x_k(ii,jj)) = f\left(\sum_{k=1}^{K} x_k(ii,jj).w_k + b_k\right) \Leftrightarrow Z = f\left(X.W + b\right) \tag{3.2}$$

A variety of function are popular and being adopted for $f(.)$ to introduce non-linearity, notably, the *sigmoid* function, *Tanh*, *ReLU*, *LReLU*, and *ELU*.

Sigmoid and *Tanh* falls under the category of saturating non-linearity which tends to saturate when initialized weights are too high. Additionally, if gradient tends to zero, it might as well not exist making null or very small updates. This is called vanishing gradient problem [He et al. (2015)]. New non-saturating nonlinearities, such as *ReLU* [Nair and Hinton (2010)] has been proposed to discard this problem. A *ReLU* activates by thresholding the negative inputs to zero and passing the positive inputs unchanged as in

$$A(x_k(ii,jj)) = max(0, Z(x_k(ii,jj)) \tag{3.3}$$

where x_k is the input to the *ReLU* unit.
ReLU is proven to be computationally efficient and effective for convergence. ReLU has one drawback as it may permanently kill the neuron and never be activated if an activation is below zero. To cope with this issue, "LReLU" is proposed by [Maas et al. (2013)], allowing propagation of neuron also for deactivated neurons.

Similarly, ELU also alleviates the vanishing gradient problem via identity for positive values. In contrast to ReLU, ELU have negative values which allow them to push mean unit activations closer to zero. Mathematically, it can be shown as:

$$f(x_k) = \begin{cases} x_k, & \text{if } x_k > 0 \\ \alpha(exp(x_k) - 1), & \text{if } x_k \leq 0 \end{cases} \tag{3.4}$$

This helps to enable faster learning as they bring the gradient closer to the natural gradient [Clevert et al. (2015)].

Pooling

The pooling performs sub-sampling along the spatial dimensions of feature maps using predefined functions (e.g. maximum, average etc.) on a local region (receptive field) to summarize the signal spatially preserving discriminant information. It provides a form of robustness to the network by improving translation invariance as pooling operation operates on small windows (2×2 or 3×3) into single values. Additionally, it reduces the computational cost of the network by discarding redundant information and reducing the spatial resolution of feature map [Muruganandham (2016), Volpi and Tuia (2017)]. Moreover, this operator nullifies the effect of small translation and rotation on image or feature maps, which is very important for object detection and classification. Pooling layer is responsible for sampling the output of convolutional layer preserving the spatial location of an image, as well as selecting the most useful feature for next layer [Nogueira et al. (2017)].

The standard pooling strategies are maximum pooling and average pooling. The former returns the maximum values in the receptive field while latter returns the average of the group of activation over the receptive field. It has been observed that average pooling might not perform well as the lower value of activation can cancel the larger one. In this case, max pooling performs better as this propagate the information of absence/presence of particular features [Volpi and Tuia (2017)].

A pooling operator performs by using $l_p \times b_p$ pooling window. Let us consider that $A(x_k(ii, jj))$ is an output after applying activation function. The output $x_k(ii_p, jj_p)$ after applying maximum pooling with stride interval s_p is

$$x_k(ii_p, jj_p) = \max_{0 \leq i_p \leq l_p - 1,\ 0 \leq jj_p \leq b_p - 1} A(x_k(ii, jj)) \tag{3.5}$$

In maximum pooling, the input K channel $((L - l)/s + 1)) \times ((B - b)/s + 1))$ sized image is downscaled to the size of $((L - l)/s.s_p + 1) \times ((B - b)/s.s_p + 1))$. In max pooling layer, there is no learn-able parameter. The overview of max pooling can be demonstrated in the Figure 3.2b.

Fully Connected Layer/ Fully Convolutional Layer

In conventional CNN, the fully connected layer is usually attached at the end of the network after several convolution and pooling layers. The layers are connected to the entire input volume (from previous convolution and pooling layer) or all neuron (from the previous fully connected layer) and connect them to every single neuron in its layer. This is analogous to the way that neurons are arranged in traditional forms of ANN. Due to the reason that fully connected layer occupies most of the parameters, over-fitting is the usual case in this. To prevent this problem, dropout regularization method [Srivastava et al. (2014)] is usually applied. This technique randomly drops the neuron outputs, which do not contribute to forward pass and back-propagation anymore [Krizhevsky et al. (2012), Mnih (2013), Nogueira et al. (2017), Saito and Aoki (2015)]. Spatial information in an image is lost when the network undergoes through full connected layer as they receive activation from all the input neuron. This seriously hampers effective learning of network for semantic segmentation problem [Muruganandham (2016)].

One of the ways out is to add convolutional layer representation that is equivalent to the fully connected layer by applying convolution kernel whose dimension coincides with the previous layer. The main advantage of this is that the connection is equivalent to fully connected layer and

the output resolution is in synchronization with input images with no increase in the number of parameters.

Deconvolutional Layer

Fully convolutional layer alone is not effective for segmentation problem as it greatly reduces the spatial resolution of output to have a big receptive field for accurate classification performance. To increase the resolution of the output map, the effective way is to use so-called "deconvolutional" layer. These layers up sample the feature maps from the previous layer. One of the ways to up sample the feature is map is by performing interpolation, which is used in the thesis. The interpolation basically depends on the extent that expresses the extent of interpolation and amount of contribution that is needed from a pixel value to its neighboring positions. The effectiveness of interpolation is also depending upon the overlap in the input. The interpolation is then performed by multiplying the values of the kernels by every input and then performing addition to the overlapping to the output. The central part of the up-sampled feature is computed by adding the contribution from neighboring kernels and outer border obtained by solely the contribution from one kernel. This can be well demonstrated in Figure 3.3. For $2\times$ up-sampling, where constant 4×4 kernel is used for scaling. Moreover, interpolation kernel is also another set of learn-able parameters which is set by using bilinear interpolation [Maggiori et al. (2017)].

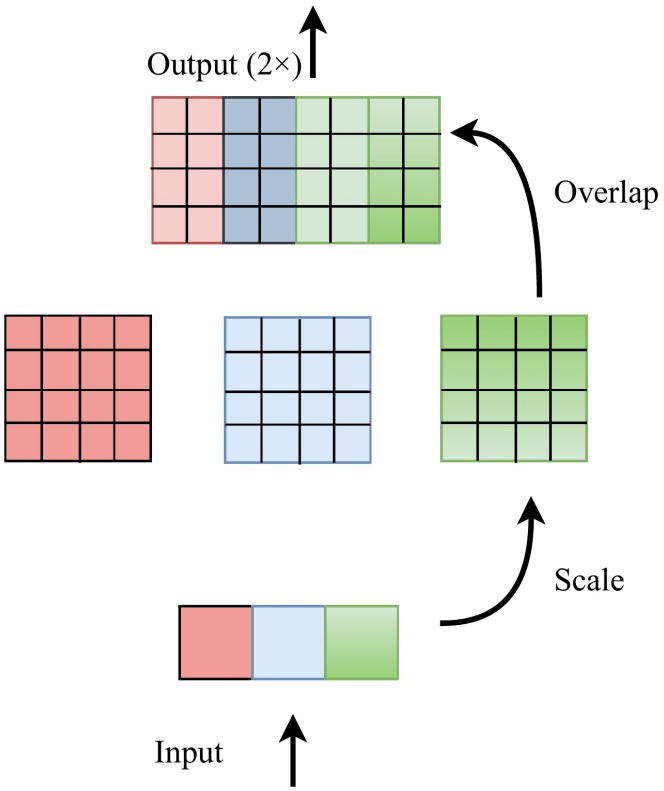


Figure 3.3: Deconvolutional layer for upsampling (modified after [Maggiori et al. (2017)])

Classifier

The output from the fully connected layers or its equivalent convolutional layer is used by classifier layer to calculate the probabilistic output of each class. The most common classifier for multi-class

classification is softmax [Lillicrap et al. (2015)] and for a binary classification problem, it reduces to being a logistic regression. The softmax function is multinomial logistic function that generates vector of real value in the range or (0,1) representing categorical probability distribution for each output class [Alshehhi et al. (2017), Krizhevsky et al. (2012), Mnih (2013), Nogueira et al. (2017), Saito and Aoki (2015), Szegedy et al. (2015)].

Let us assume $l_m \times b_m \times K$ is the form of the output of the CNN, where K is the number of channels of output image patch $m.X = [x1, x2, \ldots, xk]^T$ represent the pixel value in the output of fully connected layer and softmax function is used to convert each pixel value into class probability vector $m = [m1, m2, \ldots, mk]^T$. The equation shows how softmax function predicts the probability of the j^{th} class given the sample vector X.

$$M_{w,b}(X) = P(y = j|X; W, b) = \frac{exp(x.w_j)}{\sum_{k=1}^{K} exp(x.w_k)} \tag{3.6}$$

where W represents the weight [Alshehhi et al. (2017), Muruganandham (2016), Nogueira et al. (2017)].

Regularization

Over-fitting is a serious problem for the deep learning networks with a large number of parameters where networks are powerful enough to fit itself extremely well in the training data . It is recommended to avoid over-fitting as much as possible. For this, regularization techniques are developed.

Dropout is one of the effective and simple regularization technique used in training phase to avoid over-fitting. The term 'dropout' simply refers to dropping out units (hidden or visible) in a neural network; meaning temporarily removing neurons along with its incoming and outgoing connections. During training, random dropping principle is chosen to drop the neuron based on the probability value, where specific probability signifies neuron to be active. First introduced by Srivastava et al. (2014), it is implemented as dropout layer with probability value p. In the prediction phase, all neurons are kept active.

Another popular regularization method is L_2 regularization on which squared magnitude of all parameters are added to the loss function and the total loss is minimized as usual. The L_2 norm, also called regularization penalty $(R(W))$ is calculated as:

$$R(W) = \sum_i \sum_j w_{i,j}^2 \tag{3.7}$$

where i, j is size of weight matrix W with elements addressing as $w_{i,j}$. This is scaled by regularization strength α, and added to the loss function [CNN (2017)].

3.2.2 Training

The learning process in a CNN is sub-divided into three fundamental steps; namely forward computation, loss optimization, back-propagation and parameter updating.

Forward computation

Firstly, an input image is fed through the pre-processing stage. Then, it is fed through neural network architecture, as described in section 3.2.1, consisting series of convolution, pooling and

fully connected/convolutional layer and/or deconvolutional layer. The network returns the class label for input, governing its probability of belonging to a certain class. For semantic segmentation, the class label for each pixel is provided.

Loss optimization

The set of class probability score provided by network needs to be optimized by adjusting values of parameters such as the weight of filters and bias, that is being learned in the network. Optimization problem defines the uncertainty in determining the optimal set of parameters quantified by the loss function. For softmax classifier, the cross-entropy loss for each vector is the negative log likelihood of the training dataset N under the model.

$$L(W,b) = -\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} \times logM_{w,b}(x^{(i)})) \tag{3.8}$$

where y represents a possible class, x is data of an instance, W is the weights, i is the specific instance and N represents a total number of instances [Muruganandham (2016), Nogueira et al. (2017)].

Back-propagation

Once the loss function is defined, training of convolutional network must be done for extracting the parameters that minimize the loss. For this case, the concept of back-propagation is used which is the fundamental concept in learning. Some optimization algorithm such as stochastic gradient descent(SGD) is used to gradually update the weight and bias in search for the optimal solution. This is done computing derivative $\partial L/\partial W_{ij}$ of loss function and derivative $\partial b/\partial b_i$ of bias, with respect to weight w_{ij} (weight value between two neurons i, j in two proximal layers and bias b_i respectively, as:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial L(W,b)}{\partial W_{ij}^{(l)}} \tag{3.9}$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha\frac{\partial L(W,b)}{\partial b_{i}^{(l)}} \tag{3.10}$$

where α denotes the learning rate, a parameter that determines how much an updating, influence the current value of weight i.e. how much model learns in each step. Back-propagation algorithm obtains the partial derivatives of a cost function. This calculates the error changes as each weight is increased or decreased slightly, governed by learning rate. The algorithm then computes each error derivative by computing rate at which error changes per unit change in activity level. This error is calculated by classifier considering predicted and desired output. For other previous layers, this error is propagated between each pair of layers and error generated in the previous layer using the chain rule.

In practice, learning is performed using stochastic gradient descent (SGD) i.e. by estimating loss on a small subset of the training set, called mini-batch [Alshehhi et al. (2017), Muruganandham (2016), Nogueira et al. (2017), Volpi and Tuia (2017)].

3.2.3 Hyper-parameters

Hyper-parameters are the specific type of specific valued variable for a neural network which is set prior to the actual training process. Several kinds of methods are existing to set these values for

initializing the network. The first and basic one is manual, on which hyper-parameters are set by hand, usually using prior knowledge of the problem, and guessing the parameter values. Parameters are necessary to be modified until optimal value is obtained. The second one is called search algorithms which acts on feasible ranges for hyper-parameters available to provide all combination of parameters to train the network. The search algorithm is random in nature [Bergstra and Bengio (2012)]. The last approach is to create an automatic approach that can optimize the performance of the model according to the problem at hand. The generalization capability of the network is configured to optimize the choice of parameters chosen by the search algorithm [Snoek et al. (2012)].

To feed the network with the data in training phase, one of the following three methods is generally implemented.

- **Batch Gradient descent**: the cost function gradient is calculated by using the entire dataset.
- **Mini Batch Gradient descent**: a sub-set of training dataset (called a mini-batch) is fed into the network and update in cost function gradient is made using that dataset.
- **Stochastic gradient descent**:parameters are updated for each training samples.

Learning rate

The learning rate signifies how quickly the gradient updates to the parameter follows the gradient direction. Small learning rate causes the model to take much time to converge; whereas large learning rate cause model to diverge and loss might fluctuate indefinitely. It is usually the best way to initialize training with standardized learning rate and update by scaling with a decay factor periodically. This decay factor is also considered as one of the hyper-parameters and depends on the mini-batch size and number of iterations. The standardized input is typically set between 1 and 10^{-6}. The learning rate typically decreases with time and updating of the learning rate can be formulated as:

$$\epsilon_t = \epsilon_0 \quad \forall t < \tau \tag{3.11}$$

$$\epsilon_t = \epsilon_0 . t^{\alpha} \tag{3.12}$$

where τ and α are set up to adapt depending upon the present thresholds of the loss function.

Mini-batch size

Typically, in most of the network models, mini-batch is chosen over batch and stochastic gradient descent updating rule due to its adaptation of advantages of both other options while minimizing the limitations. This signifies that the mini-batch gradient descent is not as noisy as stochastic gradient descent and not as inefficient as batch gradient descent. Besides this, the number of images inside this mini-batch is dependent on the computational power available at hand.

Number of iteration

The most common way to set this parameter is using the principle of early stopping. Early stopping simply stops training once a performance on validation sets stops increasing. This can be a powerful tool to prevent over fitting. The evaluation on the validation set should be done infrequently; for example, every time the algorithm has seen several times newer examples than there are in the validation set.

Weight initialization

The capability of training algorithm to reach local minimum is heavily dependent on the initialization scheme of weight matrices and bias [Mishkin and Matas (2015)]. Bias is typically initialized to 0, but often weight is initialized a zero-mean Gaussian with a small standard deviation (0.1 or 0.01).

Regularization

Validation set acts a major role in setting weight decay $\alpha$and dropout probability p. The model can be evaluated on the validation set during training and optimal value can be determined. Weight decay is L_2 regularization term that penalizes big weights. The weight loss decay value determines how dominant this for gradient computation and generally added to avoid over-fitting by avoiding peaky weights. Weight decay is usually connected with loss function, so it is also called weight loss decay. The default value of weight decay lies around 10^{-3} [CNN (2014)]. Dropout is also another type of regularization. Generally, dropout is kept to the default value of 0.5, which has proven to be effective [Srivastava et al. (2014)] but playing with dropout is also a good option.

3.3 Transfer Learning

Training deep network from scratch requires a considerable amount of training data as well as lots of computational power. In many cases, there is a persistent problem of less amount of training data, therefore training a new network is a quite challenging task. So, many recent developments in machine learning/computer vision used a pre-trained network and tested using common benchmarks such as ImageNet. The use of transfer learning allows one to use pre-existing model having learned weight and fine tune the network to make it suitable for particular use [Muruganandham (2016)].

A pre-trained network can act either as fixed feature extractor for the specific task or as an initialization for fine tuning the parameters [Nogueira et al. (2017)]. Fine tuning option is the specifically good option for a new dataset which is large but not large enough to fully train the network. This is effective as it can significantly improve the performance of a final classifier. Fine tuning performs the fine adjustment of the parameters of the pre-trained network by initializing the training of the network taking current setting of the pre-trained networks. We can do fine tuning of all the layers of the convolutional network or it is also possible to freeze earlier layer and just fine tune some higher-level portion of the network. This is motivated by the observation that earlier portion contains some generic features (e.g. edge detector or color blob detectors) which is suitable for all kind of tasks, but latter layers of the convolutional network become progressively more specific to the details of the classes contained in the original dataset [Learning (2017)].

A pre-trained network can also be used as fixed feature extractor for any image since earlier feature can learn generic features which is suitable for a myriad of tasks. This is generally accomplished by removing the last layer before classification layer (usually a fully connected layer) and using the rest as feature extractor. The strategy of using a pre-trained convolutional network as feature extractor is very useful due to its simplicity as no retraining and fine tuning is required [Nogueira et al. (2017)]. Furthermore, this technique has been already used and obtained remarkable results in several kinds of image recognition tasks using deep feature trained on ImageNet [Nogueira

et al. (2017)].

Full trained network strategy is to train a network from scratch with random initialization of the filter weight. This is generally used in the case when the dataset is sufficient enough to make network converge. Many advantages of using full training network can be presented, such as i) extractor tuned for specific kind of dataset which tends to generate accurate results; ii) full control over the network. However, fully training network demands high computational power as there is a risk of over-fitting [Nogueira et al. (2017)].

3.4 Post Processing

A trade-off between localization accuracy and classification performance are inherent in DCNN, resulting in best performance and rough position of an object but not really delineating their boundaries. Coupling classification capability of DCNN with the fully connected CRFs can be one way to produce accurate semantic segmentation recovering object boundaries at the level of detail. CRF has traditionally been employed to smooth noise segmentation map, typically coupling nodes in neighborhood favoring same label assignment to spatially proximal pixels. This short-range CRF method produces quite smooth and homogeneous classification results and fails to model local smoothness. To overcome this limitation, fully connected CRFs are connected to DCNN, extending it to deep CRFs. This allows to combine single pixel prediction and shared structure through unary and pairwise terms elegantly by establishing pairwise potential on all pair of pixels in the remote sensing image. The energy function for model is

$$E(x) = \sum_i \theta_i(x_i) + \sum_{i,j} \theta_{i,j}(x_i, x_j) \tag{3.13}$$

where x represents label assignment for each pixel, $\theta_i(x_i)$ represents pixel-wise unary likelihood which is equivalent to $-logP(x_i)$ where $P(x_i)$ is label assignment probability at pixel i computed by DCNN.

Efficient interference can be achieved by establishing pairwise potential while using a fully connected graph i.e. connecting all pairs of image pixels i, j. The pairwise edge potential can be defined as a linear combination of Gaussian kernels and has a form

$$\theta_{i,j}(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{K} \omega_m k_m(f_i, f_j) \tag{3.14}$$

where μ is a label compatibility function and $k_m(f_i, f_j)$ is a Gaussian kernel, depends on the feature (defined by f) extracted from pixel i and j and weighted by parameter ω_m. $\mu(x_i, x_j) = 1$ if $x_i \neq x_j$, zero otherwise, as in Potts model, means that only nodes with distinct labels are penalized. The kernel can be further subdivided into two parts as in form:

$$k_m(f_i, f_j) = \omega_1 exp(\frac{-|p_i - p_j|^2}{2\sigma_\alpha^2} - \frac{|I_i - I_j|^2}{2\sigma_\beta^2}) - \omega_2 exp(\frac{-|p_i - p_j|^2}{2\sigma_\eta^2}) \tag{3.15}$$

where the first bilateral term is called appearance kernel and depends on pixel color intensities (I_i and I_j) and pixel position (p_i and p_j) and the second term is called smoothness kernel, only depends on pixel positions. The former terms encourage to assign a similar label to nearby pixels having similar color intensity while the latter term is responsible for removing small isolated regions. The hyper-parameters σ_α, σ_β and σ_η controls the scales of Gaussian kernels. Parameter σ_α controls the degree of nearness and σ_β of similarity [Shotton et al. (2009)]. Finally, the search

for optimal label assignment for each pixel can be done by minimizing CRF energy E(x), taking consideration of spatial correlation between them [Bittner et al. (2017), Liang-Chieh et al. (2015), Krähenbühl and Koltun (2011), Mnih (2013), Shotton et al. (2009)].

3.5 Algorithm Comparison

In this section, the CNN model adopted by [Mnih (2013)], [Saito and Aoki (2015)] and [Marcu and Leordeanu (2016)] is discussed, whose result is compared with the results of proposed algorithm (discussed in section 4.1). The result of comparison is presented in section 7.3.

Mnih (2013)

In this paper, patch-based CNN approach for single class prediction is used for building and road separately; which is a pioneering work in the field of CNN for building extraction. The CNN input is extracted from the Principal Component Analysis (PCA) to reduce a dimensionality of original image; and later, these PCA vectors are used to fine-tuning RBM to extract buildings and road network. CRF is used as post-processing technique to refine the previous output for final building layer.

The aerial imagery of Massachusetts dataset of spatial resolution 1 m is used which is divided into 64×64 RBG images patches for input. The CNN architecture network consists of three convolution layers. The first layer is with 64 filters of size 12 and stride 4, followed by max pooling of size 3 and stride 1. The second layer is with 112 filters of size 4 and stride 1; which is followed by the third layer with 112 filters of size 3 and stride 1. This convolution layer is followed by two fully connected layers with 4096 units and 256 vectors, which is subsequently reshaped into 16×16 image with one band.

The research further extended the above mentioned unstructured CNN network to structured deep CRFs. This is structured in the sense that it adds dependencies between the output which was the limitation of the previous mentioned network.

Saito et al. (2016)

The paper used single as well as multiple class prediction using CNN network to extract road network and buildings using the same dataset used by Mnih (2013). For single class prediction, they used the same CNN architecture as used by Mnih (2013). In addition to this, they used an additional maxout layer with dropout regularization at fully connected layer to increase the performance. Also, they implemented model averaging with spatial displacement (MA) technique for smoothing in outputs after semantic segmentation and CIS function to suppress the effect of the background.

Marcu and Leordeanu (2016)

The paper proposed dual stream deep network model to extract buildings using two independent pathways, one for local and another for global context and later combined in final layer processing. VGG-net [Simonyan and Zisserman (2014)] is used due to its capability to detect local and object level information due to smaller filter size while Alex-net [Krizhevsky et al. (2012)] is used as it considers information from a large area around the object of interest due to large filter size, and

later combined.

Concentrating on CNN architecture, VGG-net which is used for local context; takes $64 \times 64 \times 3$ as an input patch, considered as a local patch. The full VGG-16 architecture was used leaving final fully connected layer. This consist of 13 convolution layers in total. First two convolution layers consist of 64 filters of size 3 and stride 1 which is followed by pooling layer of size 2 and stride 2. Following this, there are two convolution layers having 128 filters of size 3 and stride 1, followed by pooling layer of size 2 and stride 2. Then after, there exist three convolution layers consisting of 256 filters of size 3 and stride 1 followed by pooling layer of the same size as before. Finally, there is two group of convolution layers having a count of three convolution layers, each having 512 filters of size 3 and stride 1 and each group followed by pooling layer of size 2 and stride 2. All these convolution layers are followed by two fully connected layers each having 4096 unit and consisting dropout layer in between them.

AlexNet in this architecture used for global context and contains all the physical architecture of original AlexNet except final fully connected layer. It takes input patch having dimension $256 \times 256 \times 3$, which is named as a global patch. It consist of four convolution layers in total with two fully connected layers. First convolution layer consists of 96 filters of size 1 and stride 4 followed by pooling layer of size 3 and stride 2. Second convolution layer consists of 256 filters of size 3 and stride 1 followed by pooling layer of size 3 and stride 2. Then after, there are consecutive three convolution layers, first two having 384 filters of size 3 and stride one and final having 256 filters of size 3 and stride 1 followed by pooling layer of size 3 and stride 2. Finally, following this, there are two fully connected layers each having 4096 units and dropout layer in between them.

These two architectures are then combined, which are composed of three fully connected layers. These fully connected layers have 8192 units, 4096 units, and 256 vectors respectively. These final 256 vectors are then subsequently reshaped to predicted labeled patch having 16×16 dimension.

4 ALGORITHM DESIGN

This chapter discusses the design decisions of the selected network architecture and training mechanism as well as the CRFs post processing algorithm.

4.1 Proposed Method

This thesis work proposes an enhanced and improved FCN network for effective and efficient building segmentation and extraction from satellite images. Two aspects of the selected FCN network are enhanced: (1) modification of FCN architecture; (2) adaptation of CRFs as post-processing. These two enhancements give rise to two variations of the FCN network, as shown in Table 4.1. For the first variation, ELU activation function is introduced in place of ReLU in original FCN. The assumption made here is that introduction of ELU can speed up learning in deep neural networks, offer higher classification accuracy and give better generalization performance than the original one [Badrinarayanan et al. (2017)]. In the second variation, the first variation is combined with post-processing CRFs algorithm to enhance the boundary of the result from the network and possibly increase the accuracy of the overall model. The explanation of the proposed and improved FCN network is presented below.

Method	Abbreviation	Description
Selected network	**FCN**	Fully convolutional network
Variation of selected network	**ELU**-FCN	FCN + **ELU activation**
Proposed Method	ELU-FCN-**CRFs**	FCN + ELU activation + **CRFs**

Table 4.1: Variation of the selected CNN models: ELU activation function and CRF

This approach has two stages, which is equivalent to classical supervised classification; namely training stage and classification stage, as shown in Figure 4.1. During training stage, the image-label pair is input into the modified FCN network (ELU-FCN) as the training samples. The modified FCN network then predicts the class label. The error between the predicted class label and the input Ground Truth Label (GTLabel) is calculated by using the designed algorithm and backpropagated through the network using the chain rule. The parameters of the modified FCN network are then updated using mini-batch gradient descent method. The mini-batch gradient descent optimization method was chosen due to its adaptation of advantage of both batch gradient descent and stochastic gradient descent, which makes it less noisy and more efficient than other. The detailed description is presented in section 3.2.3. The whole iteration will be stopped when the loss converges. For this, validation image-label pair is used. In classification stage, the final trained modified FCN network is then used to predict the rough class label of the input image. The rough class prediction, with the input image, is then inputted to the CRF post-classification processing algorithm to generate final refined binary classification output. The details of this method are presented below.

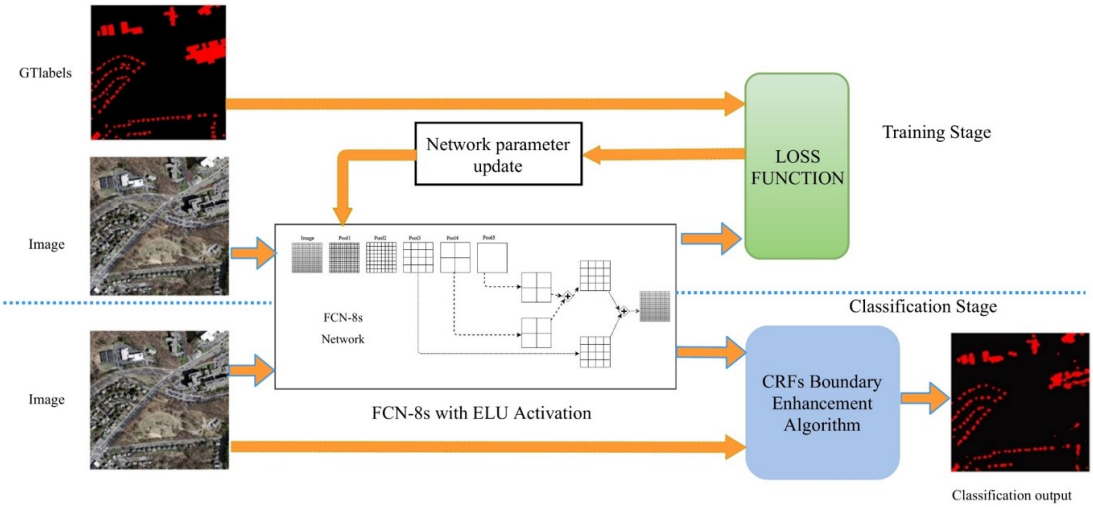


Figure 4.1: The general pipelines of the proposed approach: The training stage and the classification stage (modified after [Fu et al. (2017)])

4.1.1 Network Architecture

CNN is currently a state of art technology in visual recognition tasks such as classification and detection. One of the DCNN networks is VGG which is runner-up in ImagNet Large-Scale Visual Recognition Challenges (ILSVRC) in 2014 [Simonyan and Zisserman (2014)]. Although recently emerged deeper CNN network, such as ResNet [He et al. (2015)] and Inception-V4 [Szegedy et al. (2015)] have a lower error rate in many visual recognition tasks, VGG networks have clear structures and compact memory requirements. This allows VGG to be easily extended and applied [Fu et al. (2017)]. The advantages of using the VGG over other networks is its simplistic architecture with homogeneous 3×3 convolution kernels and 2×2 max pooling throughout the pipeline [Muruganandham (2016)]. Among other architectures of VGG model, VGG16 model (16 layered Network) is one of the strong candidates with an error rate of 8.5%. So, we chose VGG16 model as the baseline fixed feature extractor. Based on this, we constructed FCN model by replacing final 3 fully connected layers (two layers with 4096 neurons and one with 1000 neurons) by one convolutional layer. For this research, fully convolutional network i.e. FCN8s architecture [Long et al. (2015)] is used due to its high efficiency. Following the idea of [Clevert et al. (2015)], ELU activation function is used in the place of original ReLU to increase the generalized performance during training as well as the accuracy of classification.

Fully Convolutional Network

The 16 layers of the selected VGG network are divided into 5 convolution stages, grouped in a pair of 2 or 3 convolution layers, followed by 3 final fully connected layers before softmax classifier. Going through the fully connected layers, the $2-D$ structure of input images maintained by the convolution-pooling layers is lost. So, the output of standard CNN after classifier is only the $1-D$ distribution over class, which is only suitable for 'image-label' mode i.e. one label for one image. Although they have large advantages in single scene classification as presented in studies of [Hu et al. (2015)], it is not quite fruitful in case of remote sensing applications. The reason behind this is the fact that $2-D$ dense class map is required as an output for many remote sensing applications, e.g. building extraction. For handling this problem and thus to maintain $2-D$ properties of an image, the FCN model was implemented by replacing the last three fully connected layers of

original VGG with their equivalent 1×1 convolutional layers.

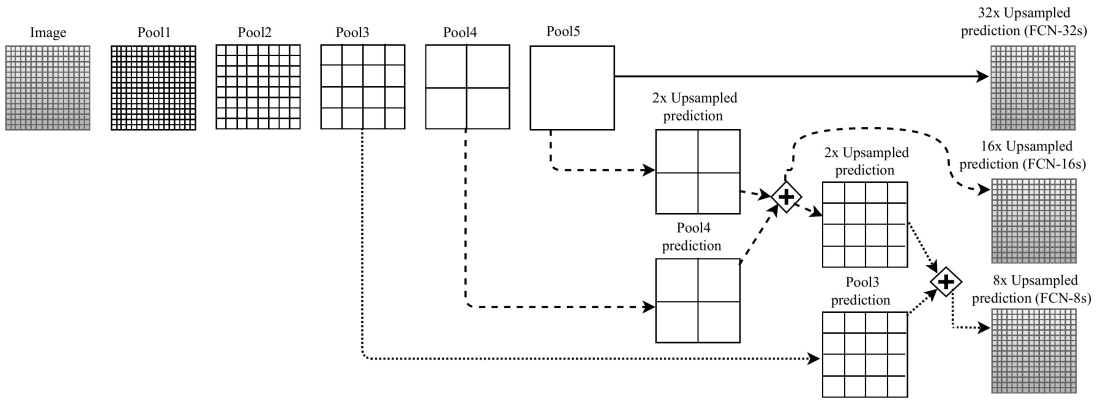


Figure 4.2: Visualization of the VGG-FCN architecture. The figure depicts skip connection architecture devised in [Long et al. (2015)]. Only pooling and prediction layer are shown, omitting intermediate convolution layer. The image shows the FCN-32s (without skip connections) on top, FCN-16s at middle and FCN-8s variant at the bottom.

Introduction of skip connections at the end of FCN resulting 3-model was achieved as described in [Long et al. (2015)]. Figure 4.2 shows the $3-$model achieved by using skip connections are FCN-8s with skip connections from the pool3 and pool4 layers, FCN-16s with skip connections from pool4 alone, and FCN-32s without the use of any skip connections. The reason behind using skip connection is to aggregate the feature learned at low or medium layers with the higher layers (shown right in figure 6) and to help classifier to predict from aggregated features. Another major advantage of this approach is the preservation of the spatial information. The fully connected layers pairwise connect each neuron with every single neuron of its preceding layer; so spatial information is lost. In contrast, convolutional layers only connect to the neurons in its effective receptive field in a deep network [Muruganandham (2016)]. Moreover, the parameter estimated for final fully connected layers are 140 million, which will be subsequently discarded simply by using features at the higher intermediate pooling layers. For this, additional convolutions are applied for each of the pool5, pool4, and pool3 features before feeding them into the classifier. Here, a deconvolutional layer is used to resize final score predicted in small feature space into input image size, which allows the FCN architecture to take in images of any input size. More importantly, the chosen designed model (FCN-8s) of FCN architecture, the feature maps at all three stages (layer 7, layer 10 and layer 13) from VGG network are used making it more robust.

Thus, we adopted the FCN-8s model for building extraction using aerial imagery. The output number (channels) of last convolutional layer is set equal to the number of class required (for this research, it is 2 for building classification). The feature maps are a heat map of corresponding classes, which are subsequently up-sampled to match with original image size.

4.1.2 Network Training

The general procedure of the approach adopted for training the network is demonstrated in Figure 4.3. All training image-GTLabel pair (see in Table 5.2) are input into the modified FCN classification network as training samples. The softmax function is used to predict the class distribution in categorical output utilizing the output feature map generated by final convolutional

layer. The theoretical background on how this function works is well explained in section 3.2.2.

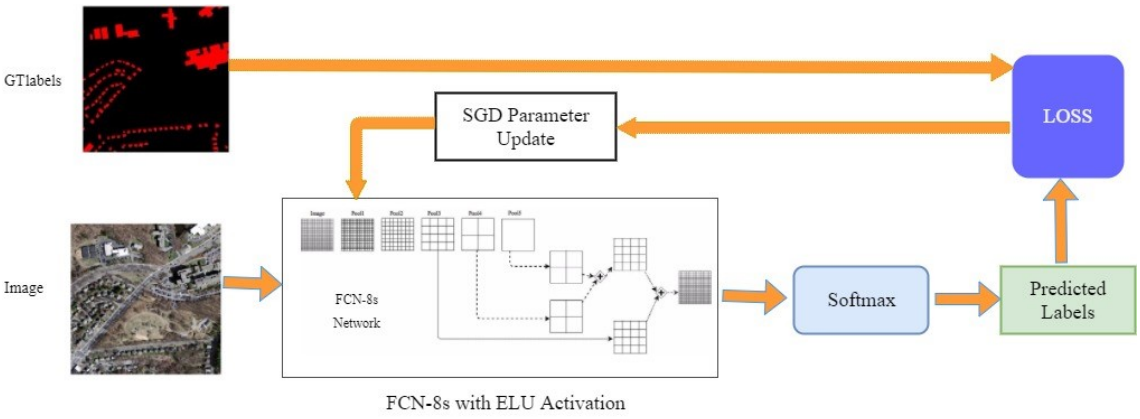


Figure 4.3: General pipeline of network training (modified after [Fu et al. (2017)].

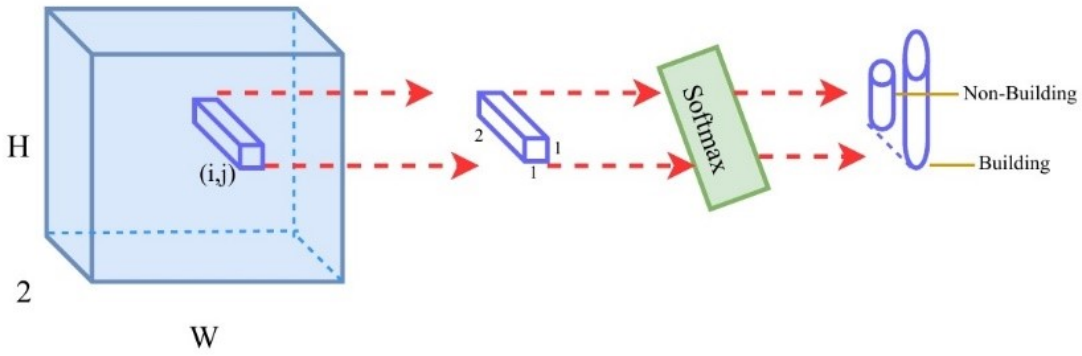


Figure 4.4: Softmax function performed after output feature map at end of CNN (modified after [Fu et al. (2017)].

The output of our modified FCN network is $L \times B \times K$ feature map, ($K = 2$ in our case) having the same dimension as an original image. Here L and B represent dimension (length and breadth) of an output of final convolution layer of modified FCN network and K represent output dimension. The square hole at location (i, j) represents feature vector with 2 elements, corresponding to the same pixel in an original image. The softmax function converts this feature vector into the 2-D probabilistic vector. This vector is the discrete distribution of probability of that pixel at location (i, j) falling into respective classes. The softmax does the same kind of experiment to the entire image to produce dense classification [Fu et al. (2017)]. The procedure is well demonstrated in Figure 4.4.

The result from the comparison between GTLabels and predicted label after applying softmax function is used to calculate cross entropy loss. This loss value is then back-propagated to update the parameters of network using mini-batch gradient descent. The theoretical background on how this procedure works is well explained in section 3.2.2.

4.1.3 Post Classification Processing Using the Trained Network

The modified FCN network involves up-sampling operations which result in blurring the classification boundaries. A trade-off between localization accuracy and classification performance are inherent in DCNN, resulting in best performance and rough position of an object but not really delineating their boundaries. The same condition applies to the modified FCN network explored in the thesis. Coupling classification capability of modified FCN network with the fully connected CRF can be one way to produce accurate classification results recovering object boundaries at the level of detail. Several works [Bittner et al. (2017), Liang-Chieh et al. (2015), Krähenbühl and Koltun (2011), Mnih (2013), Shotton et al. (2009)] used CRFs as post-processing to refine the image segmentation results. So, following their idea, we adopted the fully connected CRFs to refine our rough class prediction. The theoretical explanation on how CRF works is well explained in section 3.4.

The complete procedure of building classification using trained and modified FCN network with

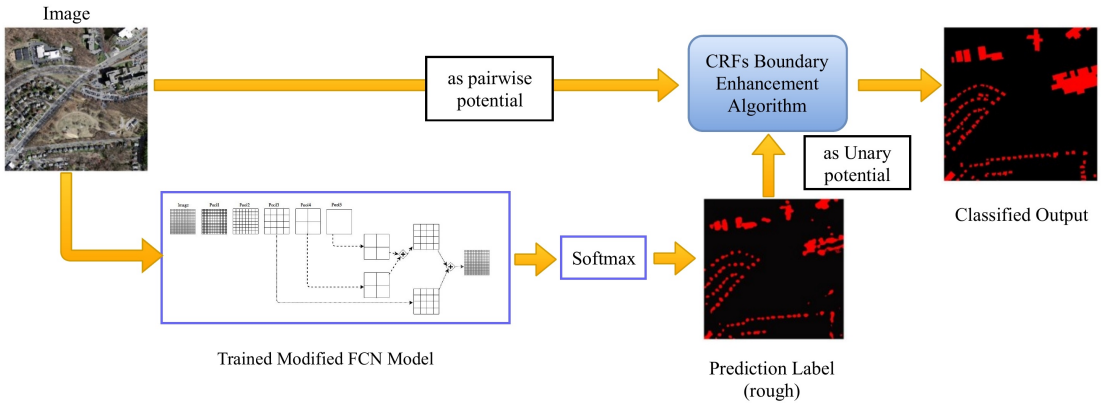


Figure 4.5: General methodology of building classification using trained network in conjunction with CRF post processing algorithm (modified after [Fu et al. (2017)].

CRFs post processing is presented in Figure 4.5. Here, the rough class distribution predicted by the modified FCN network is input as the unary potential giving position information only, and the original image act as pairwise potential giving both position and color information.

5 DATASETS AND EXPERIMENTAL DESIGN

This chapter discusses the dataset considered for the implementation of designed algorithms, as well as the experiments to determine the optimal value of hyper-parameters and parameters for CRF algorithms. The first section describes the dataset adopted for the thesis, and it is followed by the section which describes data preparation and preprocessing. Finally, the last section presents the experimental design for performing sensitivity analysis experiments to determine the optimal value of hyper-parameters of CNN and parameters for CRFs algorithms.

5.1 Data Description

In this thesis, open Massachusetts building dataset prepared by Mnih (2013) was used. The dataset possessed 151 images of the state of Massachusetts. Each image was 1500×1500 pixels RGB images, at the spatial resolution of $1m$, covering an area of 2.25 square kilometers. Target maps used for the images were prepared by using open sourced OSM and were also made readily available, in rasterized format. The sample of this dataset is shown in Figure 5.1.

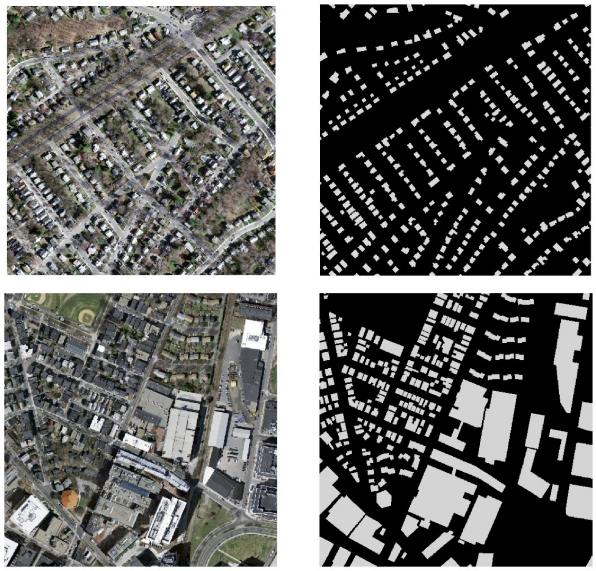


Figure 5.1: Two sample aerial images from Massachusetts building dataset; row refers to each image (a) aerial image, (b) building mask, which acts as a ground truth of the corresponding image.

The original images were arbitrarily split into training, validation, and test datasets as shown in

Table 5.1. A set of images with GTLabels pair (Training set) is used to train the model. The performance of the model trained by training set is then evaluated using the test set, which also composes a set of images and GTLabels pair. A validation set is used to validate the best possible model which is obtained by using sensitivity experiments. Generally, this is done by using the validation set to tune the parameter of the model during training and periodically evaluating in the validation set.

Training	Validation	Testing
137	4	10

Table 5.1: An overview of the Massachusetts building datasets used, which randomly split into training, validation and test datasets.

This dataset is considered as one of the challenging building datasets to predict using any networks [Marcu and Leordeanu (2016)]. The whole dataset contained a wide range of urban, suburban, and rural regions with a total area of 339.75 square kilometers. Quick qualitative review over the randomized subset of dataset showed that there are some buildings under the cover of a tree blocking out the buildings from vantage points of the viewer. Moreover, some buildings are also under the shadow of big buildings. Additionally, some building patches are missing in GTLabels where there exists building in images. This collective nature of images hampered in prediction, which is discussed in detail in the section 7.2.2.

5.2 Data Preprocessing

Data preprocessing (also called data preparation) is imperative when working with deep learning models. The main purpose of data preprocessing is to transform the datasets so that the information contains with in the datasets is best exposed to the network. The original labels contained in original Massachusetts building dataset were three channeled RGB GTLabel images having two classes (255 for building and 0 for non-building) throughout the dataset. It is imperative to convert this image into black and white image with hard binary level of two class for the network adopted for the thesis as it demands the label to be of one channel with hard binary label. This is achieved by first converting the original GTLabel into greyscale value and then into the hard-binary label (1 for building and 0 for non-building).

Various data augmentation techniques were also employed in the thesis work. Firstly, original datasets were cropped systematically into smaller dimension to increase dataset as well as to adapt computational power of workstation used in the thesis. The 1500×1500 sized images and labels were cropped into non-overlapping segments of size 500×500 across the train, test and validation splits. The new number of training, validation and test datasets are shown in Table 5.2.

Training	Validation	Testing
1233	36	90

Table 5.2: New set of datasets after cropping; containing training, validation, and test datasets.

The datasets presented in Table 5.2 were augmented by introducing random rotation and adding noise randomly over the image. The reason behind rotating each pair of image and label by a random angle during training was not to favor object in any orientation in the resulting models

which helps in making the model robust. The dataset obtained thus was fed through preprocessing pipeline to normalize the image. The main purpose behind normalization of images before feeding it into ANN model is to convert all the distribution of pixel value in each channel into a same specific range. The thesis work involved simple mean centering of the training set as a normalization technique where R-G-B image mean value obtained from ImageNet during VGG were subtracted from across the entire training set. This serves to center the data.This also guide the network toward guarantee stable convergence of weight and biases. The standard z-score normalization of the training set was not applied because all channel has a range from 0-255 and there is no effect of z-score normalization on the result.

5.3 Experimental Design

The experiments carried out in this research is mostly built on top of deep learning framework "Tensorflow" [Tensorflow (2017)], in python environment. Tensorflow is an open sourced deep learning framework developed at Google, allowing a user to implement various algorithms quickly and efficiently. This is fundamental for implementing neural network algorithm. Due to its wide range of ready-to-use functions as well as the community support for this framework, Tensorflow was chosen over other well-known frameworks (i.e. caffe, torch, theano).

This section presents the framework design setup for experimenting the sensitivity of hyper-parameters on the designed CNN network. Each set of each hyper-parameters i.e. learning rate, batch size, number of iterations, weight loss decay and dropout were used for each experiment and noted and analyzed for determining optimal value of each hyper-parameters. Same procedure was followed to determine optimal value of parameter of CRF post processing algorithm during parameter tuning experiments. The detailed description of these steps is explained in the section 5.3.1 and 5.3.2. The results of these experiments will be presented in chapter 7.

5.3.1 Sensitivity to Hyper-parameters

Hyper-parameters are the specific "higher-level" properties of model which can't be directly learned from the regular training process and should be fixed prior to the training process. The optimal value of hyper-parameters help CNN network to discover the parameters of the model that result in robust prediction. The architecture described in section 4.1 was used and similar configuration of CNN was used in all setting. Each sensitivity analysis experiment was designed with training and validation set built. The F1-score and computational time per iteration were considered as criteria for evaluating sensitivity analysis experiments.

The hyper-parameters investigated in these experiments were the learning rate, batch size, number of iterations, weight loss decay and dropout regularization. A small set of each hyper-parameter was used for the experiments to minimize the computational cost. Previous similar studies and theoretical background behind each hyper-parameters were the basis behind creating each set of hyper-parameter for sensitivity experiments. A constant value of other hyper-parameters were set while changing one hyper-parameter so that we can determine the hyper-parameter that our CNN is most sensitive to, based on a comparison of percentage increase/decrease in F1-score and computational time.

Effect of Learning Rate

The pace/speed with which the neural network learns new things from a data is called learning rate. The detailed explanation is presented in section 3.2.3. The effect of the learning rate (L) hyper-parameter by varying it to a CNN with a fixed configuration of other parameters and hyper-parameters was investigated, as shown in Table 5.3. Training of the network was done with mini-batch gradient descent over all 1233 training samples, tuned and evaluated with all 36 validation samples. The relaxed F1-score (described in chapter 6) was used as a performance metric to evaluation. The range of L value used in the experiments are $10^{-3}, 5*10^{-4}, 10^{-4}, 5*10^{-5}, 10^{-5}$.

Hyper-parameters	Value
Learning rate (L)	$[10^{-3}, 5*10^{-4}, 10^{-4}, 5*10^{-5}, 10^{-5}]]$
Batch Size (n)	3
Number of iterations (N)	50000
Weight decay loss (W)	10^{-5}
Dropout (D)	0.5

Table 5.3: CNN sensitivity experiments on the effect of the learning rate hyper-parameters used.

Effect of Batch Size

Batch size defines the total number of training example present in a single batch that going to be propagated through the network. The detailed explanation is presented in section 3.2.3. The effect of the batch size (n) hyper-parameter by varying the size of training data input in each iteration for training the CNN was investigated. We trained the model using mini-batch gradient descent with all the arrangement done in learning rate experiment except the value of learning rate is fixed for the value shown in Table 5.4. The value of n value used in the experiments are 2, 3, 4 and 5.

Hyper-parameters	Value
Learning rate (L)	10^{-5}
Batch Size (n)	[2,3,4, 5]
Number of iterations (N)	50000
Weight decay loss (W)	10^{-5}
Dropout (D)	0.5

Table 5.4: CNN sensitivity experiments on the effect of the batch size hyper-parameters used.

Effect of Number of Iterations

Number of iterations signifies here is the number of batches before stopping the training process. The detailed explanation is presented in section 3.2.3. The effect of the number of iterations (N) used in the model was also investigated. The network with the fixed configuration in Table 5.5 was trained using mini-batch gradient descent. The values used for N are $20000, 30000, 40000, 50000, 60000$.

Effect of Weight Loss Decay

Weight loss decay is a regularization term which prevents the weights from growing too large. The detailed explanation is presented in section 3.2.3. The effect of the weight loss decay (W) of CNN network with the configuration presented in Table 5.6, trained using mini-batch gradient descent, was studied. The values of W tested are $10^{-3}, 10^{-4}, 10^{-5}$.

Hyper-parameters	Value
Learning rate (L)	10^{-5}
Batch Size (n)	3
Number of iterations (N)	[20000, 30000, 40000, 50000, 60000]
Weight decay loss (W)	10^{-5}
Dropout (D)	0.5

Table 5.5: CNN sensitivity experiments on the effect of the number of iteration used.

Hyper-parameters	Value
Learning rate (L)	10^{-5}
Batch Size (n)	3
Number of iterations (N)	50000
Weight decay loss (W)	$[10^{-3}, 10^{-4}, 10^{-5}]$
Dropout (D)	0.5

Table 5.6: CNN sensitivity experiments on the effect of the weight loss decay hyperparameters used.

Effect of Dropout Regularization

Dropout is one of the regularization technique which helps in avoiding over-fitting by temporarily removing neurons. The detailed explanation is presented in section 3.2.3. The effect of the dropout (D) used by CNN network was also studied. The network with the fixed configuration in Table 5.7 was trained with mini-batch gradient descent. The values of D tested are 0.25, 0.5 and 0.75.

Hyper-parameters	Value
Learning rate (L)	10^{-5}
Batch Size (n)	3
Number of iterations (N)	50000
Weight decay loss (W)	10^{-5}
Dropout (D)	[0.25, 0.5, 0.75]

Table 5.7: CNN sensitivity experiments on the effect of the dropout hyperparameter used.

5.3.2 Parameter Tuning Experiments for CRF Post Processing Algorithms

In this section, we discuss the experiments that were conducted to determine the optimal values for parameters of CRF post processing algorithm. The relaxed F1-score in validation set were evaluated for each pair of parameter was made a basis for the optimal value of the parameter.

Hyper-parameters	Value
Appearance kernel – nearness parameter σ_α	[2, 3, 5, 10, 20]
Appearance kernel- similarity parameter σ_β	[5,10, 15, 20, 30]
Smoothing kernel – nearness parameter σ_η	[2, 3, 5, 10, 20]
No of iterations	[5,10, 15, 20, 30]

Table 5.8: Parameter tuning experiments for CRF post processing: parameter values.

The parameters investigated in these experiments were Gaussian kernels (σ_α and σ_β as a component of appearance kernel, and smoothness kernel σ_η as a component of smoothness kernel) and total number of iterations. The detailed explanation is presented in section 3.4. The additional parameters called weight parameters ($w(1) and w(2)$) were set to 1, as defined by Krähenbühl and Koltun (2011).

A small set of each parameter was used for the experiments to minimize the computational cost, as shown in Table 5.8. Previous similar studies and theoretical background behind each parameters were the basis behind creating each set of parameter for sensitivity experiments. A constant value of other parameters were set while changing one parameter so that we can determine the parameter that our CNN is most sensitive to, based on a comparison of percentage increase/decrease in F1-score.

6 PERFORMANCE COMPARISON

This chapter explains the approaches that were used for comparison of the proposed and pre-existing classification algorithms. The first section presents final configuration of designed CNN classifier based on the knowledge gained in the experiments described in chapter 4. The second section discusses the quantitative performance metrics for comparison of different classifiers. Finally, the last two section presents the qualitative approaches for comparison.

6.1 Classifiers

This section presents the classifiers that participate in the performance comparison. The first three classifiers are the variations of proposed classifier and were compared to validate the significance of each add strategies on chosen FCN classifier towards increase in performance. The last classifiers are set of pre-existing classifiers which was implemented and tested using same Massachusetts building datasets. The results from proposed classifier was compared with the result from pre-existing classifiers to test the performance.

6.1.1 FCN

Network architecture CNN presented in section 4.1 that utilize fully convolutional layer instead of fully connected layers at the end of VGG-network was chosen to learn the classification rule. The overall implementation process for adopted CNN network is demonstrated in Figure 6.1. As a part of adopted CNN network, VGG-16 network pre-trained on ImageNet was adopted as a feature extractor freezing 13 convolutional layers. This is illustrated in Figure 6.1 using red dotted box. This transfer learning method as a feature extractor was employed to cope up over-fitting issue due to less availability of training data and for fast convergence of training. Apart from VGG-network, the constant 7×7 receptive field was used in the fully convolutional layer at the end of the chosen network. This was devised by [Long et al. (2015)] and used in the research keeping in mind that average dimension of buildings matches with the same dimension. The receptive field with lesser dimension was not used as it takes a longer time to train the network. The weight was initialized randomly in each layer (apart from the frozen VGG-13 layer) with zero-mean Gaussian distribution with standard deviation of 0.01. The theoretical basis towards using this value is described in section 3.2.3. For optimization of parameters during training of network, Adam optimization algorithm based on mini-batch gradient descent was applied. The theoretical background behind optimization is presented and discussed in section 3.2.2. Adam optimization was adopted due to its high performance in practice in compared to other optimization algorithms such as RMSprop, Adadelta, AdaGrad etc. [Ruder (2016)]. Moreover, adaptation of learning rate during training is automatically synchronized with iterations in this algorithm. The optimal values of hyper-parameters (Table 6.1) showing good results in the sensitivity analysis experiments (see subsections 5.3.1) for the CNN were adopted. The results of sensitivity analysis experiments are

discussed in section 7.1.1.

Hyper-parameters	Value
Learning rate	$5 * 10^{-5}$
Batch size	5
Number of iterations	30000-40000
Weight loss decay	$1 * 10^{-5}$
Dropout	0.5

Table 6.1: CNN hyper-parameters: - Learning and regularization parameters..

6.1.2 FCN with ELU Activation Function

To improve the performance of original FCN architecture, ELU activation function was adopted in place of ReLU. Thus, the results from FCN classifier, described in sub-section 6.1.1, were compared with FCN with ELU strategy. The basis for comparison is presented in section 6.2, 6.3 and 6.4. For a fair comparison, the architectural details of the CNN and hyper-parameter value, described in sub-section 6.1.1, were preserved.

6.1.3 FCN+ ELU with CRF Post Processing

The CRF post processing algorithm was implemented for enhancing coarse classification result after the trained FCN+ELU network. This is used at the end of chosen CNN classifier for smoothing the boundary and further remove the noise presented in coarse classification results. Thus, the results of the CNN classifier described in sub-section 6.1.1 and 6.1.2 were compared with a modified CNN classifier that uses CRF post processing. The basis for comparison is presented in section 6.2, 6.3 and 6.4. For a fair comparison, the architectural details of the CNN and hyper-parameter values, described in subsection 6.1.2, were preserved. However, in contrast to the CNN above where output is the final class labels, the class probability maps produced by softmax classifier was fed as inputs to the CRF for final binary labeling. The optimal parameter values for CRF obtained from parameter tuning experiments in section 5.3.2 was adopted, which is presented in Table 6.1. The results of parameter tuning are discussed in section 7.1.2.

Parameters	Value
Appearance kernel – nearness parameter σ_α	3
Appearance kernel- similarity parameter σ_β	10
Smoothing kernel – nearness parameter σ_η	3
No of iterations	15

Table 6.2: Parameters for CRF post processing.

6.1.4 Pre-existing CNN classifier

The results from pre-existing CNN classifiers, namely Mnih (2013), Saito et al. (2016) and Marcu and Leordeanu (2016), discussed in section 3.5, were used to compare the result from the proposed classifier.

6.2 Performance Metrices for Comparison

Various performance measures were used to compare the adopted classification approaches described in section 3.5 and 4.1. All the experiments conducted in the thesis were based on the

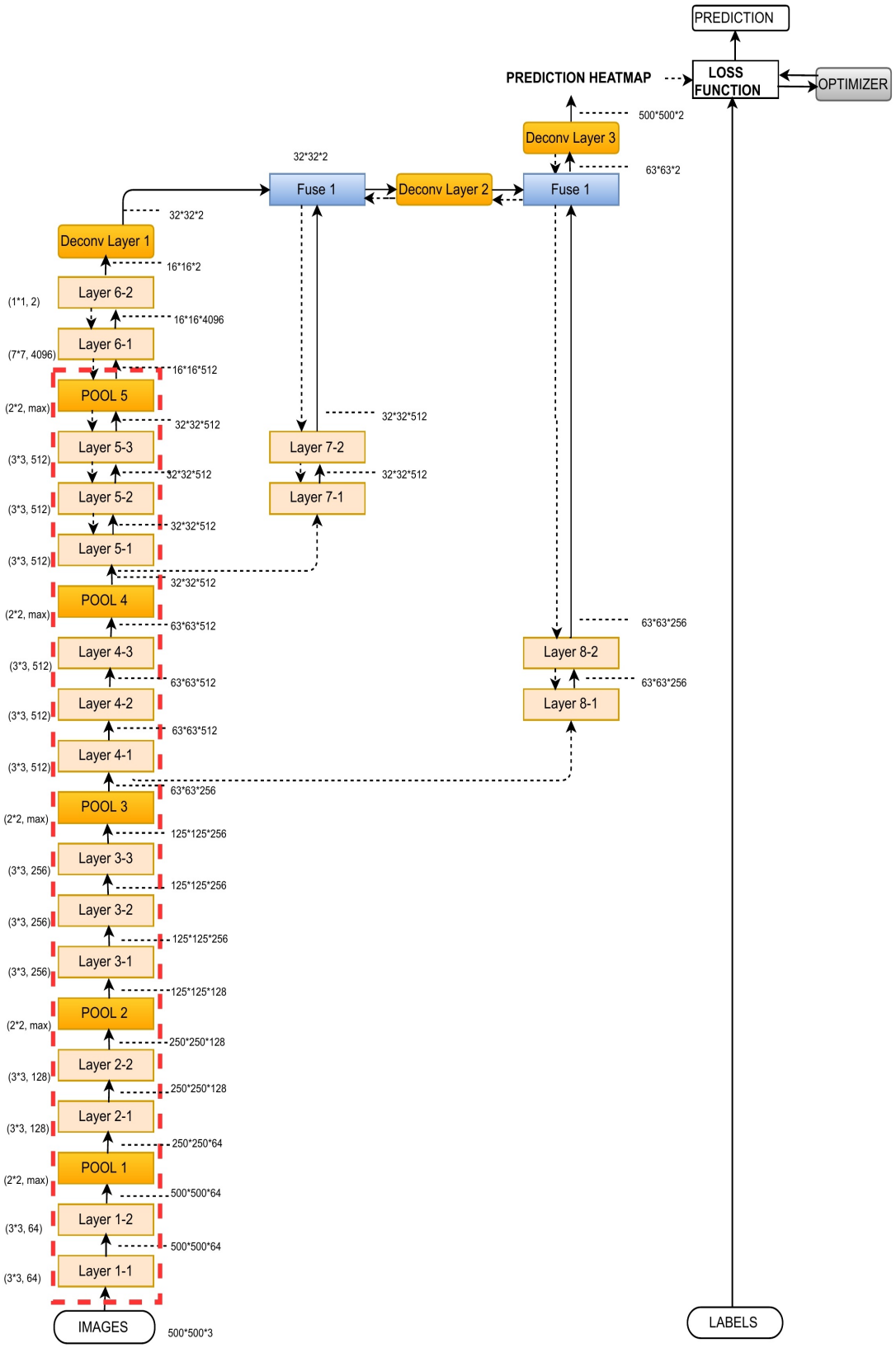


Figure 6.1: Fully convolutional architecture adopted in the research. Red dotted layers indicate frozen layer (weight taken from pre-trained VGG layer) where training is not considered.

several performance metrics such as precision, recall, F1-score, and intersection over union (IoU) measure. For this, the classifiers were evaluated with the test set as opposed to the true ground truth and the resulting performance metrics for each of the classifiers recorded and evaluated. All performance metrics considered in the research are based on four classification output, as shown in confusion matrix in Table 6.2. This is because building extraction is considered as the binary problem where building pixels are positive and the remaining non-building pixels are negative. The four classification outputs are termed as True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). TP denotes the number of target pixels (building pixels) correctly classified; TN denotes the number of non-target pixels (background pixels) correctly classified; FP is the number of non-target pixels classified as targets; and FN is the number of target pixels classified as non-target.

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

Table 6.3: Description of true positive, false positive, false negative and true negative.

All performance metrics considered for the research are explained as follows. The relationship of all performance metrics with four classification outputs are also presented in form of equation.

Precision and Recall

Precision (also known as correctness) refers to the ratio of a correctly classified positive pixel (buildings) to all predicted positive pixel (building) by the classifier; whereas recall (also known as completeness) is the proportion of correctly classified positive pixel among all true target pixels [Badrinarayanan et al. (2017)]. These are defined as follows.

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

Precision and recall measure were chosen as the primary metrics because high-class imbalance was present in the data. This is because the buildings are generally sparse compared to the background in the study area.

F1-score and IoU measure

Best performing models on each dataset are assessed mainly with the F1-score and intersection over union (IoU) measure.

F1-score is the weighted average of precision and recall. This takes both false positive and false negative into account. This is quite fruitful if you have uneven distribution. It is also considered as good measure of performance as it considers both precision and recall. F-score is derived from the precision and recall values, for pixel-based evaluation [Muruganandham (2016)].

$$F_{measure} = \frac{(1 + \beta^2)TP}{(1 + \beta^2)TP + \beta^2 FN + FP} \tag{6.3}$$

For simplicity $\beta = 1$, termed as F1-score which can be revised in term of precision and recall as

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{6.4}$$

Another useful metrics is IoU measure which is an average value of the intersection of the prediction and ground truth regions over the union of them. In our case, IoU measure is defined as the number of pixels labeled as building on both in ground truth and predicted footprint, divided by the total number of pixel labeled as buildings in each of them [Muruganandham (2016)].

$$IoU = \frac{TP}{n_{pred} + n_{gt}} \tag{6.5}$$

Where n_{pred} is the number of pixels labeled as building in predicted footprint and n_{gt} is the one in ground truth.

This can be interpreted in term of precision and recall as

$$IoU = \frac{Precision * Recall}{Precision + Recall - Precision * Recall} \tag{6.6}$$

Although F-score and IoU measure are both positively correlated, both measure were used for performance metrics. The reason behind this is that IoU tends to penalise single instance of bad classification more than f-score quantitatively even when both agree that this one instance is bad. This is more intitutive to use IoU when taking average of these score over set of inferences. Similarly, IoU have a square effect on the errors relative to the F-score [stackexchange (2017)].

Modification of Performance Metrics

[Mnih (2013)] stated that the raster label was generated by using vector map in OSM platform and the generated pixel labels are accurate only up to a few pixels. So, to compare with the results reported by the pre-existing classifier (see section 3.5), same metric was used in the thesis work to evaluate the results. Averaged relaxed precision and recall scores (after this relaxed precision and relaxed recall) instead of exact ones, were proposed for evaluation. The relaxed precision is defined as the fraction of predicted building pixels that are within ρ pixels of a true building pixels, while relaxed recall is defined as the fraction of true building pixels that are within ρ pixels of a predicted building pixels [Saito and Aoki (2015)]. This is common practice to evaluate this kind of prediction which is also used in [Wiedemann et al. (1998)]. For all experiments discussed in this research, slack parameter ρ was set to 3 pixels that is the same value used in [Alshehhi et al. (2017), Marcu and Leordeanu (2016), Mnih (2013) , Saito et al. (2016)]. For the given value of slack parameter, a positively classified pixel is considered as correct if it is within same value of pixels from any positive pixel in the ground truth, This is realistic approach, as the border of buildings in ground truth generally are some pixel off due to generation procedure [Saito et al. (2016)].

Similarly, relaxed F1-score and relaxed IoU measure were also computed using relaxed precision and recall for synchronization. For this calculation, relaxed precision and relaxed recall at the break even point are considered. Breakeven point considered here is that point where there is the minimum difference between precision and recall value for all classifiers.

6.3 Visual Inspection and Comparison

The research also compared the classification of building maps predicted by the proposed variation of the classification method as a part of evaluation. Representative parts of the study area were

chosen with different complexity, structure and count of buildings; namely urban area, rural area, and shoreline area. The classified maps were inspected for occurrence of buildings in predicted map and their shape, as compared to their ground truth label for comparison.

6.4 Computational Time and Complexity

Computational time and complexity was also considered for the comparison of classification approach. This metric considers the time taken by each of the classifiers during training. Moreover, network complexity in term of parameters to be learned during training was also taken into consideration for comparison.

7 RESULTS AND DISCUSSION

This chapter reports the findings of the experiments and performance comparison as described in chapter 5 and 6 respectively. The first section reports the sensitivity analysis experiments performed with the CNN algorithm. The immediate next sub-section discusses the results of the parameter tuning experiments for CRF post processing algorithms. The succeeding section presents the finding of the analysis done for comparison of the variations of the proposed method. Finally, the last section presents the comparison of the proposed method with the pre-existing classifiers.

7.1 CNN Design Experiments

7.1.1 CNN Sensitivity Analysis

This section presents the results of the sensitivity analysis performed with the CNN network described in section 4.1. The relaxed F1-score of the network on validation dataset and computational time of training were recorded and analyzed by varying hyper-parameters.

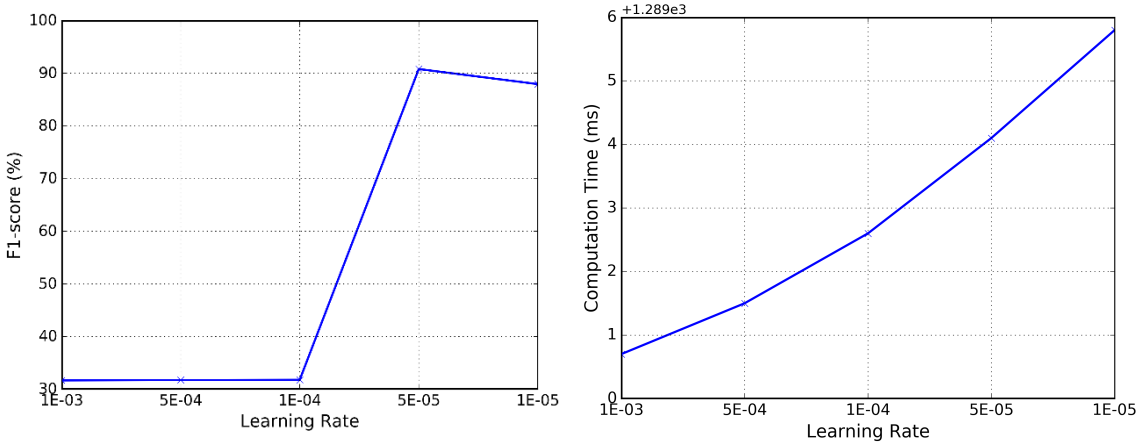
Effect of Learning Rate

Figure 7.1a illustrates how the classification performance of proposed CNN behaves under different learning rate. With a decrease in learning rate from $1 * 10^{-3}$, there is an increasing trend in F1-score from 31.69% peaking to 90.82% for learning rate of $5 * 10^{-5}$. After that, there is a slight decrease in F1-score, approximately of 2%, with the value of learning rate of $1 * 10^{-5}$. The computational time of training per each iteration goes on increasing from 1289ms to 1294ms with a decrease in learning rate, as demonstrated in Figure 7.1b.

The learning rate plays an important role in the convergence of the network for efficiently and effectively training a model and produce accurate results. The exponential function used for loss can make the network to diverge when higher learning rate is adopted. The improved performance of the CNN network using lower value of learning rate comes in expense of higher computational time. It is sometimes impossible to train the network due to the added computational complexity.

For the learning rate in the range between 10^{-3} to 10^{-4}, there is very low performance (F1-score), which can be justified by higher learning rate. The reason behind the constant accuracy at these values can be explained by observing the loss graph for those learning rate in Figure 7.2. At those learning rate, the loss graph converges at higher loss value than optimal one. It is observable that, at the higher learning rate, the network diverged and lost in the abyss of local minimum and never got chance to approach towards the global minimum.

The learning rate of value $5 * 10^{-5}$ was chosen for our analysis because at this rate the optimal F1-score is obtained which balanced complexity and performance. Below that value of learning



(a) (b)

Figure 7.1: Effect of varying learning rate (a) on F1-score, (b) on computational time

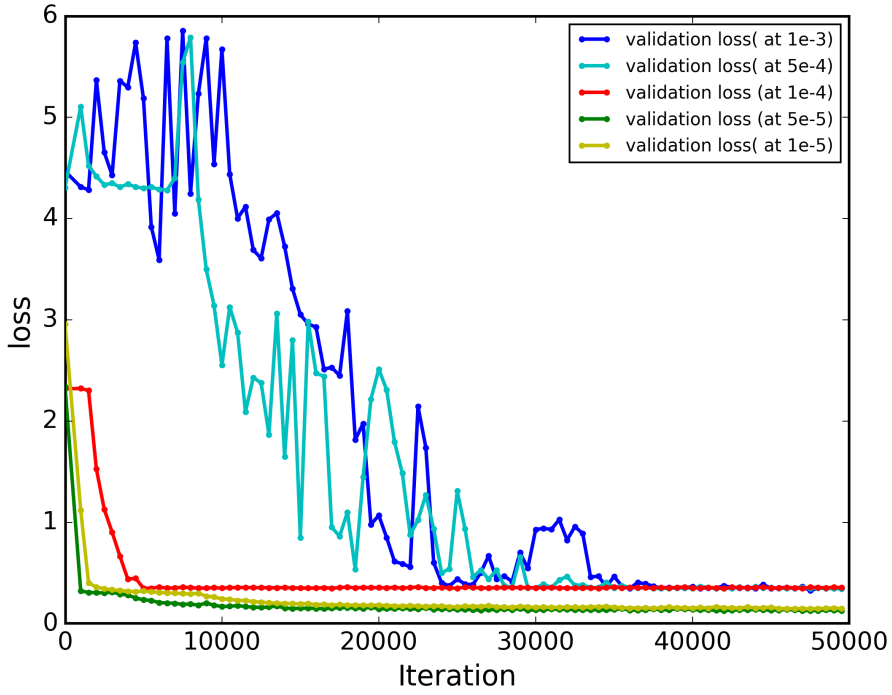
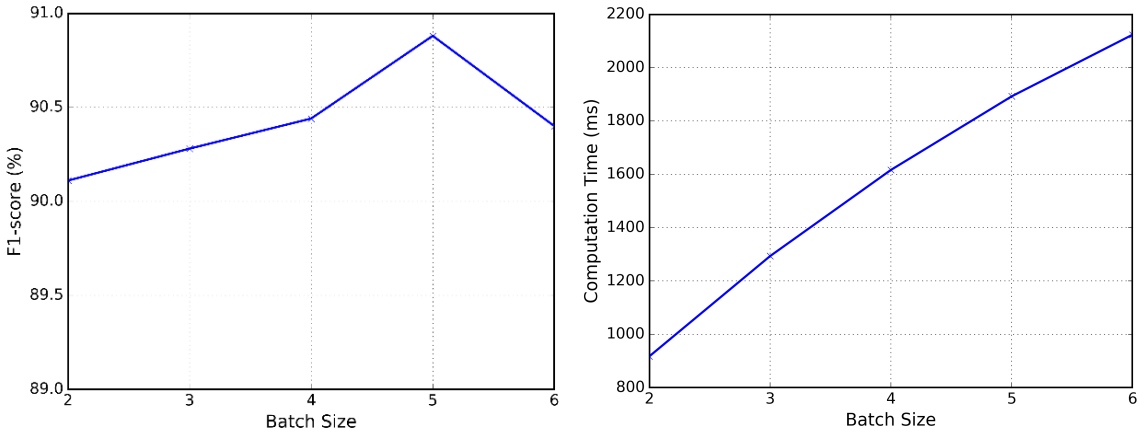


Figure 7.2: Iteration model loss (cross entropy) plot on validation dataset with different learning rate



(a) (b)

Figure 7.3: Effect of varying batch size (a) on F1-score; (b) on computational time

rate, the F1 scored is decreased as the network becomes hard to train.

Effect of Batch Size

Batch size signifies the number of training samples used at an instant during training to make single update of the model parameters. Figure 7.3 shows how F1-score as well as computational time changes with different batch size used for training. In Figure 7.3a, it is observable that, as batch size increases, there is increase in F1-score starting from 90.11%. The F1 score reaches its peak of value 90.88% when five images are used per batch. This increase in accuracy was achieved in expenses with the computational time, reaching 1892ms per iteration for 5 images per batch from 917ms for 2 images per batch for training, as shown in Figure 7.3b. The increase in F1-score with an increase in batch size is sensible as the optimal learning of network is associated with the maximum number and/or a variety of contextual feature that network can learn with. There is a slight decrease at the end may be due to the problem of over-fitting. So, for the further analysis batch size of 5 image was considered.

Effect of Number of Iterations

The number of iterations represents the number of round of optimization applied during training period. The training error decreases with higher number of iteration up to a certain number and beyond that value the network is over fitted to the training data. Figure 7.4 shows the variations in F1-score with an increase in the number of iterations, ranging from 20000 to 60000 with an increment of 10000 iterations. The maximum value of F1-score is obtained at 40000 iteration reaching value 90.82%. As seen in Figure 7.4, the F1- score remains almost constant till 60000 iteration around a value of 90.5%.

Observing loss graph in Figure 7.5a, we can see that the loss (error) decreased and started converging from the stage of 30000 iterations onwards. Also, there is a constant increase in validation accuracy till 40000 iterations (approx.) and then become somehow constant. this trend can be observed in accuracy graph in Figure 7.5b. Furthermore, there is no sign of over-fitting with more number of iteration after convergence, thanks to the use of dropout regularization in network training. So, the obvious choice for the number of iteration is somewhere around $30000 - 40000$.

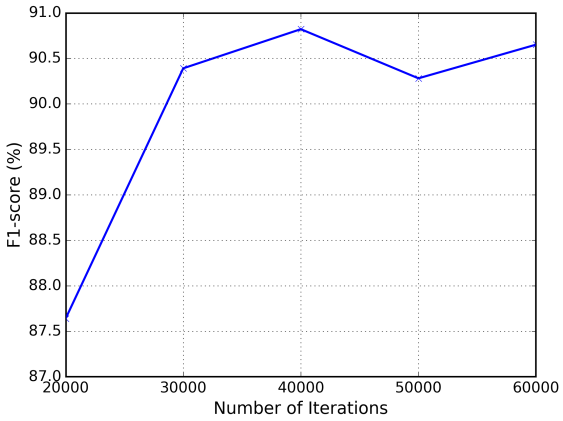
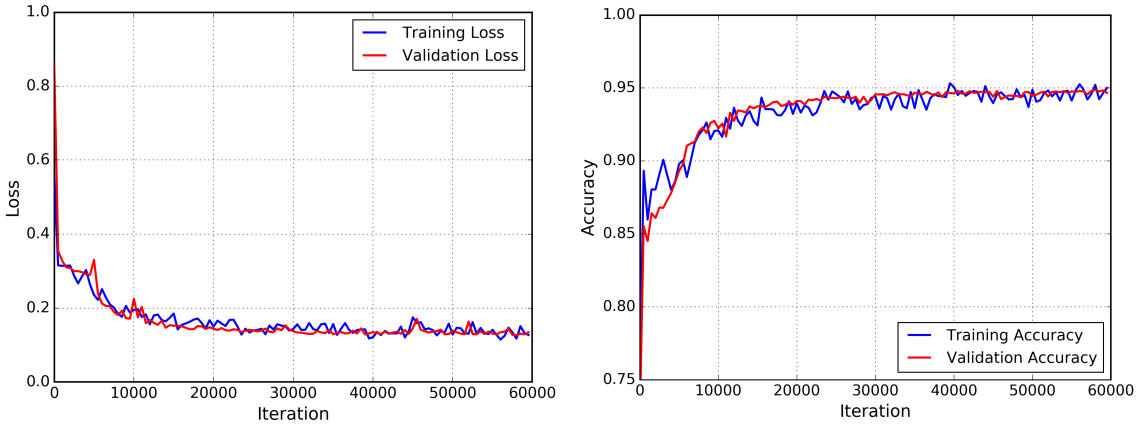


Figure 7.4: Effect of varying learning rate on F1-score

But, one should observe loss graph and accuracy graph while training, to stop the training at an optimal point.



(a) (b)

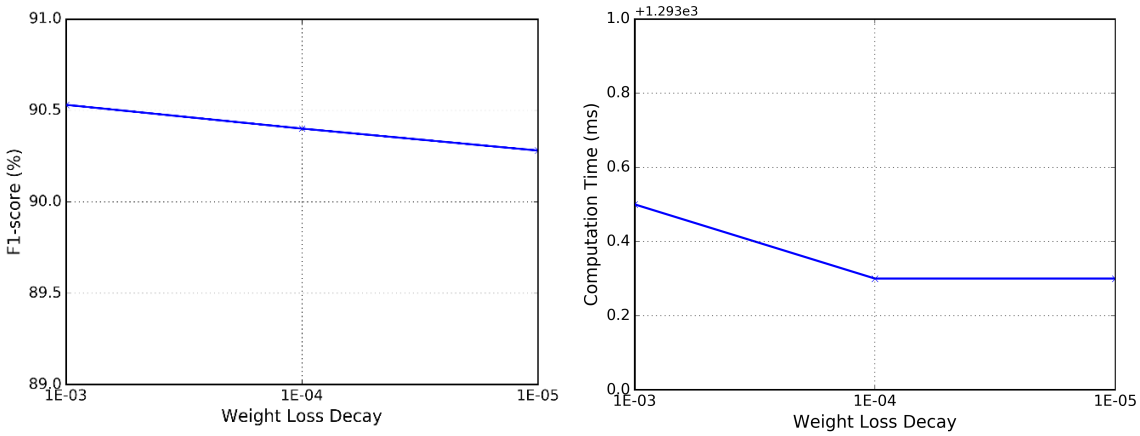
Figure 7.5: Iteration plot on training and validation sets (a) model loss (cross entropy) plot; (b) accuracy plot

Effect of Weight Loss Decay

Figure 7.6 shows how the performance of adopted CNN network changes as we vary the weight loss decay used in the network. We can observe no or little change in F1-score and computational cost as we vary weight loss decay value in a range of $1 * 10^{-3}$ to $1 * 10^{-5}$. By this observation, we can conclude that weight loss decay has little influence on the performance and computational cost of CNN network adopted. For future analysis, weight loss decay value of $1 * 10^{-5}$ was chosen.

Effect of Dropout

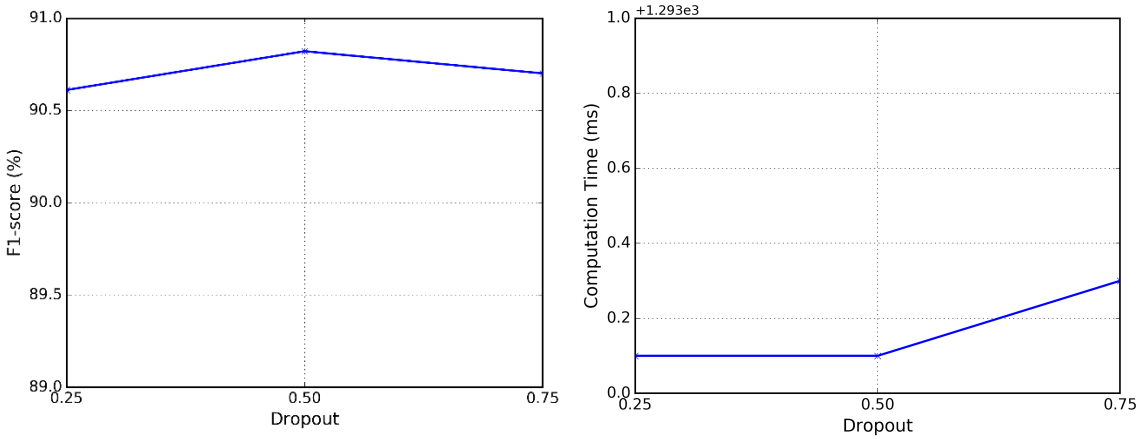
Dropout probability (p) value signifies the percentage of neuron that participates in forward and back-propagation and rest value signify the percentage of a neuron to drop out temporarily at that moment for the propagation of parameters. Figure 7.7 shows the change in performance of adopted CNN with change in dropout value. We can see no notable increment in F1-score (from 90.61% to 90.70%) for the increase in a value of dropout from 0.25 to 0.75 respectively. Additionally, there is no meaningful change in computational time for a change in dropout, remain somehow



(a) (b)

Figure 7.6: Effect of varying weight loss decay (a) on F1-score; (b) on computational time

stagnant at 1293.1 ms. This is well illustrated on figure 7.7a and 7.7b.



(a) (b)

Figure 7.7: Effect of varying dropout (a) on F1-score; (b) on computational time

So, according to analysis, any value of dropout can be chosen as there is no significant improvement in performance for changing dropout regularization probability. But, Srivastava et al. (2014) suggested that in maximum case, using 50% dropout results in the maximum amount of regularization. So, dropout value of 0.5 was chosen for further analysis.

7.1.2 CRFs Parameter Tuning Results

For determining optimal value of appearance kernel parameters (σ_α, σ_β), smoothing kernel parameter (σ_ρ) and number of iterations, parameter tuning experiment was conducted. Figure 19 shows the results of the parameter tuning experiments for CRF post processing algorithm described in subsection 3.4

.

Each of the parameter shows increasing trend of F1-score for an initial increase of value and then onward shows decreasing trend. Figure 7.8a shows that there is an increase in F1-score when we increase a value of parameter σ_α from 2 to 3; but after that, it shows a continuous decrease

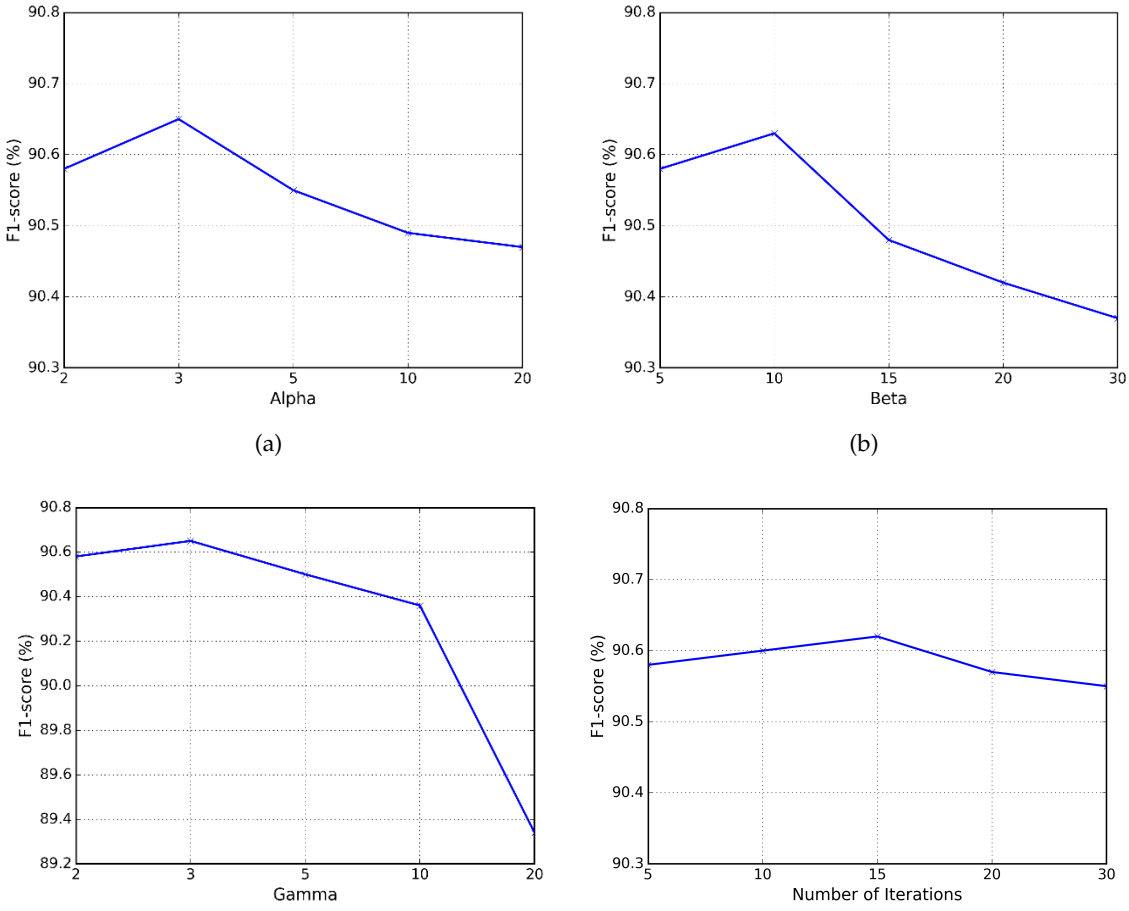


Figure 7.8: Effect of varying parameters of CRFs post processing on F1-score (a) appearance kernel σ_α; (b) appearance kernel σ_β; (c) smoothing kernel σ_η and (d) number of iterations.

in F1-score. However, the change in a value of σ_α has no significant effect in F1-score having an only difference of approximately of 0.1%. Similar trend can be seen for the parameter (σ_ρ) in figure 7.8c. But decrease in F1-score after the value of 3 is rapid as compared to F1-score for σ_α. Figure 7.8b illustrates the behavior of performance of the algorithm in term of F1-score with respect to parameter σ_β. Similar trend as other parameters can be seen with an increase of the value of F1-score at an initial value (from 5 to 10) and subsequently decrease then after. However, figure 7.8d shows the stagnant behavior of F1-score with an increase in the number of iterations.

Although there are some trends in the change in value of F1-score by changing the parameters of the CRFs algorithm, there is not much change in F1-score quantitatively. This overall scenario tells that there seems no significant change in F1-score by changing the value of the parameters. But, the values with the highest F1-score were selected for optimal value of parameters of CRFs.

7.2 Comparison of Variation of Proposed Classifier

In this sub-section, comparison of variation of the proposed model was done using several measures explained in section 6.2. This experiment aims to illustrate that each of the proposed strategies, presented in section 4.1 can really improve the performance. Firstly, ELU-FCN is compared to FCN for the ELU strategy. Secondly, ELU-FCN is compared to ELU-FCN-CRFS for CRF strategy. Finally,

all the variations are compared for the significance of combined proposed strategy. The Figure 7.9 shows that each model is properly set up and trained until the loss converges and accuracy reached to the maximum. The best iteration of FCN and ELU-FCN model is 40000 iterations (approx. 40 epoch).

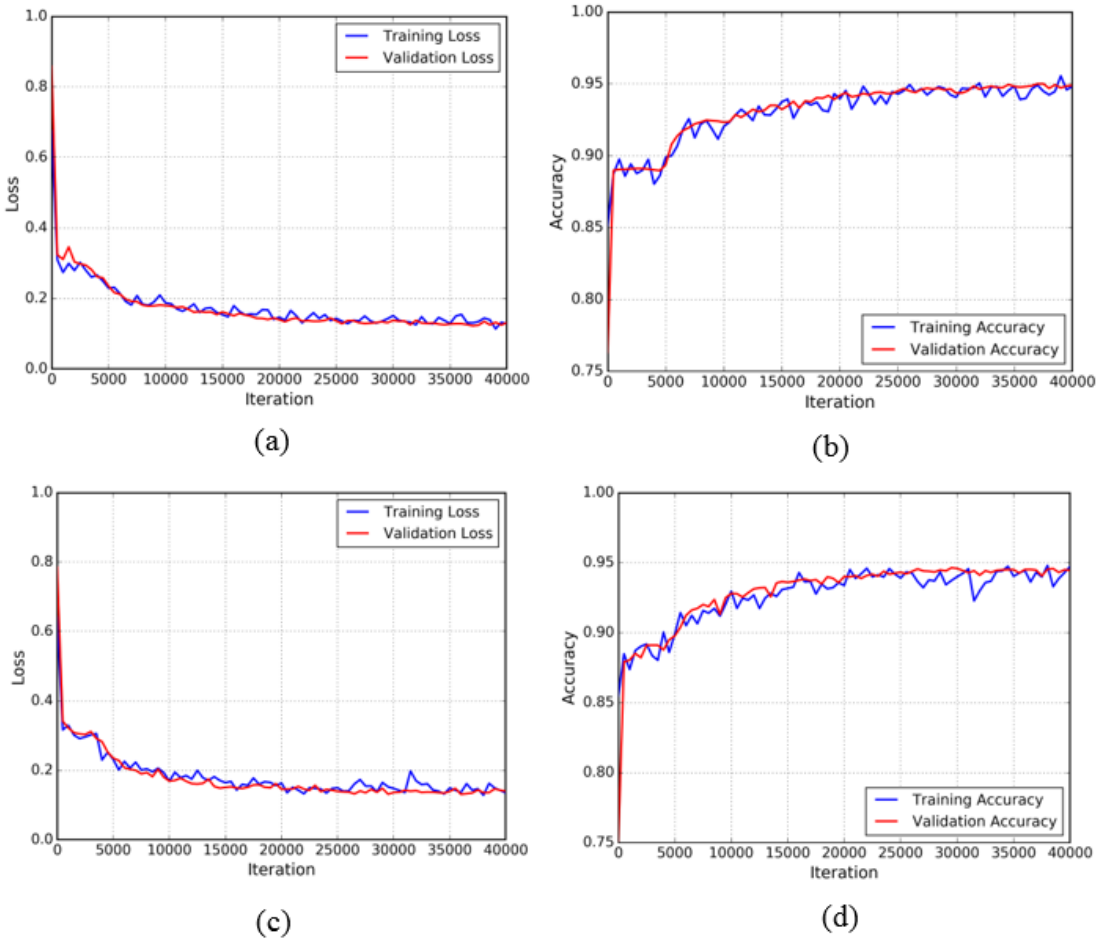


Figure 7.9: Iteration plot on Massachusetts satellite data sets of variation of proposed methods i.e. FCN-8s and ELU-FCN. X refers to the iteration and y refers to the different measures. Each row refers to different model. (a) plot of model loss (cross entropy) on training and validation datasets; and (b) plot of accuracy on training and validation datasets.

7.2.1 Performance Analysis for the Variation of the Classifier

This section presents and discusses the comparative analysis of the classifiers using the performance metrics described in section 6.2.

The result is shown in Table 7.1, the comparison between baselines (FCN) and variations of the proposed techniques. It shows that our proposed network with all strategies (ELU-FCN-CRFs) outperforms other methods. More details are discussed below to show that each of the proposed techniques can really improve an accuracy.

Result of Enhanced FCN (ELU-FCN)

Our first strategy aims to increase an accuracy of the network by using ELU as an activation function (used ELU-FCN) rather than the traditional one, ReLU (use in FCN).Table 7.1 shows that

	Models	**Precision** (%)	**Recall** (%)	**F1-score** (%)	**IOU** (%)
Baseline	FCN	94.76	91.63	93.09	86.96
Variation Method	ELU-FCN	94.79	93.42	93.81	88.93
Variation Method	ELU-FCN-CRFs	95.07	93.40	93.93	89.08

Table 7.1: Results on the testing data of Massachusetts aerial image between the variations of our proposed techniques in terms of precision, recall, F1-score, and IOU measure

ELU-FCN (93.81%) outperform the original FCN (93.09%) in term of F-score, with an increase in F1-score of around 1%. There is also an improvement in IOU measure value from 86.96% to 88.93% by choosing ELU-FCN. This rise in value of F1-score and IOU measure value is mainly due to the higher recall. The increase in recall (around 2%) for ELU-FCN signifies the increased robustness of modified FCN to predict the buildings. But, the precision value remains stagnant which suggests that modified FCN didn't significantly contribute on reducing the noise as compared to that predicted by FCN network. In both models, it is observable that precision value is always greater than recall value. This signifies the model generates less false buildings as compared it misses to predict the correct buildings. This is mainly due to model's weakness to predict smaller buildings. This suggests that ELU is more robust than ReLU in detecting building pixels. The experiment confirms that use of ELU enhances the performance of CNN classifier.

Result of Post Processing CRFs (ELU-FCN-CRFs) on Enhanced FCN (ELU-FCN)

Further improvement is done by integrating CRFs into the modified CNN network as discussed in section 6.1.3. The strategy to use CRF is to sharpen the building boundaries and to filter false building patches. The Table 13 shows that the F1-score of ELU-FCN-CRFs (93.93%) is a bit superior to ELU-FCN (93.81%). Similarly, this combined method shows its slight superiority in term of IOU measure value (89.08%) than ELU-FCN (88.93%) with an increase of 0.1%. This increase in F1-score and IOU measure value as compared to ELU-FCN is majorly attributed to increasing in precision (an increase of 0.4%); with no change in recall value. This signifies that post processing CRFs refined the noise (false buildings) produced by ELU-FCN. However, the constant value of recall signifies that CRFs is not able to recover the building pixels. Additionally, the precision value is greater than recall value for both model, as the same scenario seen in previous models. This experiment concludes that there is little but not significant improvement in adding post processing CRF, which confirms the experiments conducted by Long et al. (2015) numerically.

Combined Result

The combined result shows that F1-score of ELU-FCN-CRFs (93.93%) is superior to FCN-8s (93.09%) and ELU-FCN (93.81%). Similarly, the proposed method also showed its superiority in term of IOU measure value, with an increase in 2.1% than FCN network. This increase in F1-score and IOU measure value is attributed to increase in both precision and recall value than original FCN network. This suggests that improved FCN is able to reduce false buildings as well as able to increase in prediction capability of network. So, we can conclude that there is a significant improvement in combined method than base FCN network.

7.2.2 Visual Inspection and Comparison of Classification Maps

Visual inspection and comparison of classification maps are done by taking the representative area for the images with special focus on different characteristics of buildings and surrounding. Figure 7.10 presents an example of an area from Massachusetts data and result of predictions after applying FCN network, modified FCN network and post processing CRFs.

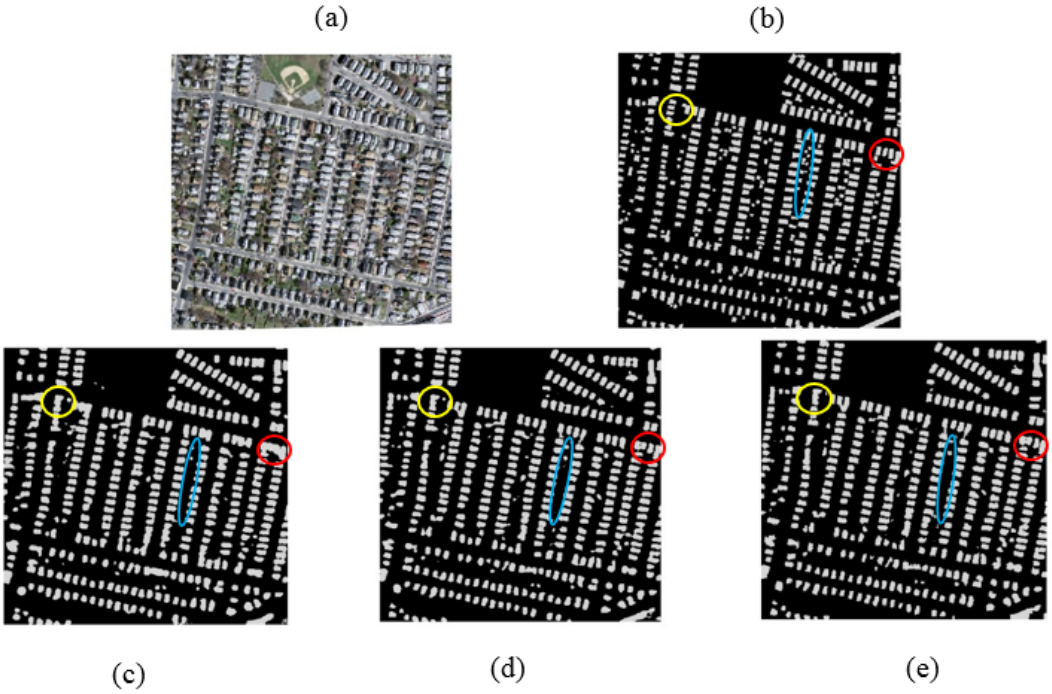


Figure 7.10: Visual comparison of three variation of proposed techniques using sample aerial test images of Massachusetts area. (a)original input image; (b) ground truth map; (c) output of FCN; (d) output of ELU-FCN; and (e) output of ELU-FCN-CRFs.

The predicted labels, at Figure 7.10(c), shows that FCN-8s network assigns same class to most of the buildings, except some adjacent small buildings. They either appear as one connected region or doesn't appear as buildings. Figure 7.10(d) shows output of predicted label after the introduction of ELU-FCN network. This shows improved quality of predicted output than base FCN network. Some of the adjacent connected buildings got well segmented by using ELU-FCN network, but this is not valid for all the buildings. This is well demonstrated by comparing building outputs encircled by the red circle, where an introduction of ELU-FCN improved the result than FCN network. This verifies the reason behinds little increase in precision value when base FCN-8s model is improved. Furthermore, CRFs is capable to filter false building patches (see Figure 7.10(e)). This characteristic of CRFs helped classifier to increase further the precision. Many connected regions predicted by base FCN are neither be refined by ELU-FCN nor by ELU-FCN-CRFs, especially at dense residential area where there is low spacing between the buildings. This is demonstrated by comparing the results for the region encircled in the yellow circle. This is the main reason behind only little but insignificant increase in precision value after choosing ELU-FCN over FCN. To eliminate this error, it may require another level of segmentation to differentiate building from the background, as suggested by Yu et al. (2016). Similarly, smaller buildings are not well predicted by either of these classifiers. These are either absent in the predicted building label or joined with adjacent buildings to form one building, enclosed by the blue circle. It is due to the use of fixed

receptive field (7 pixel), which is also explained in detail by Noh et al. (2015).

The FCN network produced irregular building outlines as compared to the ground truth for both small and big buildings. There is little enhancement in the boundary of the buildings by adopting ELU-FCN. However, an introduction of post processing CRFs able to further refine the shape of the buildings. This is demonstrated in Figure 7.11 with area enclosed by the green circle.

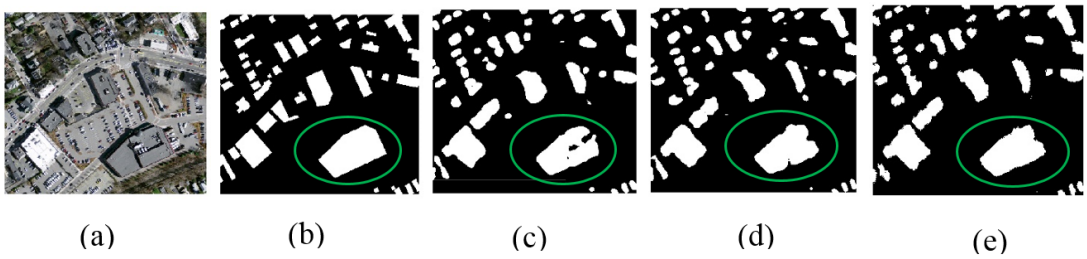


Figure 7.11: Visual comparison of three variation of proposed techniques on big buildings using sample aerial test images of Massachusetts area. (a)original input image; (b) ground truth map; (c) output of FCN; (d) output of ELU-FCN; and (e) output of ELU-FCN-CRFs.

In summary, it can be concluded that the performance of models for predicting big buildings is better than that of the smallest buildings but less accurate than compared to the medium-sized buildings. The main reason behind this is due to the use of the constant receptive field. In addition, although there is no significant improvement in quantitative analysis using post-processing CRFs; qualitative results show good results especially in case of edge enhancement of buildings and refinement of big buildings. Low improvement in the quantitative measure can be attributed to its low performance in case of small and nearby buildings, which covers the major part of the study area.

To have a deep insight of the model performance and interpretation of the quantitative measure of performance, we visualized the prediction of three networks. This further analyzes the ability of the network to extract buildings with variation in appearance, size, occlusion, and denseness. Representative images are chosen from test dataset and visualized with the prediction from three networks. Figure 7.12 visualizes the predictions on the parts of the test set by color coding, in which each of the colors represent whether it is true or false.

The progression from left to right in each row in Figure 7.12 suggests that the proposed methods worked well with modification of base FCN network. The figures also suggest that the network is good at detecting multiple sized buildings with different appearances and at different circumstances (shown by green color); whether it be large and high-rise building area or residential area or industrial area.The false positives (shown by blue pixels) and false negatives (shown by red pixels) present in predicted labels shows several failure cases for our proposed model and illustrate several problems with the data. The network is penalized for making a building prediction where there is building in the image but not in the ground truth, showing up in the prediction as false positive. Besides that, small adjacent buildings are appeared as a connected region, giving rise to false positive at the spacing between buildings. This is because of low spacing between buildings and due to shadows cast by buildings or occlusion on the spacing between them. Majority of buildings which are classified as false negative in the predicted label are smaller buildings, which

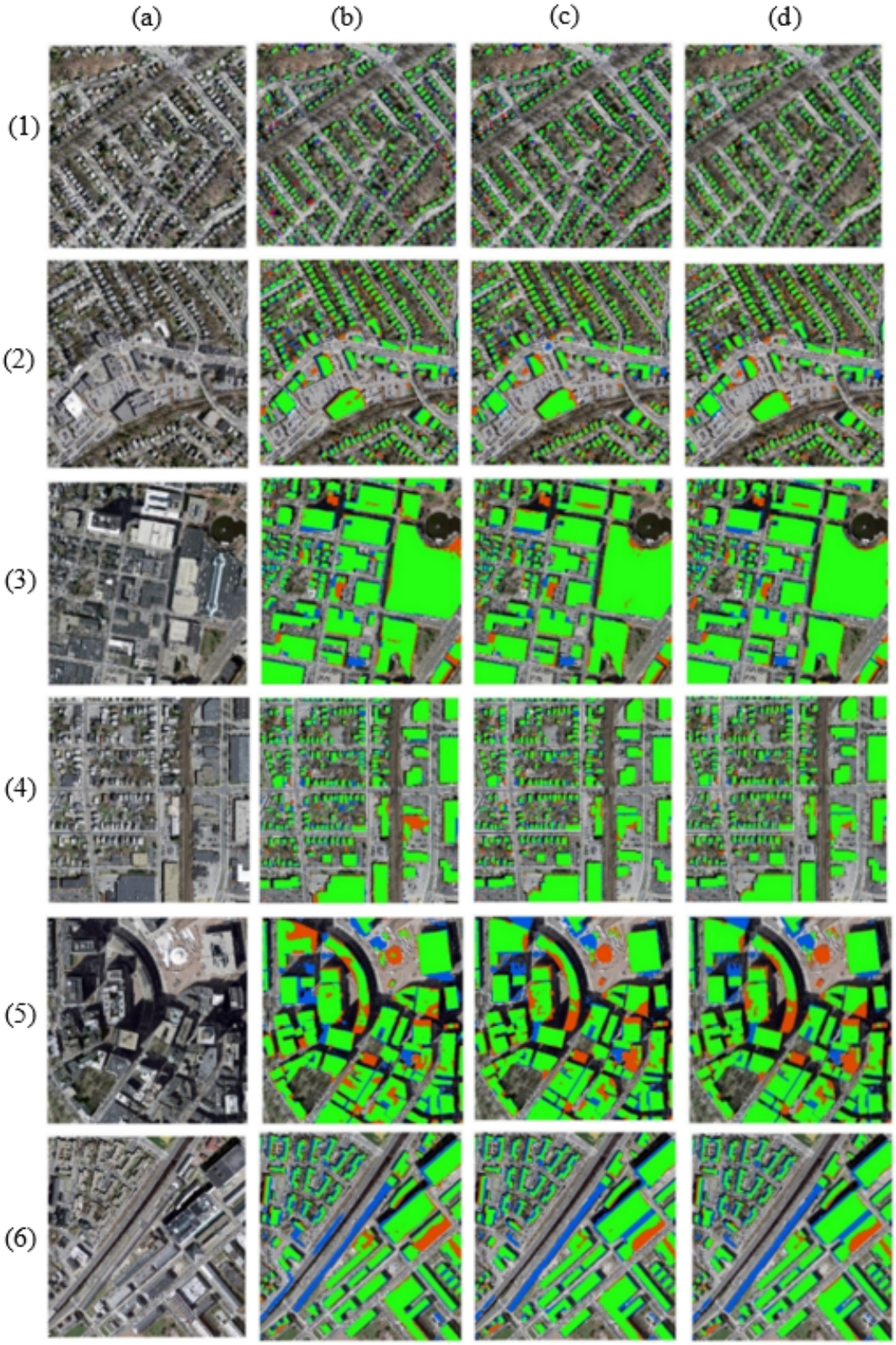


Figure 7.12: Visualization of prediction of three variation of proposed model on building detection tasks with original extracted test images of Massachusetts dataset. (a) input images. (b) result of base FCN network. (c) results of ELU-FCN and (d) results of ELU-FCN-CRFs. Green pixels are true positives, red pixel are false negative, blue pixels are false positive and background pixels are true negatives.

can be attributed to the constant receptive field used in the network. Moreover, false negative in some large building is also due to the use of the constant receptive field. Another reason for false negative is also due to the shadow of a tall buildings and tree cover over the buildings.

Table 7.2 shows the resulting F1-score for each patch of images. We can see that the F1-score of the proposed method (ELU-FCN-CRFs) higher than FCN and ELU-FCN in almost every patch of images. But, ELU-FCN-CRFs is superior to ELU-FCN only by the difference of 0.03%. However, there is much more improvement in F1-score of ELU-FCN-CRFs than that of FCN by a margin of 1.1%.

Image ID	**1**	**2**	**3**	**4**	**5**	**6**	**Mean**
FCN	97.13	95.22	95.06	97.64	91.05	88.48	94.09
ELU-FCN	97.79	95.33	**97.70**	98.31	91.36	90.44	95.16
ELU-FCN-CRF	**97.89**	**95.36**	97.69	**98.37**	**91.38**	**90.45**	**95.19**

Table 7.2: F1-score of variation of proposed techniques at selected region of test images.

7.2.3 On Computational Time and Complexity

All the analysis with the different classifiers were run on NVIDIA Tesla K40 of 10968 MiB. There is no significant effect in computational time in changing activation function from traditional ReLU activation function to ELU activation function. Both of them took around 21 hours to train up to 40000 iterations (roughly 40 epoch). The addition of CRF post processing increased the computational time of the ELU-FCN-CRF classifier. This added around half an hour additional time. In term of complexity, FCN and ELU-FCN are equally complex with the same series of convolution layers followed by nonlinear activation and pooling operations. Additionally, post processing CRFs adds some complexity; but far less as compared to CNN networks. So, we can say that proposed method (ELU-FCN-CRFs) is a bit more complex than FCN and ELU-FCN.

7.3 Comparison with Pre-existing Classifier

7.3.1 Performance Comparison

Table 7.3 shows the result of the different models including the adopted approach using Massachusetts building dataset. Note that the proposed method has been implemented and tested on an experimental dataset, whereas other three results are adopted from original published paper Marcu and Leordeanu (2016). The result shows that our proposed method is superior to all other method except one from Marcu and Leordeanu (2016), in term of F1-score. Our proposed method yields higher F1-score than Mnih (2013) and Saito et al. (2016), at 1.82% and 1.63% respectively but lags behind Marcu and Leordeanu (2016) only by a margin of 0.3%. It is also interesting to show that all of the variations of the proposed model stood superior than the result from [Mnih (2013)] and [Saito et al. (2016)].

Model	Mnih (2013)	Saito et al. (2016)	Marcu and Leordeanu (2016)	Proposed method
F1-score	92.11	92.30	94.23	93.939

Table 7.3: Results on the testing data of Massachusetts aerial image in term of F1-score.

7.3.2 On Computational Complexity

Apart from performance measure, it is worth comparing the model based on computational complexity. The difference between proposed CNN and other approaches is the use of pixel to pixel based (end to end) rather than patch-based approach. The complexity of the CNN network generally associated with the number of parameters to be learned during training. The CNN-based classifier used by Mnih (2013) and Saito et al. (2016) contains only three convolutional layers followed by two fully connected layers. Whereas, there is tremendous increase in parameters to be learned during training when used network proposed by Marcu and Leordeanu (2016). This is due to the reason that it combines VGG-16 with Alex-Net with an addition of extra three fully connected layers at the end. Fully convolutional networks only have parameter equivalent to only one fully connected layer in place of 3 fully connected layers of VGG-16. This implies FCN network has 13 layers of convolution and equivalent convolutional layer of a one fully connected layer. The detail explanations are given in section 3.5.

So, considering a complexity of the network, CNN used by Mnih (2013) and Saito et al. (2016) are least complex. Chosen FCN network is a bit complex than the previous one due to more number of convolution layers (13 compared to 3). But, CNN network adopted by Marcu and Leordeanu (2016) is far more complex due to the combination of two individual CNN network each having same or more complexity as FCN at one place, with more fully connected layers at the end.

7.4 Summary of Results

In summary, the thesis presents the results of all analysis carried out on the proposed algorithm for building extraction.The results from CNN sensitivity analysis on the hyper-parameters: learning rate, batch size, number of iterations, weight decay rate and dropout are presented. For two hyper-parameters called batch size and number of iteration, increasing the value of the hyper-parameters also increased F1-score until reaching to peak until certain value; than they saw a drop in F1-score. For the learning rate, the converse is true as there is decrease in F1-score with increase in learning rate. But, for weight decay rate and dropout hyper-parameters, there is little or no effect on F1-score in varying the hyper-parameters. Similar trend is seen by the change in hyper-parameters on the computational time. There is increased trend on computational time with increase in batch size and number of iterations, decreasing trend with increase in learning rate and little or no effect with change in weight loss decay and dropout. Furthermore, an addition of CRFs for post processing further reduces the noise that remains after CNN. This improves the F1-score in expenses of computational time, but the change is not significant quantitatively. However, the qualitative analysis shows good improvement in prediction results. We first compared the variation of the proposed method; the proposed method outperforms all the variation of it in term of all performance measure (qualitative and quantitative) except computational time. When compared the proposed methods with the pre-existing methods, proposed CNN classifier outperforms the network proposed by Mnih (2013) and Saito et al. (2016) in term of F1-score; however it lags behind by small margin with the model proposed by Marcu and Leordeanu (2016). But, the proposed method is far less complex than network proposed by Marcu and Leordeanu (2016) with much less number of parameters to be learned.

With the results discussed in this chapter, the thesis is concluded in the next chapter, where the objectives presented in chapter 1 are reviewed in line with our findings and discussion from the

experiments and analysis carried out.

8 CONCLUSION AND FUTURE WORKS

8.1 Conclusion

The main objective of the research conducted in the thesis is to develop a new method for building extraction from aerial/satellite image exploiting deep learning-based CNN approach. Firstly, existing CNN network architectures used for building extraction as well as for semantic segmentation were reviewed . The best CNN architecture (fully convolution network) was chosen because of its balanced strength towards accuracy and network complexity. The effect of varying the CNN hyperparameters on the performance of classifier were investigated. From the knowledge obtained from sensitivity analysis experiments, the setup for designing CNN was finalized for classification. The original FCN model was modified by introducing ELU in place of ReLU to speed up learning and gain high classification accuracy. Additionally, post classification processing CRF algorithm was implemented for improvement of boundary of buildings and to further increase in the performance. The comparison of proposed CNN method against variations of the proposed method was carried out using various performance measures. The performance measures used in the comparison includes precision, recall, F1-score, IoU measure, visual inspection, and comparison of classification maps and computational time and complexity. The experiment was conducted on a Massachusetts building dataset and the chosen method was compared with previously used CNN architecture for building extraction using the same dataset used in the thesis. The preliminary results on comparison of variation of proposed method suggest that our proposed ELU-FCN-CRFs outperforms the original and other variation of proposed methods on aerial imagery for F1-score and IoU measure with little or negligible increment in computational time. For comparison of proposed method with all other existing methods/classifier, the proposed method outperforms the network proposed by Mnih (2013) and Saito et al. (2016) in term of F1-score; however, it lags behind by small margin with the model proposed by Marcu and Leordeanu (2016). But, focusing on network complexity, the proposed method is far more less complex than network proposed by Marcu and Leordeanu (2016).

The objectives of the thesis are discussed here which is presented in chapter 1, with its explanations.

1. To review and evaluate the potential of state-of-art deep learning algorithms to automatically extract building using very high-resolution satellite imagery.

   In chapter 2, the different CNN based network architectures were discussed for building extraction and semantic segmentation. Among available networks, evaluation was done based on the characteristics demanded fulfilling main objectives: high accuracy, easy implementation, and less expensive computation. Some recent improvement in deep learning such as activation functions as well as post processing techniques were evaluated for further improvement of potential algorithm for building extraction.

2. To design, implement and analyze the performance of a working streamlined classifier based on the chosen CNN architecture .

   Based on the selected network architecture at section 2.4, some theoretical background on how the architecture operates is presented and discussed in chapter 3 and 4. To determine the optimal structure of the CNN classifier, design experiments were executed, as presented in chapter 5. The effect of variation in hyper-parameters such as learning rate, batch size, number of iterations, weight decay rate and dropout on the performance of classifier were carried out in CNN sensitivity experiments. The results in subsection 7.1.1 show the effect of variation in hyper-parameters on the F1-score of classification results and computational time taken during training. For hyper-parameter batch size and the number of iteration, increasing the value of the hyper-parameters also increase F1-score until reaching to peak until certain value than they see a drop in F1-score. For the learning rate, the converse is true as there is decrease in F1-score with increase in learning rate. But, for weight decay rate and dropout hyper-parameters, there is little or no effect on F1-score in varying the hyper-parameters. For a computational time, there is increasing trend for an increase in batch size; decreasing trend for an increase in learning rate and no effect on changing weight loss decay and dropout regularization. This experiment gives an optimal value of hyper-parameters to implement the designed CNN architecture, presented in subsection 6.1. Furthermore, parameter tuning experiments were performed to determine the optimal value of parameters for post processing CRFs. Experiments showed that there is no significant difference in changing each parameter on final performance of the classifier. However, parameter yielding relative high value of F1-score considered as optimal one, which is presented in subsection 6.1.

3. To compare the performance of formulated deep learning classifier against alternative classification methods

   The variations of the proposed classifier were evaluated using various performance measures: relaxed precision, recall, F1-score, IoU measure, visual inspection and comparison of classification maps and computational time and complexity, presented in subsection 6.2. However, comparison of proposed classifier with pre-existing classifier was done based on relaxed F1-score and network complexity.

   The comparison of variation of the proposed method is presented in subsection 7.2. The finding shows that our proposed method (ELU-FCN-CRFs) outperforms the two variations of proposed methods (original FCN-8s and ELU-FCN) on aerial imagery for F1-score and IoU measure with little or negligible increment in computational time. In comparison with other existing classifiers, proposed CNN classifier outperforms the network proposed by Mnih (2013) and Saito et al. (2016) in term of F1-score; however it lags behind by small margin on F1-score with the model proposed by Marcu and Leordeanu (2016). But, the proposed method is far less complex than network proposed by Marcu and Leordeanu (2016) with much less number of parameters to be learned.

At the end, the thesis presented a CNN based deep learning approach for automatic building extraction. For this, various existing CNN architecture for semantic segmentation were investigated

and best one was chosen one based on both performance and complexity. The performance of chosen approach was analyzed with the existing methods and found that the chosen approach stood superior considering performance measure and network complexity at one place. In conclusion, the chosen approach has great potential and can be beneficial in automatic building extraction which can be subsequently used for several large-scale monitoring systems crucial for urban planning, disaster management, navigation and several other geospatial applications.

8.2 Future Works

The results presented in chapter 7 have much scope for improvement, with more possible scenarios to be explored. The listed possible ways to improve the result are recommended.

- In the thesis, only some data augmentation techniques were used. More variety of data augmentation techniques are recommended, which can possibly increase the performance.
- The performance of the network is limited by using the constant receptive field, which has a direct effect on predicting small and big buildings. So, hierarchical increase in receptive field is recommended to nullify the effect of the constant receptive field.
- There are also many connected neighboring building patches for the small building where there is less gap between them, increasing false positive. Post processing CRFs does not even eliminate this error. So, one more level of segmentation to distinguish background from the buildings is recommended to remove this error, hence increasing the accuracy of output.
- The manual approach was used for hyper-parameters fine tuning in most case taking only few set of hyper-parameters. Extensive experiment in the large subset of hyper-parameters can be performed or algorithmic approach can be adopted to explore the hyper-parameter space to ensure that the best hyper-parameters are selected.
- Another state of art technology of CNN based network for semantic segmentation can be explored and implemented if someone is really looking towards an increase in performance regardless the computational complexity.
- The problem for multi-class segmentation, dealing with the detection of roads and buildings both, can also be achieved by using the proposed network. These two are strong candidate for determining urban growth and change. So, this approach can be a strong component for assessing and monitoring urban growth.
- Exploration of more choice of semantic segmentation, optimization technique and/or post processing techniques is also recommended for future work.

Bibliography

ALSHEHHI, R., P. R. MARPU, W. L. WOON, and M. DALLA MURA (2017). "Simultaneous extraction of roads and buildings in remote sensing imagery with convolutional neural networks". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 130, pp. 139–149.

AUDEBERT, N., B. LE SAUX, and S. LEFÈVRE (2016). "Semantic segmentation of earth observation data using multimodal and multi-scale deep networks". In: *Asian Conference on Computer Vision*. Springer, pp. 180–196.

BADRINARAYANAN, V., A. KENDALL, and R. CIPOLLA (2017). "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39(12), pp. 2481–2495.

BERGSTRA, J. and Y. BENGIO (2012). "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13(Feb), pp. 281–305.

BITTNER, K., S. CUI, and P. REINARTZ (2017). "Building extraction from remote sensing data using fully convolutional networks". In: *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 42, p. 481.

CASADO, R. and M. YOUNAS (2015). "Emerging trends and technologies in big data processing". In: *Concurrency and Computation: Practice and Experience* 27(8), pp. 2078–2091.

CHEN, L.-C., G. PAPANDREOU, I. KOKKINOS, K. MURPHY, and A. L. YUILLE (2016). "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *arXiv preprint arXiv:1606.00915*.

CLEVERT, D.-A., T. UNTERTHINER, and S. HOCHREITER (2015). "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289*.

FAA (2017). *UAS integration pilot program*. https://www.faa.gov/uas/programs_partnerships/uas_integration_pilot_program/, . [Online; accessed 12-11-2017].

FU, G., C. LIU, R. ZHOU, T. SUN, and Q. ZHANG (2017). "Classification for high resolution remote sensing imagery using a fully convolutional network". In: *Remote Sensing* 9(5), p. 498.

FUKUSHIMA, K. and S. MIYAKE (1982). "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, pp. 267–285.

GERKE, M, C HEIPKE, and B.-M. STRAUB (2001). "Building extraction from aerial imagery using a generic scene model and invariant geometric moments". In: *Remote Sensing and Data Fusion over Urban Areas, IEEE/ISPRS Joint Workshop 2001*. IEEE, pp. 85–89.

GLOBE, D. (2017). *Open data for disaster recovery*. https://www.digitalglobe.com/. [Online; accessed 12-11-2017].

HE, K., X. ZHANG, S. REN, and J. SUN (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

HE, K., X. ZHANG, S. REN, and J. SUN (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

HU, F., G.-S. XIA, J. HU, and L. ZHANG (2015). "Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery". In: *Remote Sensing* 7(11), pp. 14680–14707.

HUANG, Z., G. CHENG, H. WANG, H. LI, L. SHI, and C. PAN (2016). "Building extraction from multi-source remote sensing images via deep deconvolution neural networks". In: *Geoscience and Remote Sensing Symposium (IGARSS), 2016 IEEE International*. IEEE, pp. 1835–1838.

HUBEL, D. H. and T. N. WIESEL (1962). "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160(1), pp. 106–154.

HUERTAS, A. and R. NEVATIA (1988). "Detecting buildings in aerial images". In: *Computer vision, graphics, and image processing* 41(2), pp. 131–152.

INGLADA, J. (2007). "Automatic recognition of man-made objects in high resolution optical remote sensing images by SVM classification of geometric image features". In: *ISPRS journal of photogrammetry and remote sensing* 62(3), pp. 236–248.

KARANTZALOS, K. and N. PARAGIOS (2009). "Recognition-driven two-dimensional competing priors toward automatic and accurate building detection". In: *IEEE Transactions on Geoscience and Remote Sensing* 47(1), pp. 133–144.

KIM, T. and J.-P. MULLER (1999). "Development of a graph-based approach for building detection". In: *Image and Vision Computing* 17(1), pp. 3–14.

KRÄHENBÜHL, P. and V. KOLTUN (2011). "Efficient inference in fully connected crfs with gaussian edge potentials". In: *Advances in neural information processing systems*, pp. 109–117.

KRIZHEVSKY, A., I. SUTSKEVER, and G. E. HINTON (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

LAFARGE, F., X. DESCOMBES, J. ZERUBIA, and M. PIERROT-DESEILLIGNY (2010). "Structural approach for building reconstruction from a single DSM". In: *IEEE Transactions on pattern analysis and machine intelligence* 32(1), pp. 135–147.

LEARNING, T. (2017). *Convolutional neural Network for Visual Recognition*. http://cs231n.github.io/. [Online; accessed 12-December-2017].

LECUN, Y., L. BOTTOU, Y. BENGIO, and P. HAFFNER (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86(11), pp. 2278–2324.

LEVITT, S. and F. AGHDASI (1998). "An investigation into the use of wavelets and scaling for the extraction of buildings in aerial images". In: *Communications and Signal Processing, 1998. COMSIG'98. Proceedings of the 1998 South African Symposium on*. IEEE, pp. 133–138.

LIANG-CHIEH, C., G. PAPANDREOU, I. KOKKINOS, K. MURPHY, and A. YUILLE (2015). "Semantic image segmentation with deep convolutional nets and fully connected crfs". In: *International Conference on Learning Representations*.

LILLICRAP, T. P., J. J. HUNT, A. PRITZEL, N. HEESS, T. EREZ, Y. TASSA, D. SILVER, and D. WIERSTRA (2015). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971*.

LIN, M., Q. CHEN, and S. YAN (2013). "Network in network". In: *arXiv preprint arXiv:1312.4400*.

LIU, Z., X. LI, P. LUO, C. C. LOY, and X. TANG (2017). "Deep learning markov random field for semantic segmentation". In: *IEEE transactions on pattern analysis and machine intelligence*.

LONG, J., E. SHELHAMER, and T. DARRELL (2015). "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

MAAS, A. L., A. Y. HANNUN, and A. Y. NG (2013). "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1, p. 3.

MAGGIORI, E., Y. TARABALKA, G. CHARPIAT, and P. ALLIEZ (2016). "Fully convolutional neural networks for remote sensing image classification". In: *Geoscience and Remote Sensing Symposium (IGARSS), 2016 IEEE International*. IEEE, pp. 5071–5074.

MAGGIORI, E., Y. TARABALKA, G. CHARPIAT, and P. ALLIEZ (2017). "Convolutional neural networks for large-scale remote-sensing image classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 55(2), pp. 645–657.

MARCU, A. and M. LEORDEANU (2016). "Dual local-global contextual pathways for recognition in aerial imagery". In: *arXiv preprint arXiv:1605.05462*.

MAYER, H. (1999). "Automatic object extraction from aerial imagery—a survey focusing on buildings". In: *Computer vision and image understanding* 74(2), pp. 138–149.

MISHKIN, D. and J. MATAS (2015). "All you need is a good init". In: *arXiv preprint arXiv:1511.06422*.

MNIH, V. (2013). "Machine learning for aerial image labeling". PhD thesis. University of Toronto (Canada).

MURUGANANDHAM, S. (2016). *Semantic segmentation of satellite images using deep learning*.

MYINT, S. W., N. S.-N. LAM, and J. M. TYLER (2004). "Wavelets for urban spatial feature discrimination". In: *Photogrammetric Engineering & Remote Sensing* 70(7), pp. 803–812.

NAIR, V. and G. E. HINTON (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

NEWS, S. (2017). *U.S. Government eases restrictions on DigitalGlobe*. http://spacenews.com/40874us-government-eases-restrictions-on-digitalglobe/. [Online; accessed 12-12-2017].

NOGUEIRA, K., O. A. PENATTI, and J. A. dos SANTOS (2017). "Towards better exploiting convolutional neural networks for remote sensing scene classification". In: *Pattern Recognition* 61, pp. 539–556.

NOH, H., S. HONG, and B. HAN (2015). "Learning deconvolution network for semantic segmentation". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1520–1528.

O'SHEA, K. and R. NASH (2015). "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458*.

PENG, J and Y. LIU (2005). "Model and context-driven building extraction in dense urban aerial images". In: *International Journal of Remote Sensing* 26(7), pp. 1289–1307.

PLANET (2017). *Planet doubles sub-1 meter imaging capacity with successful launch of 6 skysats*. https://www.planet.com/pulse/planet-doubles-sub-1-meter-imaging-capacity-with-successful-launch-of-6-skysats/. [Online; accessed 22-12-2017].

RONNEBERGER, O., P. FISCHER, and T. BROX (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241.

RUDER, S. (2016). "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747*.

SAITO, S. and Y. AOKI (2015). "Building and road detection from large aerial imagery". In: *Image Processing: Machine Vision Applications VIII*. Vol. 9405. International Society for Optics and Photonics, 94050K.

SAITO, S., T. YAMASHITA, and Y. AOKI (2016). "Multiple object extraction from aerial imagery with convolutional neural networks". In: *Electronic Imaging* 2016(10), pp. 1–9.

SHAH, A., E. KADAM, H. SHAH, S. SHINDE, and S. SHINGADE (2016). "Deep residual networks with exponential linear unit". In: *Proceedings of the Third International Symposium on Computer Vision and the Internet*. ACM, pp. 59–65.

SHOTTON, J., J. WINN, C. ROTHER, and A. CRIMINISI (2009). "Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context". In: *International Journal of Computer Vision* 81(1), pp. 2–23.

SHU, Y. (2014). "Deep Convolutional Neural Networks for Object Extraction from High Spatial Resolution Remotely Sensed Imagery". PhD thesis. University of Waterloo.

SIMONYAN, K. and A. ZISSERMAN (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

SNOEK, J., H. LAROCHELLE, and R. P. ADAMS (2012). "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems*, pp. 2951–2959.

SRIVASTAVA, N., G. HINTON, A. KRIZHEVSKY, I. SUTSKEVER, and R. SALAKHUTDINOV (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15(1), pp. 1929–1958.

SUMER, E. and M. TURKER (2013). "An adaptive fuzzy-genetic algorithm approach for building detection using high-resolution satellite images". In: *Computers, Environment and Urban Systems* 39, pp. 48–62.

SZEGEDY, C., W. LIU, Y. JIA, P. SERMANET, S. REED, D. ANGUELOV, D. ERHAN, V. VANHOUCKE, A. RABINOVICH, et al. (2015). "Going deeper with convolutions". In: Cvpr.

TENSORFLOW (2017). *An open-source machine learning framework for everyone*. https://www.tensorflow.org/. [Online; accessed 11-August-2017].

VAKALOPOULOU, M., K. KARANTZALOS, N. KOMODAKIS, and N. PARAGIOS (2015). "Building detection in very high resolution multispectral data with deep learning features". In: *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*. IEEE, pp. 1873–1876.

VOLPI, M. and D. TUIA (2017). "Dense semantic labeling of subdecimeter resolution images with convolutional neural networks". In: *IEEE Transactions on Geoscience and Remote Sensing* 55(2), pp. 881–893.

WIEDEMANN, C., C. HEIPKE, H. MAYER, and O. JAMET (1998). "Empirical evaluation of automatically extracted road axes". In: *Empirical evaluation techniques in computer vision*, pp. 172–187.

YU, H., W. YANG, G.-S. XIA, and G. LIU (2016). "A color-texture-structure descriptor for high-resolution satellite image classification". In: *Remote Sensing* 8(3), p. 259.

YUAN, J. (2016). "Automatic building extraction in aerial scenes using convolutional networks". In: *arXiv preprint arXiv:1602.06564*.

YUAN, J. and A. M. CHERIYADAT (2014). "Learning to count buildings in diverse aerial scenes". In: *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, pp. 271–280.

A ATTACHMENTS

In this appendix, the tensorflow python codes used for data augmentation, network design, training CNN, CRF post processing, inference and accuracy assessment are presented.

A.1 Data Augmentation and Data Reader

```
#====================================================================
#.......................DATA AUGUMENTATION AND DATAREADER .....................
#====================================================================
#.............................import  libraries ....................
import numpy as np
import os
import scipy.misc as misc
import random
from PIL import Image
#-----------------Class for reading training and validation data---------

class Data_Reader:
# Initiate folders were files are and list of train images
    def __init__(self, ImageDir, GTLabelDir="", BatchSize=1,Suffle=True):
                        #ImageDir directory were images are
                        #GTLabelDir Folder where ground truth
        self.NumFiles = 0     # Number of files in reader
        self.Epoch = 0        # Training epochs passed
        self.itr = 0        #Iteration

#Image directory
        self.Image_Dir=ImageDir     # Image Dir
        if GTLabelDir=="":        # If no label dir use
            self.ReadLabels=False
        else:
            self.ReadLabels=True
        self.Label_Dir = GTLabelDir
        # Folder with ground truth pixels was annotated (optional for
        training only)
        self.OrderedFiles=[]
        # Read list of all files
        self.OrderedFiles += [each for each in os.listdir(self.Image_Dir) if
        each.endswith('.PNG') or each.endswith('.JPG') or
        each.endswith('.TIF') or each.endswith('.GIF') or
        each.endswith('.png') or each.endswith('.jpg') or
        each.endswith('.tiff') or each.endswith('.gif') ]

 # Get list of training images
        self.BatchSize=BatchSize  #Number of images used in single training operation
        self.NumFiles=len(self.OrderedFiles)
        self.OrderedFiles.sort()        # Sort files by names
```

```
        self.SuffleBatch()          # shuffle file list

#.............Suffle list of files in group.............................
def SuffleBatch(self):
        self.SFiles = []
        Sf=np.array(range(np.int32(np.ceil(self.NumFiles/self.BatchSize)+1)))
        *self.BatchSize
        random.shuffle(Sf)
        self.SFiles=[]
        for i in range(len(Sf)):
            for k in range(self.BatchSize):
                  if Sf[i]+k<self.NumFiles:
                      self.SFiles.append(self.OrderedFiles[Sf[i]+k])


#.......................Read and augment next batch of images and labels..............
    def ReadAndAugmentNextBatch(self):
        if self.itr>=self.NumFiles:              # End of an epoch
            self.itr=0
            self.SuffleBatch()
            self.Epoch+=1
        batch_size=np.min([self.BatchSize,self.NumFiles-self.itr])


#-----------Augment Images and labels--------------
        for f in range(batch_size):
#.............Read image and labels from files.............................
           Img = misc.imread(self.Image_Dir + "/" + self.SFiles[self.itr])
           Img=Img[:,:,0:3]
           LabelName=self.SFiles[self.itr][0:-5]+".tif"
           # Assume Label name is same as image
           if self.ReadLabels:
              Label= misc.imread(self.Label_Dir + "/" + LabelName)
           self.itr+=1

#...........Set Batch image size according to first image in the batch.................
           if f==0:
                Sy,Sx,Depth=Img.shape
                Sy = np.int32(Sy)
                Sx = np.int32(Sx)
                Images = np.zeros([batch_size,Sy,Sx,3], dtype=np.float)
                if self.ReadLabels: Labels= np.zeros([batch_size,Sy,Sx,1], dtype=np.int)


#................. rotate each image at random angle...................
       ran_number = random.random()
           degree = ran_number*360
           Im = Image.fromarray(Img)
           Img = Im.rotate(degree)
           #Img = misc.imresize(Img, [Sy, Sx], interp = 'bilinear')
           if self.ReadLabels:
               Lab = Image.fromarray(Label)
               Label = Lab.rotate(degree)
               #Label = misc.imresize(Label, [Sy, Sx], interp = 'nearest')

  #................. add random noise to the input image......
           Img = np.float32(Img)
           if np.random.rand() <0.4:
               Img *=np.ones(Img.shape)*0.95 +
```

```
101                 np.random.rand(Img.shape[0],Img.shape[1],Img.shape[2])*0.1
102             Img[Img>255]=255
103             Img[Img<0]=0
104 
105 #---------------Add images and labels to to the batch--------
106             Images[f]=Img
107             if self.ReadLabels:
108                    Labels[f,:,:,0]=Label
109 
110 #.............Return augumented images and labels.....................
111         if self.ReadLabels
112             return Images, Labels       # return image and pixelwise labels
113         else:
114             return Images         # Return image
115 
116 #-----Read next batch of images and labels with no augmentation-------
117 def ReadNextBatchClean(self):   #Read image and labels without augmenting
118         if self.itr>=self.NumFiles:       # End of an epoch
119             self.itr=0
120             #self.SuffleBatch()
121             self.Epoch+=1
122         batch_size=np.min([self.BatchSize,self.NumFiles-self.itr])
123 
124         for f in range(batch_size):
125 ##.............Read image and labels from files..............
126             Img = misc.imread(self.Image_Dir + "/" + self.OrderedFiles[self.itr])
127             Img=Img[:,:,0:3]
128             LabelName=self.OrderedFiles[self.itr][0:-5]+".tif"
129             # Assume label name is same ending
130             if self.ReadLabels:
131                Label= misc.imread(self.Label_Dir + "/" + LabelName)
132             self.itr+=1
133 #............Set Batch size according to first image..............................
134             if f==0:
135                  Sy,Sx,Depth=Img.shape
136                  Images = np.zeros([batch_size,Sy,Sx,3], dtype=np.float)
137                  if self.ReadLabels: Labels= np.zeros([batch_size,Sy,Sx,1], dtype=np.int)
138 
139 #...................Load image and label to batch...................
140             Images[f] = Img
141             if self.ReadLabels:
142                Labels[f, :, :, 0] = Label
143 #.........................Return images and labels..................
144         if self.ReadLabels:
145                return Images, Labels          # return image and and pixelwise labels
146         else:
147                return Images           # Return image
```

A.2 Building Full Convolutional Neural Network

```
1
2  #========================================================================
3  # BUILD FULLY CONVOLUTIONAL NEURAL NET
4  #========================================================================
5
6  #............................... import libraries.............
7  import inspect
8  import os
9  import TensorflowUtils as utils
10 import numpy as np
11 import tensorflow as tf
12
13
14 VGG_MEAN = [103.939, 116.779, 123.68]
15 # Mean value of pixels in R G and B channels
16
17 #.....................Class for building the FCN neural network based on VGG16.........
18 class BUILD_NET_VGG16:
19     def __init__(self, vgg16_npy_path=None):
20         if vgg16_npy_path is None:
21            path = inspect.getfile(BUILD_NET_VGG16)
22            path = os.path.abspath(os.path.join(path, os.pardir))
23            path = os.path.join(path, "vgg16.npy")
24            vgg16_npy_path = path
25             print(path)
26
27         self.data_dict = np.load(vgg16_npy_path,
28 encoding='latin1').item()         #Load weights of trained VGG16 for encoder
29         print("npy␣file␣loaded")
30 #............................... Build Net.....................................
31 def build(self, rgb,NUM_CLASSES,keep_prob):
32 #Build the fully convolutional neural network
33 #(FCN) and load weight for decoder based on trained VGG16 network
34
35         self.SumWeights = tf.constant(
36 0.0, name="SumFiltersWeights")
37 #Sum of weights of all filters for weight decay loss
38
39
40         print("build␣model␣started")
41         rgb_scaled = rgb * 255.0
42
43         # Convert RGB to BGR and substract pixels mean
44         red, green, blue = tf.split(axis=3, num_or_size_splits=3, value=rgb)
45
46         bgr = tf.concat(axis=3, values=[
47             blue - VGG_MEAN[0],
48             green - VGG_MEAN[1],
49             red - VGG_MEAN[2],
50         ])
51
52 #----Build network encoder based on VGG16 network and load the trained VGG16 weights----
53        #Layer 1
54         self.conv1_1 = self.conv_layer_ELU(bgr, "conv1_1")
55         #Build Convolution layer and load weights
56         self.conv1_2 = self.conv_layer_ELU(self.conv1_1, "conv1_2")
```

```
    #Build Convolution layer +Relu and load weights
        self.pool1 = self.max_pool(self.conv1_2, 'pool1')
        #Max Pooling

       # Layer 2
        self.conv2_1 = self.conv_layer_ELU(self.pool1, "conv2_1")
        self.conv2_2 = self.conv_layer_ELU(self.conv2_1, "conv2_2")
        self.pool2 = self.max_pool(self.conv2_2, 'pool2')
        # Layer 3
        self.conv3_1 = self.conv_layer_ELU(self.pool2, "conv3_1")
        self.conv3_2 = self.conv_layer_ELU(self.conv3_1, "conv3_2")
        self.conv3_3 = self.conv_layer_ELU(self.conv3_2, "conv3_3")
        self.pool3 = self.max_pool(self.conv3_3, 'pool3')
        # Layer 4
        self.conv4_1 = self.conv_layer_ELU(self.pool3, "conv4_1")
        self.conv4_2 = self.conv_layer_ELU(self.conv4_1, "conv4_2")
        self.conv4_3 = self.conv_layer_ELU(self.conv4_2, "conv4_3")
        self.pool4 = self.max_pool(self.conv4_3, 'pool4')
        # Layer 5
        self.conv5_1 = self.conv_layer_ELU(self.pool4, "conv5_1")
        self.conv5_2 = self.conv_layer_ELU(self.conv5_1, "conv5_2")
        self.conv5_3 = self.conv_layer_ELU(self.conv5_2, "conv5_3")
        self.pool5 = self.max_pool(self.conv5_3, 'pool5')
##----------Build Net Fully connvolutional layers--------------
W6 = utils.weight_variable([7, 7, 512, 4096],name="W6")
# Create tf weight for the new layer with initial weights with normal
random distribution mean zero and std 0.02
b6 = utils.bias_variable([4096], name="b6")
# Create tf bias for the new layer with initial weights of 0
self.conv6 = utils.conv2d_basic(self.pool5 , W6, b6)
# Check the size of this net input is it same as input or is it 1X1
        self.elu6 = tf.nn.elu(self.conv6, name="elu6")

        self.elu_dropout6 = tf.nn.dropout(
self.elu6,keep_prob=keep_prob)
# Apply dropout for training need to be added only for training

        W7 = utils.weight_variable([1, 1, 4096, 4096], name="W7")   # 1X1 Convolution
        b7 = utils.bias_variable([4096], name="b7")
        self.conv7 = utils.conv2d_basic(self.elu_dropout6, W7, b7)    # 1X1 Convolution
        self.elu7 = tf.nn.elu(self.conv7, name="elu7")
        self.elu_dropout7 = tf.nn.dropout(
self.elu7, keep_prob=keep_prob)
# Another dropout need to be used only for training

        W8 = utils.weight_variable(
[1, 1, 4096, NUM_CLASSES],name="W8")
# Basically the output num of classes imply the output is already the
# prediction this is flexible can be change however in multinet
# class number of 2 give good results
        b8 = utils.bias_variable([NUM_CLASSES], name="b8")
        self.conv8 = utils.conv2d_basic(self.elu_dropout7, W8, b8)
        # annotation_pred1 = tf.argmax(conv8, dimension=3, name="prediction1")
#----------------------------Build Decoder ---------------
        # now to upscale to actual image size
        deconv_shape1 = self.pool4.get_shape()
        # Set the output shape for the the transpose convolution output
        take only the depth since the transpose convolution will have
        to have the same depth for output
```

```
        W_t1 = utils.weight_variable(
[4, 4, deconv_shape1[3].value, NUM_CLASSES],
name="W_t1")
# Deconvolution/transpose in size 4X4
# note that the output shape is of  depth
# UM_OF_CLASSES this is not necessary in will need to be fixed
#if you only have 2 categories

        b_t1 = utils.bias_variable([deconv_shape1[3].value], name="b_t1")
        self.conv_t1 = utils.conv2d_transpose_strided(self.conv8, W_t1, b_t1,
        output_shape=tf.shape(self.pool4))
 # Use strided convolution to double layer size
 # (depth is the depth of pool4 for the later element wise addition
    self.fuse_1 = tf.add(self.conv_t1, self.pool4, name="fuse_1")
# Add element wise the pool layer from the decoder
        deconv_shape2 = self.pool3.get_shape()
        W_t2 = utils.weight_variable([4, 4, deconv_shape2[3].value,
        deconv_shape1[3].value], name="W_t2")
        b_t2 = utils.bias_variable([deconv_shape2[3].value], name="b_t2")
        self.conv_t2 = utils.conv2d_transpose_strided(self.fuse_1, W_t2,
        b_t2, output_shape=tf.shape(self.pool3))
        self.fuse_2 = tf.add(self.conv_t2, self.pool3, name="fuse_2")

        shape = tf.shape(rgb)
        W_t3 = utils.weight_variable([16, 16, NUM_CLASSES,
        deconv_shape2[3].value], name="W_t3")
        b_t3 = utils.bias_variable([NUM_CLASSES], name="b_t3")

        self.Prob = utils.conv2d_transpose_strided(self.fuse_2, W_t3, b_t3,
        output_shape=[shape[0], shape[1], shape[2], NUM_CLASSES], stride=8)

     #----------Transform probability vectors to label maps-----------------
        self.Pred = tf.argmax(self.Prob, dimension=3, name="Pred")
        self.Pred1 = tf.expand_dims(self.Pred, dim = 3)
        self.probabilities = tf.nn.softmax(self.Prob)
        print("FCN model built")
#########################################################################
    def max_pool(self, bottom, name):
        return tf.nn.max_pool(bottom, ksize=[1, 2, 2, 1],
        strides=[1, 2, 2, 1], padding='SAME', name=name)
#########################################################################
    def conv_layer(self, bottom, name):
        with tf.variable_scope(name):
            filt = self.get_conv_filter(name)

            conv = tf.nn.conv2d(bottom, filt, [1, 1, 1, 1], padding='SAME')

            conv_biases = self.get_bias(name)
            bias = tf.nn.bias_add(conv, conv_biases)

            relu = tf.nn.relu(bias)
            return relu

#########################################################################
    def conv_layer_ELU(self, bottom, name):
        with tf.variable_scope(name):
            filt = self.get_conv_filter(name)

            conv = tf.nn.conv2d(bottom, filt, [1, 1, 1, 1], padding='SAME')
```

```
175
176             conv_biases = self.get_bias(name)
177             bias = tf.nn.bias_add(conv, conv_biases)
178             elu = tf.nn.elu(bias)
179             return elu
180
181 ###################Build fully convolutional Layer####################
182     def fc_layer(self, bottom, name):
183         with tf.variable_scope(name):
184             shape = bottom.get_shape().as_list()
185             dim = 1
186             for d in shape[1:]:
187                 dim *= d
188             x = tf.reshape(bottom, [-1, dim])
189
190             weights = self.get_fc_weight(name)
191             biases = self.get_bias(name)
192
193             # Fully connected layer. Note that the '+' operation
194             automatically
195             # broadcasts the biases.
196             fc = tf.nn.bias_add(tf.matmul(x, weights), biases)
197
198             return fc
199 ########################Get VGG filter ###########################################
200     def get_conv_filter(self, name):
201         var=tf.Variable(self.data_dict[name][0], name="filter_" + name)
202         self.SumWeights+=tf.nn.l2_loss(var)
203         return var
204 ##########################################################################
205     def get_bias(self, name):
206         return tf.Variable(self.data_dict[name][1], name="biases_"+name)
207 ##########################################################################
208     def get_fc_weight(self, name)
209         return tf.Variable(self.data_dict[name][0], name="weights_"+name)
210 ##########################################################################
```

A.3 Checking VGG Model

```
#======================================================================
# CHECK VGG MODEL
#======================================================================
#.....................import libraries.............
import os
#------Check if pretrain vgg16 models and data are available-------------
def CheckVGG16(model_path):   # Check if pretrained vgg16 model available
#and if not try to download it
TensorflowUtils.maybe_download_and_extract(model_path.split('/')[0],
"ftp://mi.eng.cam.ac.uk/pub/mttt2/models/vgg16.npy")
# If not exist try to download  pretrained vgg16 net for network initiation
    if not os.path.isfile(model_path):
       print("Error: Cant find pretrained vgg16 model for network
       initiation. Please download model from:")
       print("ftp://mi.eng.cam.ac.uk/pub/mttt2/models/vgg16.npy")
       print("Or from:")
       print("https://drive.google.com/file/d/0B6njwynsu2hXZWcwX0FKTGJKRWs/view?
       usp=sharing")
       print("and place in the path pointed by model_path")
```

Note: This can be download manually from:"https://drive.google.com/drive/folders/0B6njwynsu2hXcDYwb1hxMW9HMEU" or from ftp://mi.eng.cam.ac.uk/pub/mttt2/models/vgg16.npy; and placed in the $/Model_Zoofolder$ in the code dir

A.4 Training of Fully Convolutional Neural Network for Building Extraction

```
#========================================================================
# TRAIN FULLY CONVOLUTIONAL NEURAL NETWORK FOR BUILDING EXTRACTION
#========================================================================
#.....................Import Libraries.............................
import tensorflow as tf
import numpy as np
import Data_Reader
import BuildNetVgg16
import os
import CheckVGG16Model
import scipy.misc as misc
import time
#.....................configure memory allocation.....................
config = tf.ConfigProto()
config.gpu_options.allocator_type ='BFC'
config.gpu_options.per_process_gpu_memory_fraction = 0.90

#.............................Input and output folders................
Train_Image_Dir="Training␣data/image/"      # Images and labels for
training
Train_Label_Dir="Training␣data/label/"      # Annotation for training
UseValidationSet=True # do you want to use validation set in training
Valid_Image_Dir="validation␣data/image/"  # Validation images that will be
used to evaluate   training
Valid_Labels_Dir="validation␣data/training␣label/"  # annotation for validation
logs_dir= "logs/" # logs directory where trained model and information will be stored
if not os.path.exists(logs_dir): os.makedirs(logs_dir)
model_path="Model_Zoo/vgg16.npy" # Path to pretrained vgg16 model for encoder
learning_rate=5e-5      #Learning rate for Adam Optimizer
CheckVGG16Model.CheckVGG16(model_path)  # Check if pretrained vgg16 model
#available and if not try to download it
#-----------------------------Other Parameters--------------------------------
TrainLossTxtFile=logs_dir+"TrainLoss.txt"       #Where train losses will
#be written
ValidLossTxtFile=logs_dir+"ValidationLoss.txt"      # Where validation
#losses will be written
TrainaccuracyTxtFile=logs_dir+"TrainAccuracy.txt"
# where train accuracies will be written
ValidaccuracyTxtFile=logs_dir+"ValidationAccuracy.txt"
# where validation accuracies will be written
Batch_Size=5            # Number of files per training iteration
Weight_Loss_Rate=1e-5   # Weight for the weight decay loss function
MAX_ITERATION = int(50000)         # Max  number of training iteration
NUM_CLASSES = 2 #Number of class buildings and non-buildings

#.........................Solver for model raining.................
def train(loss_val, var_list):
    optimizer = tf.train.AdamOptimizer(learning_rate)
    grads = optimizer.compute_gradients(loss_val, var_list=var_list)
    return optimizer.apply_gradients(grads)
.........................
def main(argv=None):
    tf.reset_default_graph()
    keep_prob= tf.placeholder(tf.float32,
name="keep_probabilty")       #Dropout probability

```

```
#.....................Placeholders for input image and labels..........
    image = tf.placeholder(tf.float32,
shape=[None, None, None, 3],
name="input_image")
#Input image
    GTLabel = tf.placeholder(tf.int32,
shape=[None, None, None, 1],
name="GTLabel")         #Ground truth labels for training

  #.....................Build FCN Net..............
    Net =  BuildNetVgg16.BUILD_NET_VGG16(
vgg16_npy_path=model_path)    #Create class for the network
    Net.build(image, NUM_CLASSES,keep_prob)
    # Create the net and load initial weights

#.....Get loss functions for neural network one loss
#function for each set of label....
    Loss = tf.reduce_mean((tf.nn.sparse_softmax_cross_entropy_with_logits(
    labels=tf.squeeze(GTLabel, squeeze_dims=[3]),
    logits=Net.Prob,name="Loss")))
    tf.summary.scalar("Loss", Loss)
    # Define loss function for training

#....Get accuracy function for neural network one accuracy function for
each set of Label....
    Prob = tf.reshape(Net.Pred1, [-1,])
    Prob1 = tf.cast(Prob, tf.int32)
    correct_prediction = tf.equal(Prob1, tf.reshape(GTLabel, [-1,]))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    #Define accuracy function for training

   #...................Create solver for the ......................
    trainable_var = tf.trainable_variables()
    # Collect all trainable variables for the net
    train_op = train(Loss, trainable_var)
    #Create Train Operation for the net
    print("setting up summary op...")
    summary_op = tf.summary.merge_all()

#---------------------Create reader for data set------------
    TrainReader = Data_Reader.Data_Reader(Train_Image_Dir,
    GTLabelDir=Train_Label_Dir,BatchSize=Batch_Size)
                #Reader for training data
    if UseValidationSet:
        ValidReader = Data_Reader.Data_Reader(Valid_Image_Dir,
        GTLabelDir=Valid_Labels_Dir,BatchSize=Batch_Size)
                  # Reader for validation
        datasess = tf.Session()          #Start Tensorflow session

# -------------load trained model if exist-------------
    print("Setting up Saver...")
    saver = tf.train.Saver()
    sess.run(tf.global_variables_initializer())
                #Initialize variables
    ckpt = tf.train.get_checkpoint_state(logs_dir)
    if ckpt and ckpt.model_checkpoint_path:
          # if train model exist restore it
        saver.restore(sess, ckpt.model_checkpoint_path)
        print("Model restored...")
```

```
116
117 #--------------------Create files for saving loss and accuracy-------------------------
118     f = open(TrainLossTxtFile, "w")
119     f.write("Iteration\tloss\t␣Learning␣Rate="+str(learning_rate))
120     f.close()
121     if UseValidationSet:
122        f = open(ValidLossTxtFile, "w")
123        f.write("Iteration\tloss\t␣Learning␣Rate=" + str(learning_rate))
124        f.close()
125
126 #..............Start Training loop: Main Training...................
127     for itr in range(MAX_ITERATION):
128         Images, GTLabels =TrainReader.ReadAndAugmentNextBatch()  # Load
129         augmented images and ground true labels for training
130        feed_dict = {image: Images,GTLabel:GTLabels, keep_prob: 0.5}
131         st = time.time()            # set current time
132         sess.run(train_op, feed_dict=feed_dict)        # Train one cycle
133
134 # --------------Save trained model------------------------
135         if itr % 500 == 0 and itr>0:
136             print("Saving␣Model␣to␣file␣in␣"+logs_dir)
137             saver.save(sess, logs_dir + "model.ckpt", itr)   #Save model
138
139 #...................Write and display train loss and train accuracy.............
140         if itr % 10==0:
141             print(time.time()-st)   #time taken by given iteration
142             # Calculate train loss and accuracy
143             TLoss, Taccuracy, summary_str =sess.run([Loss, accuracy,
144             summary_op], feed_dict=feed_dict)
145             print("Step"+str(itr)+"Train␣Loss="+str(TLoss)+
146             "Train␣accuracy="+str(Taccuracy))
147
148             #Write train loss and train accuracy to file
149             with open(TrainLossTxtFile, "a") as f1,
150             open(TrainaccuracyTxtFile, "a") as f2:
151                 f1.write("\n"+str(itr)+"\t"+str(TLoss))
152                 f2.write("\n"+str(itr)+"\t"+str(Taccuracy))
153                 f1.close()
154                 f2.close()
155 #.....Write and display Validation Set Loss by running loss on all validation images ....
156         if UseValidationSet and itr % 500 == 0:
157             SumLoss=np.float64(0.0)
158             SumAccuracy=np.float64(0.0)
159             Sumprecision = np.float64(0.0)
160             Sumrecall = np.float64(0.0)
161             NBatches=np.int(np.ceil(ValidReader.NumFiles/ValidReader.BatchSize))
162             print("Calculating␣Validation␣on" + str(ValidReader.NumFiles) +
163             "Images") # Go over all validation image
164             for i in range(NBatches):
165               Images, GTLabels= ValidReader.ReadNextBatchClean()
166         # load validation image and ground true labels
167                 feed_dict = {image: Images,GTLabel: GTLabels ,keep_prob: 1.0}
168
169                 # Calculate loss for all labels set
170                 TLoss, Taccuracy, summary_str =sess.run([Loss, accuracy,
171                 summary_op], feed_dict=feed_dict)
172                 SumLoss+=TLoss
173                 SumAccuracy+=Taccuracy
174                 NBatches+=1
```

```
175             SumLoss /= NBatches
176             SumAccuracy /=NBatches
177             Sumprecision /= NBatches
178             Sumrecall /= NBatches
179             print("Validation␣Loss:"+str(SumLoss)+
180             "Validation␣Accuracy:"+str(SumAccuracy))
181             with open(ValidLossTxtFile, "a") as f1,
182             open(ValidaccuracyTxtFile, "a") as f2,:
183             f1.write("\n" + str(itr) + "\t" + str(SumLoss))
184             f2.write("\n" + str(itr) + "\t" + str(SumAccuracy))
185             f1.close()
186             f2.close()
187 main()                  #Run script
188 print("Finished")
```

A.5 Prediction and Generate Pixelwise Annotation using FCN

```
#=========================================================================
# PREDICTION AND GENERATION OF PIXELWISE ANNOTATION
#=========================================================================

#..................... import libraries.............
import tensorflow as tf
import numpy as np
import scipy.misc as misc
import sys
import BuildNetVgg16
import TensorflowUtils
import os
import Data_Reader
import CheckVGG16Model
logs_dir= "logs/"
# "path to logs directory where trained model and information will be stored"
Image_Dir="test/image"          # Test image folder
Pred_Dir="Output_Prediction/"
# Library where the output prediction will be written
model_path="Model_Zoo/vgg16.npy"  # "Path to pretrained vgg16 model for encoder"
NUM_CLASSES = 2           # Number of classes
#-------------------------------------------------------------------------------------------------------------------------
CheckVGG16Model.CheckVGG16(model_path)
# Check if pretrained vgg16 model available and if not try to download it

##########################################################################
def main(argv=None):
      #....Placeholders for input image and labels.........
    keep_prob = tf.placeholder(tf.float32, name="keep_probabilty")
          # Dropout probability
    image = tf.placeholder(tf.float32, shape=[None, None, None, 3],
    name="input_image")
    # Input image
    # -------------------------Build Net-------------
    Net = BuildNetVgg16.BUILD_NET_VGG16(
vgg16_npy_path=model_path)
# Create class instance for the net
Net.build(image, NUM_CLASSES, keep_prob)
# Build net and load intial weights (weights before training)
    # -----------Data reader for validation/testing images-----------
    ValidReader = Data_Reader13.Data_Reader(Image_Dir,  BatchSize=1)
    #--------Load Trained model if you dont have trained model see: Train.py------
    sess = tf.Session()           #Start Tensorflow session
    print("Setting up Saver...")
    saver = tf.train.Saver()
    sess.run(tf.global_variables_initializer())
    ckpt = tf.train.get_checkpoint_state(logs_dir)
    if ckpt and ckpt.model_checkpoint_path:
        # if train model exist restore it
        saver.restore(sess, ckpt.model_checkpoint_path)
        print("Model restored...")
    else:
        print("ERROR NO TRAINED MODEL IN:"+
        ckpt.model_checkpoint_path+" See Train.py for creating train network
        ")
        sys.exit()
```

```
57
58  #----------Create output directories for predicted label, one folder for each
59  #---------------granularity of label prediction--------
60      if not os.path.exists(Pred_Dir): os.makedirs(Pred_Dir)
61      if not os.path.exists(Pred_Dir + "/Label55"): os.makedirs(Pred_Dir + "/Label55")
62      print("Running␣Predictions:")
63      print("Saving␣output␣to:" + Pred_Dir)
64   #------Go over all images and predict semantic segmentation in various of classes------
65      fim = 0
66      print("Start␣Predicting␣" + str(ValidReader.NumFiles) + "␣images")
67      while (ValidReader.itr < ValidReader.NumFiles):
68          print(str(fim * 100.0 / ValidReader.NumFiles) + "%")
69          fim += 1
70          # ................................Load image.............
71          FileName=ValidReader.OrderedFiles[ValidReader.itr]    #Get input image name
72          Images = ValidReader.ReadNextBatchClean()        # load testing image
73
74          # Predict annotation using net
75          LabelPred = sess.run(Net.Pred, feed_dict={image: Images, keep_prob: 1.0})
76
77     #-------------------Save predicted labels overlay on images---------------
78
79          misc.imsave(Pred_Dir + "/Label55/" + FileName[:-5] + ".tif" +
80          NameEnd, LabelPred[0].astype(np.uint8))
81
82   ##############################################################################
83  main()                  #Run script
84  print("Finished")
```

A.6 Prediction of Label using Post Processing CRFs

```
#=============================================================================
# PREDICTION USING POST PROCESSING CRFS
#=============================================================================
#.............................. import libraries.....................
import tensorflow as tf
import numpy as np
import scipy.misc as misc
import sys
import BuildNetVgg16
import TensorflowUtils
import os
import Data_Reader
import CheckVGG16Model
import pydensecrf.densecrf as dcrf
from pydensecrf.utils import compute_unary, create_pairwise_bilateral, \
    create_pairwise_gaussian, softmax_to_unary

logs_dir= "logs/"
# "path to logs directory where trained model and information will be stored"
Image_Dir="validation␣data/image/"        # Test image folder
w=0.6# weight of overlay on image
Pred_Dir="Output_Prediction/"
    # Library where the output prediction will be written
model_path="Model_Zoo/vgg16.npy"
    # "Path to pretrained vgg16 model for encoder"
NUM_CLASSES = 2
    # Number of classes
#-------------------------------------------------------
CheckVGG16Model.CheckVGG16(model_path)
  # Check if pretrained vgg16 model available and if not try to download it
#############################################################################
def main(argv=None):
      # .............Placeholders for input image and labels.............
    keep_prob = tf.placeholder(tf.float32, name="keep_probabilty")
    # Dropout probability
    image = tf.placeholder(tf.float32, shape=[None, None, None, 3],
name="input_image")   # Input image

    # -------------------------Build Net-----------------------
    Net = BuildNetVgg16.BUILD_NET_VGG16(
vgg16_npy_path=model_path)    # Create class instance for the net
Net.build(image, NUM_CLASSES, keep_prob)
# Build net and load initial weights (weights before training)
    # -------------------Data reader for validation/testing images------
    ValidReader = Data_Reader13.Data_Reader(Image_Dir,  BatchSize=1)
    #-----Load Trained model if you dont have trained model see: Train.py------

    sess = tf.Session()           #Start Tensorflow session
    print("Setting␣up␣Saver...")
    saver = tf.train.Saver()

    sess.run(tf.global_variables_initializer())
    ckpt = tf.train.get_checkpoint_state(logs_dir)
    if ckpt and ckpt.model_checkpoint_path:
    # if train model exist restore it
        saver.restore(sess, ckpt.model_checkpoint_path)
```

```
        print("Model restored...")
    else:
        print("ERROR NO TRAINED MODEL IN: "+ckpt.model_checkpoint_path+" See
        Train.py for creating train network ")
        sys.exit()

#------Create output directories for predicted label, one folder for each
#-------------granularity of label prediction-------
    if not os.path.exists(Pred_Dir): os.makedirs(Pred_Dir)
    if not os.path.exists(Pred_Dir + "/Labelcrf"): os.makedirs(Pred_Dir +
    "/Labelcrf")
    print("Running Predictions:")
    print("Saving output to:" + Pred_Dir)
 #-----Go over all images and predict semantic segmentation in various of classes----
    fim = 0
    print("Start Predicting " + str(ValidReader.NumFiles) + " images")
    while (ValidReader.itr < ValidReader.NumFiles):
        print(str(fim * 100.0 / ValidReader.NumFiles) + "%")
        fim += 1
        # .................................Load image.............
        FileName=ValidReader.OrderedFiles[ValidReader.itr]  #Get input image name
        Images = ValidReader.ReadNextBatchClean()     # load testing image

        # ....Predict probabilities using net.................
        Probabilities = sess.run(Net.probabilities,
          feed_dict={image: Images, keep_prob: 1.0})
        softmax = Probabilities.squeeze()
        softmax = softmax.transpose((2, 0, 1))  # change the order having class at first

       #.....................................refine using crfs.........

       # The input should be the negative of logarithm of probabilities value
        unary = softmax_to_unary(softmax)
        #check softmax_to_unary for further information
        #changing input to c-contiguous
        unary = np.ascontiguousarray(unary)
       d = dcrf.DenseCRF(Images.shape[1]*Images.shape[2], 2)
        d.setUnaryEnergy(unary)

        # This potential penalizes small pieces of segmentation that are
        #spatially located
        feats = create_pairwise_gaussian(sdims=(3, 3), shape=Images.shape[1:3])
        d.addPairwiseEnergy(feats, compat=2,
                    kernel=dcrf.DIAG_KERNEL,
                    normalization=dcrf.NORMALIZE_SYMMETRIC)

        #This creates the color dependent features to refine features
        feats = create_pairwise_bilateral(sdims=(2, 2), schan=(10,10,10),
                                  img=Images, chdim=3)
        d.addPairwiseEnergy(feats, compat=20,
                     kernel=dcrf.DIAG_KERNEL,
                     normalization=dcrf.NORMALIZE_SYMMETRIC)
        Q = d.inference(10)         # inference with iterations
        res = np.argmax(Q, axis=0).reshape((Images.shape[1],
        Images.shape[2]))
             #--------------Save predicted labels overlay on images------------

        misc.imsave(Pred_Dir + "/Labelcrf35-8/" + FileName[:-5] + ".tif"
        +NameEnd, res.astype(np.uint8))
```

```
116
117 ###############################################################################
118 main()                  #Run script
119 print("Finished")
```

A.7 Accuracy Assessment

```
#=============================================================================
# TRUE POSITIVE FOR RELAX PRECISION AND RECALL
#=============================================================================

#.......................... import libraries.....................
import os
import sys
import numpy as np
from PIL import Image

#....
...... function to calculate true positive for relax precision..........
def relaxprecision(original, classified, relax):
    imoriginal = Image.open(original)
    # open ground truth label image
    imclassified = Image.open(classified)
    # open classified label image
    Pixoriginal = list(imoriginal.getdata())
    # prepare list of label from ground truth image
    Pixclassified = list(imclassified.getdata())
    # prepare list of label from classified image

    w_lim, h_lim = imclassified.size
    # size of classified image
    truepositive = 0
    for i in range(0, h_lim, 1):
        for j in range(0, w_lim, 1):
            prediction_val = Pixclassified[i*w_lim+j]
            if prediction_val == 1:
                left_top = i - relax
                if left_top <= 0:
                    left_top = 0
                right_top = i + relax
                if right_top > h_lim:
                    right_top = h_lim-1
                left_down = j - relax
                if left_down <= 0:
                    left_down =
                right_down = j + relax
                if right_down > w_lim:
                    right_down = w_lim-1
                sum = 0
                for ii in range(left_top, right_top, 1):
                    for jj in range(left_down, right_down,1):
                        sum+= Pixoriginal[ii*w_lim+jj]
                if sum > 0:
                    truepositive+=1
    return truepositive
    get the value of true positive

# .............function to calculate true positive for relax recall........
def relaxrecall(original, classified, relax):
    imoriginal = Image.open(original)
    # open ground truth label image
    imclassified = Image.open(classified)
    # open classified label image
```

```
    Pixoriginal = list(imoriginal.getdata())
    # prepare list of label from ground truth image
    Pixclassified = list(imclassified.getdata()
    # prepare list of label from classified image
    w_lim, h_lim = imoriginal.size  # size of classified image
    truepositive = 0
    for i in range(0, h_lim, 1):
        for j in range(0, w_lim, 1):
            label_val = Pixoriginal[i*w_lim+j]
            if label_val == 1:
                left_top = i - relax
                if left_top <= 0:
                    left_top = 0
                right_top = i + relax
                if right_top > h_lim:
                    right_top = h_lim-1

                left_down = j - relax
                if left_down <= 0:
                    left_down = 0
                right_down = j + relax
                if right_down > w_lim:
                    right_down = w_lim-1
             sum = 0
                for ii in range(left_top, right_top, 1):
                    for jj in range(left_down, right_down,1):
                        sum+= Pixclassified[ii*w_lim+jj]
                if sum > 0:
                    truepositive+=1
    return truepositive         # get the value of true positive
#=============================================================================

# CALCULATION OF RELAXED PRECISION, RECALL, F1MEASURE AND IOU
#=============================================================================

#.............................import libraries.....................
import scipy.misc as misc
import numpy as np
import os
from PIL import Image
import rel_pre_re
relax = 3

#.............................. input images and labels...........
Image_dir = "Output_Prediction/Label"
# directory of output test prediction
Label_dir = "test/label"          # directory of input test label
file = []
file += [each for each in os.listdir(Image_dir)
# create list of prediction label
if each.endswith('.PNG') or each.endswith('.JPG') or
each.endswith('.TIF') or each.endswith('.GIF') or
each.endswith('.png') or each.endswith('.jpg') or
 each.endswith('.tiff') or each.endswith('.tif') ]
#..................... calculation of performance metrics............
Precision = []            # empty list for precision value
Recall = []           # empty list for recall value
F1measure = []            # empty list for f1measure value
IOU = []              # empty list for IOU value
```

```
for f in file:
    image = Image_dir + "/" + f
    # individual predicted image in list
    classified = Image.open(image)
    labelname = f[0:-4]+ ".tif"   # corresponding ground truth label name
    label1 = Label_dir + "/" + labelname
    label = Image.open(label1)
    # corresponding ground truth label
    prediction = np.asarray(classified, dtype=np.int32)
    # array of predicted image
    label = np.asarray(label, dtype=np.int32)
    # array of ground truth label
    positive = np.sum(prediction==1)
    true = np.sum(label==1)
    precision_tp = rel_pre_re.relaxprecision(label1, image, relax)
    # true positive for relax precision
    recall_tp = rel_pre_re.relaxrecall(label1, image, relax)
    # true positive for relax recall

    if precision_tp > positive or recall_tp > true:
        print(positive, precision_tp, true, recall_tp)
        sys.exit('calculation is wrong.')

    precision = precision_tp/float(positive)
    # relax precision for each test image
    recall = recall_tp/float(true)
    # relax recall for each test image
    f1measure = 2*precision*recall/(precision + recall)
    # relax f1measure for each test image
    iou = precision*recall/(precision+recall-(precision*recall))
    # relax IOU for each test image
    print(labelname)
    print(precision, recall, f1measure, iou)

    Precision.append(precision)
    Recall.append(recall)
    F1measure.append(f1measure)
    IOU.append(iou)

meanprecision = sum(Precision)/len(Precision)
# mean relax precision for all test image
meanrecall = sum(Recall)/len(Recall)
# mean relax recall for all test image
meanF1measure = sum(F1measure)/len(F1measure)
#mean relax F1measure for all test image
meanIOU = sum(IOU)/len(IOU)        # mean relax IOU for all test image

print("meanprecision:", meanprecision, " meanrecall:", meanrecall,
"meanF1:", meanF1measure, "meanIOU:", meanIOU)
```

Masters Program in Geospatial Technologies

IMPROVED FULLY CONVOLUTIONAL NETWORK WITH CONDITIONAL RANDOM FIELD FOR BUILDING EXTRACTION

Sanjeevan Shrestha

Dissertation submitted in partial fulfilment of the requirements
for the Degree of *Master of Science in Geospatial Technologies*

NOVA IMS Information Management School
Universitat Jaume I
Westfälische Wilhelms-Universität Münster

Masters Program in **Geospatial Technologies**

Supported by:
Education and Culture
ERASMUS MUNDUS