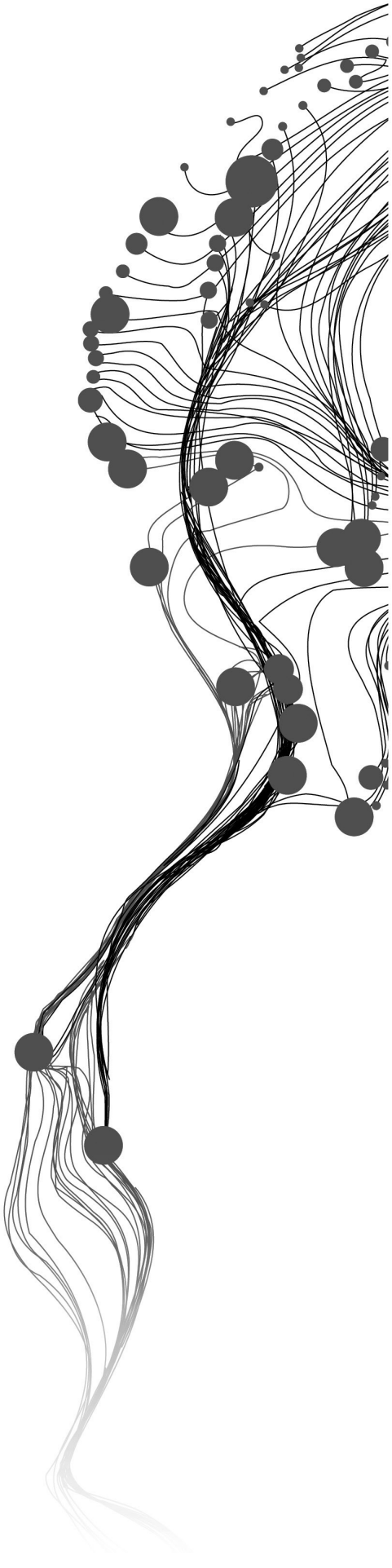
A DEEP FEATURE LEARNING APPROACH TO URBAN SCENE CLASSIFICATION

JOHN RAY A. BERGADO
March, 2016

SUPERVISORS:

Dr. C. Persello

Dr. V.A. Tolpekin
Advisor: Ms. C. M. Gevaert MSc



A DEEP FEATURE LEARNING APPROACH TO URBAN SCENE CLASSIFICATION

JOHN RAY A. BERGADO
Enschede, The Netherlands, March, 2016

Thesis submitted to the Faculty of Geo-information Science and Earth Observation of the University of Twente in partial fulfilment of the requirements for the degree of Master of Science in Geo-information Science and Earth Observation.
Specialization: Geoinformatics

SUPERVISORS:

Dr. C. Persello

Dr. V.A. Tolpekin
Advisor: Ms. C. M. Gevaert MSc

THESIS ASSESSMENT BOARD:

Prof. Dr. Ir. M.G. Vosselman (chair)

Ms. dr. B. Demir (external examiner, University of Trento)

Disclaimer

This document describes work undertaken as part of a programme of study at the Faculty of Geo-information Science and Earth Observation of the University of Twente. All views and opinions expressed therein remain the sole responsibility of the author, and do not necessarily represent those of the Faculty.

ABSTRACT

Spatial-contextual features play a vital role in the classification of very high resolution aerial images characterized by sub-decimeter resolution. However, manually extracting discriminative features is difficult and time-consuming, especially when dealing with classification problems where the objects of interest are considerably larger than the pixel size. Deep feature learning methods enable us to replace handcrafted features by automatically learning informative features from the data itself—allowing us to skip the tedious and inefficient step of manual feature extraction.
This thesis aims to design, analyze, and evaluate a deep feature learning approach to the classification of very high resolution aerial images. We carried out our experiments using sub-decimeter resolution aerial images and corresponding digital surface models (DSM) acquired in Vaihingen, Germany (ISPRS 2D semantic labeling benchmark dataset (Cramer, 2010)). The sensitivity of the classifier to several of its hyperparameters was investigated. From the knowledge obtained in the experimental analysis performed, a convolutional neural network architecture was chosen to be the core algorithm of our classification approach. We evaluate the convolutional neural network based approach against 2 other approaches: 1) using individual pixel values (1 near-infrared, 2 optical bands, and DSM), and 2) using additional handcrafted spatial-contextual features. Several performance metrics were used such as the: overall classification accuracy, statistical difference in accuracy, class F1-scores, accuracy gained from additional training samples, map quality, and computational time and complexity. Aside from these 6 metrics, the domain adaptability of the classifiers were assessed by measuring their performance on image tiles were no training samples are taken.
Experimental results show that the convolutional neural network based approach outperforms the 2 other classification approaches in all the performance metrics considered, except for computational time and complexity. This illustrates that directly learning relevant spatial-contextual features from the data can help improve the classification of very high resolution aerial images. Adopting such an approach for large-scale projects of automated classification of very high resolution images can benefit both the quality of the results and the streamlining of the classification pipeline. We leave the further investigation of recurrent convolutional neural networks and unsupervised pre-training for future research works.

Keywords

deep learning, convolutional neural networks, urban classification, aerial images

ACKNOWLEDGEMENTS

Thanks be to God. For He deserves all the glory and honor I can express. I also thank Him for all the medium and instruments He have used along the way.
I thank the Erasmus Mundus Mobility with Asia (EMMA-WEST 2013-2016) consortium for funding my study. I hope God will continue to bless your good works.
I thank the whole ITC community—staff and fellow students—for a very valuable learning experience. Especially, I would like to thank my supervisors Claudio and Valentyn, and my advisor Caroline for guiding and mentoring me through the whole research work. For steering and pushing me to the right direction when my scientific juices seem to run out. I really learned a lot from all of you and appreciate your every help.
I also thank all my friends who never ceases to be one despite my highly hermitical lifestyle. I hope to catch up with all of you soon, some time...somewhere.
And of course, I would like to thank my immediate (Pa, Ma, Imbo, Ame, Yeye) and extended (relatives, JCHW and ICF community) family for all the prayers, unconditional love and support. Thank you all! Truly, we can do all things through Christ who strengthens us.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

1.1 MOTIVATION AND PROBLEM STATEMENT

1.1.1 Urban land cover classification

Fast-paced and poorly-planned urban growth in a large number of areas around the world poses different local problems: from environmental degradation to unsustainable economy (UNPD, 2014). Such problems, caused by rapid urban development, require responsive plans and decisions from environmental planners, policy makers, and local government units. Up to date information about the urban environment is necessary for the timely development and implementation of these relevant plans and policies.

Land cover information is changing with the continuing trend in urbanization (Dewan and Yamaguchi, 2009). Remotely sensed data, in the form of satellite images and aerial photos, can provide repeatable observations of the urban environment; thus offering an important source to derive updated land cover information. After acquiring these data, a classification step is required to map these observations to land cover classes of interest to a specific application. Thus, the timeliness of land cover information depends on two sets of factors. Firstly, it will be affected by the characteristics—e.g. repeatability/revisit time—of the platform acquiring the image. And secondly, by the time required to perform the classification from end to end, including any necessary pre- or post-processing step. Ideally, platforms with shorter revisit time or higher repeatability—e.g. satellites with shorter revisit time and unmanned aerial vehicles (UAV)—and automated classification methods lead to better up-to-date land cover information.

According to Ju et al. (2005), spatial resolution plays a key role in land cover classification of remotely sensed data. This choice generally depends on the application of interest; the application being defined in terms of land cover categories in a classification problem. Several applications relevant to sound urban development such as: identification of individual trees for ecosystem management (Verheyden et al., 2002), road detection for strategic city planning (Mnih and Hinton, 2010), building detection for population estimation (Deichmann et al., 2011), and vehicle detection for traffic monitoring (Gleason et al., 2011), can only be achieved by using images with spatial resolution sufficient to identify these objects of interests—individual trees, roads, buildings, vehicles, etc. Even though manual classification of very high resolution images is generally easier compared to lower resolution ones, doing so is often not practical—especially for large extent (large-scale) mapping and/or monitoring. Moreover, photointerpretation is subjective and thus the classification of the same area by different photo-interpreters might be inconsistent. Therefore the need for timely land cover information demands the use of automated classification methods for such applications.

1.1.2 Contextual image classification

Several automated image classification approaches in remote sensing, briefly discussed by Warner et al. (2009, pp. 269–281) and Richards (2013, pp. 247–317), already exist such as the well-known

pixel-based maximum likelihood classifier (MLC), object-oriented classifiers (OOC), and other machine learning methods. The problem with MLC, being a parametric classifier, is it can give inadequate results when the image to be classified does not meet specific assumptions—i.e. normality of the data distribution, which is usually not applicable when defining land cover classes of very high resolution images. A study (Ke et al., 2010) using 5-meter resolution RapidEye satellite images and another study (Adamczyk and Osberger, 2015) using 0.6-meter and 2.44-meter resolution QuickBird satellite images have both observed this non-normality in training data. OOC, on the other hand, usually consists of an image segmentation step before classification and thus is subjected to the performance of the segmentation algorithm used. Additionally, feature extraction and feature selection steps are usually done before the classification. Feature extraction generates additional features out of the original ones to help discriminate the classes. Feature selection, on the other hand, reduces the number of features to those features that are most relevant. Having less features generally improves the computational time and the generalization of a classifier. Feature selection is also found to improve accuracy in image classification (Kumar et al., 2005; Somers and Asner, 2013).

Feature extraction is often used to derive features with contextual information. Such contextual information is relevant, especially for very high resolution aerial images to compensate for the latter's limited spectral resolution. Since the restricted number of original bands of the image is, in most cases, insufficient to discriminate the land cover classes of interest. This brings in an additional group of spatial-contextual classifiers (Li et al., 2014) with 3 major categories: texture extraction, Markov Random Field (MRF) based models, and segmentation-based OOC. All of these methods of automated image classification can be used for land cover classification of very high resolution urban scenes. However, most of these existing methods—including other non-parametric machine learning methods—rely on handcrafted features (Pacifici et al., 2009; Posner et al., 2009). Such handcrafted features obtained from a feature extraction and selection step offers intuitive understanding of features helpful for a specific problem. Although, handcrafting these features is usually a tedious—up to some extent, with trial and error involved—and time-consuming task.

Textural features are found to be helpful for image classification (Pacifici et al., 2009). Typically, textural features are extracted by computing statistical parameters of the grey level co-occurrence matrix (GLCM) developed by Haralick et al. (1973). The computation is done by applying a moving window filter to the image. This computation will require to tune parameters such as the window size and lag/offset distance. However, for images with higher spatial resolution, the distance between related pixels increases. This will require a larger window for the computation and thus will demand more computing power. At the same time, increasing the spatial resolution also drastically increases the range of possible values of parameters required to tune methods for extracting textural features. In this case, such parameter tuning will therefore be impractical and computationally infeasible. Likewise, MRF-based models will require larger neighborhood structures and set of parameters to tune; and therefore will likely have the same problems as the texture extraction approach.

1.1.3 Deep feature learning

In this thesis, a deep feature learning approach is adopted because of its following characteristics:

- Based on a nonparametric family of classifiers called Artificial Neural Networks (ANN), discussed in Du and Swamy (2014), deep learning methods can model nonparametric class probability distributions that may be present in very high resolution urban scenes.

- In contrast to other classification approaches, it integrates feature representation in the learning process and thus does not rely on handcrafted features (Deng, 2014). This eliminates the tedious step of manually engineering features.
- It benefits from learning from large datasets with training time and hardware as primary constraints (Krizhevsky et al., 2012).

Deep learning roots can be traced from artificial neural network research; the Multilayer Perceptron (MLP), a feedforward ANN, with more than one hidden layer being an example of a deep architecture. The weights of these networks are usually learned by applying the well-known backpropagation algorithm (Rumelhart et al., 1986). However, this original backpropagation algorithm, popularized in the 1980's, didn't perform well enough in practice for networks with a small number of hidden layers (Deng, 2014). Around 2006, several clever ideas on initialization/pretraining (Bengio et al., 2007; Hinton et al., 2006a), training (Bengio et al., 2007; Hinton et al., 2006b), and understanding (Glorot and Bengio, 2010) this problem of deep neural networks gave rise to the establishment (some may say revival) of a new machine learning paradigm: deep learning. This new machine learning paradigm found early applications on diverse problems—from modeling human motion data (Taylor et al., 2007) to unsupervised learning of hierarchical features for object recognition tasks (Ranzato et al., 2007).
Recent studies (Farabet et al., 2013; Pinheiro and Collobert, 2014) have demonstrated the feasibility of deep feature learning approaches for general image labeling tasks. In contrast to conventional methods in image classification, deep feature learning methods do not rely on handcrafted features. On the contrary, deep learning methods automatically learn/select the discriminative features they find appropriate for a classification task. Moreover, the works of Lee et al. (2009) and Farabet et al. (2013) showed such methods can even learn hierarchical representations of features, e.g. from parts of objects to objects and from objects to scenes.
Further recent studies show potential of deep learning to several remote sensing applications—from being a motivation of a meta-algorithm for hierarchical feature selection (Tuia et al., 2015) to different image classification (Chen et al., 2014b; Song et al., 2013) and object detection (Chen et al., 2014a; Tang et al., 2015) tasks. Deep convolutional neural networks were mostly used for object detection problems (Chen et al., 2013, 2014a; Maire et al., 2014; Saito and Aoki, 2015; Wang et al., 2015). Deep Boltzmann machines were exploited for feature learning tasks (Han et al., 2015; Yu et al., 2015) while a Deep Belief Network was used to classify land cover from radar data (Qi et al., 2014). Stacked (Denoising) Autoencoders were also used for both image classification (Chen et al., 2014b) and object detection (Tang et al., 2015) problems as well.

1.1.4 Summary and problem with difficult data

On a broader perspective, the problem of classifying very high resolution aerial images of urban areas can be associated to 2 of the 3 dimensions of the difficult data era (some prefer the term "big data"): volume, velocity, and variety (Casado and Younas, 2015). Firstly, dealing with data volume becomes more problematic with the use of higher resolution images. Secondly, the data velocity problem is amplified with the use of platforms offering highly repeatable observations such as UAV.
As a synopsis, 3 key issues can be identified in classification of very high resolution urban scenes:

- modeling nonparametric class probability distribution of land cover expected to be present in such scenes (not just a specific distributions as in the case of parametric classifiers)
- streamlining of the classification pipeline, possibly eliminating inefficient steps and automating every component

- and learning from difficult, high-volume, high-velocity data source

This research aims to develop a deep feature learning approach addressing these problems, and possibly other specific problems that may be encountered along the development of such an approach, in urban land cover classification of very high resolution aerial images.

1.2 RESEARCH IDENTIFICATION

This research revolves around the design, analysis, and evaluation of a deep feature learning approach to the classification of very high resolution aerial images of urban areas. Specifically, scenes taken from airborne platforms capable of capturing images with sub-decimeter spatial resolution is used. The development of such an approach shall be able to address, directly or indirectly, each key issue identified in the previous subsection 1.1.4: non-parametric class probability distribution, automation of classification pipeline, and learning from difficult data source. Moreover, we analyze the different aspects—e.g. architecture and performance measures—of the approach to come up with a theoretically and practically better classifier. In addition, we compare and evaluate it against alternative classification methods—one of which is using handcrafted features.

1.2.1 Research objectives

This research primarily aims to formulate a method of classifying very high resolution aerial images of urban areas by exploiting state-of-the-art deep learning algorithms. The method is expected to be adaptable as one component for a large-scale urban land cover mapping and/or monitoring system. These systems are useful for different users and use cases: from land use planning and enforcement by a city to urban slum mapping by non-governmental organizations. In addition to this main objective, a number of sub-objectives emerge:

1. To review and evaluate the potential of state-of-the-art deep learning algorithms to the classification of very high resolution aerial images of urban areas.

2. To design, implement, and analyze the performance of a working streamlined classifier based on the chosen deep learning algorithm.

3. To compare the performance of the formulated deep learning classifier against alternative classification methods using handcrafted features, e.g. GLCM.

1.2.2 Research questions

From the objectives above, the following research questions are considered:

Questions related to sub-objective 1

1. How does deep learning algorithms (e.g. convolutional neural networks, autoencoders, and Boltzmann machine variants) work in a remote sensing context?

2. Which deep learning algorithm is suited to classify very high resolution airborne images of urban areas with sub-decimeter resolution?

Questions related to sub-objective 2

1. What are the effects of varying the network architecture (e.g. feed-forward, recurrent) and dimensions (e.g. number of hidden layers, number of neurons in the hidden layers) to the performance of the classifier?
2. What are the effects of initialization and regularization (e.g. dropout, early stopping, recurrence) techniques to the performance of the classifier?
3. What performance measures (e.g. classification accuracy, computational complexity, level of automation, training sample size) are relevant for assessing the classifier?
4. Can the classifier generalize well within a domain adaptation setting, where training and test samples are taken from different images but with similar characteristics?

Questions related to sub-objective 3

1. Which approach performs better and in which aspect of performance measure?
2. How much does the performance of the feature learning and feature engineering approaches differ?

1.2.3 Innovation

Only limited research has been done with deep feature learning in a remote sensing context as most of its popular applications are found in problems with more generic context such as vision and speech information processing in computer vision and natural language processing domains. Moreover, there are no publications found specifically dealing with multiclass classification of very high resolution aerial images of urban areas using deep learning. Some deep learning algorithm-specific details, such as network architecture/dimension variation, were also not thoroughly investigated in remote sensing problems. In this work, interesting initial results were found from an implementation of recurrent convolutional neural network—an architecture that, as far as our knowledge of the literature, has never been investigated in a remote sensing context.
This research exploits the advantages of deep learning to prototype a streamlined classifier for very high resolution aerial images of urban areas. Ideally, the classifier will be able to classify very high resolution images of urban areas end-to-end in a fully automated manner: from very high resolution conventional photogrammetric products—multispectral orthophoto and/or digital surface/terrain model (DSM/DTM)—to land cover maps. This prototype has great potential to be used for large-scale mapping and/or monitoring systems utilizing platforms that can provide repeatable observations such as UAV. Similar attempts to create such fully automated systems exist in the literature. Although, they either only use high resolution (2.5 meter and 10 meter) images (Kemper et al., 2015)—and therefore limits the land cover classes to objects identifiable in that resolution—depend on handcrafted features (Gressin et al., 2014) or both (Li and Narayanan, 2004). Better classification results—in terms of performance measures to be explored in the study—are also expected from the classifier. Thus, its performance was evaluated against other classification methods using handcrafted features.

1.3 PROJECT SETUP

This research project will be divided into 3 stages:

1. review and evaluation

2. design, implementation, and analysis
3. performance comparison

1.3.1 Method adopted

In the first stage of the research project, we reviewed a number of existing deep learning algorithms. The focus of the review is to discuss the applicability of these algorithms to the problem of classifying very high resolution aerial images of urban areas. We solely based our judgment on similar works in existing literature and benchmark data/competition results. At the end of the review, we chose an algorithm used in the following stages of the research project.
The second stage dealt with the design, implementation, and performance analysis of a working streamlined classifier based on the chosen deep learning algorithm. Several experimental designs were analyzed to understand the effects of these design variations—such as changing the network dimensions/architecture, and applying initialization, regularization techniques. The sensitivity of the accuracy of the classifier to these variations were studied. We construct the classifier based on the knowledge gained from understanding the effects of these design variations.
Finally, we compared the performance of the developed deep learning approach and alternative classification approaches (with one using handcrafted features) in the third stage. Since the research project is not focused on the investigation of alternative approaches, typical features (Bekkari et al., 2011) existing in the literature were applied, specifically we used textural features from the gray level co-occurrence matrix (Haralick et al., 1973). Several metrics discussed in the following chapters were used to measure the performance of each classification approach. Figure 1.1 shows a flowchart describing the general idea behind our methodology.

1.3.2 Thesis structure

The thesis is divided in 6 chapters. This chapter introduces the motivation, research problems, questions, and objectives considered in this work. Chapter 2 reviews several deep learning algorithms and their potential for the classification of aerial images of urban areas. The end of the 2^{nd} chapter presents the chosen algorithm to be implemented as the classifier. Chapter 3 tackles the design decisions and implementation details involved in tuning the classifier. Chapter 4 explains how the performance of the tuned classifier was compared against alternative classification approaches. Chapter 5 summarizes the findings from the implementation, performance analysis and comparison experiments conducted. Finally, the thesis concludes in chapter 6 discussing insights gained from, and possible extension of, the work done.

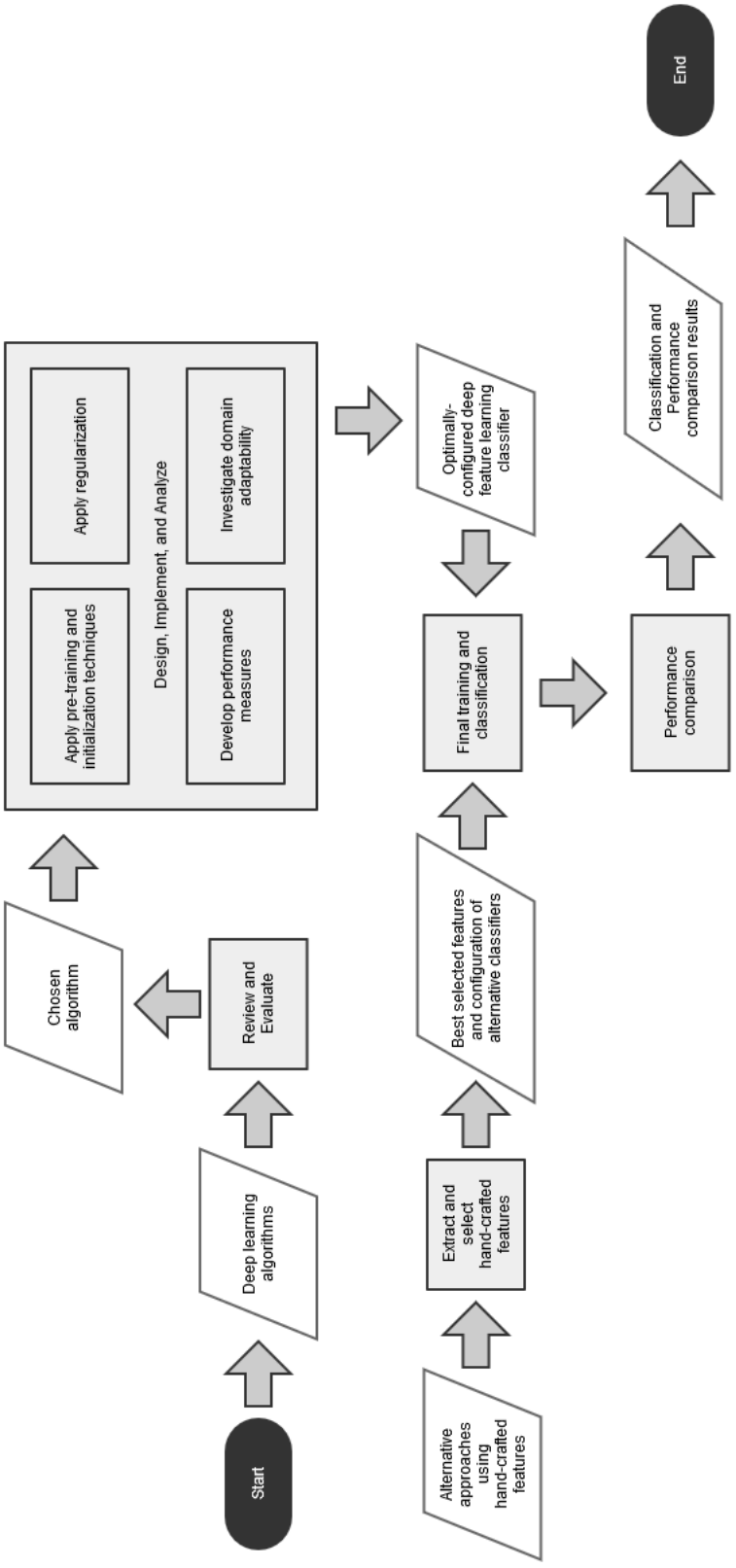


Figure 1.1: Flowchart of the general methodology

Chapter 2
Algorithm Selection

This chapter aims to present a brief review of existing deep learning algorithms while attempting to relate them to the context of image classification in remote sensing: in the end, presenting the algorithm to be implemented as the classifier. The first section discusses the basic idea of the most well-known deep learning algorithms and mentions other algorithms in brief. The first section also cites some benchmark results relevant to to this study. Finally, the last section presents the chosen deep learning algorithm to be used in classifying very high resolution aerial images of urban areas.

2.1 A CONCISE REVIEW OF DEEP LEARNING ALGORITHMS

A number of deep learning algorithms exist in the literature. This section reviews four of the most well-known deep learning algorithms: deep feedforward multilayer perceptrons (MLP), convolutional neural networks (CNN), restricted Boltzmann machines (RBM), autoencoders (AE), and some of their variants. We also refer to other noteworthy algorithms but in much less detail. The review ends with presenting relevant published and unpublished benchmark results in general image labeling—where a class is assigned to the whole image; and parsing of scenes from aerial images—where a class is assigned to each pixels comprising the image—utilizing deep learning algorithms.

2.1.1 Artificial neural networks

The biological brain of animals inspired the development of a group of statistical learning models called artificial neural networks. Analogous to the brain structure, these networks employs interconnected computational units (called artificial neurons) characterized by: their architecture (i.e. how the artificial neurons are organized), the operation each neuron performs, and the learning rules governing them (Du and Swamy, 2014, pp. 9–12). Synapses—connecting neurons in the biological brain—exhibit a manner of plasticity: i.e. their connection strength changes (sometimes vanishes or forms a new one) in response to different patterns of stimulation. Such plasticity of the synaptic connections drives the underlying processes on how learning and memory works inside the brain (Du and Swamy, 2014, pp. 1–5). Artificial neural networks work in a similar way: learning happens as the network changes the weights (parameters) of the connections between its neurons. We can think of an artificial neuron as a computational unit performing an affine transformation (linear combination multiplied by the weights of the connection)

$$a_j = W_{0,j} + \sum_{i=1}^{I} z_{i,j-1} W_{i,j} \tag{2.1}$$

of units with incoming connection, followed by a (usually) non-linear operation, for example the sigmoid

$$z_j = \frac{1}{1 + \exp(-a_j)} \tag{2.2}$$

and the hyperbolic tangent

$$z_j = \tanh(a_j) \tag{2.3}$$

functions; where a_j is a pre-synaptic activation (before applying non-linearity) of a neuron in the j^{th} layer with I number of connections from the preceding $j-1^{th}$ layer, z_j is the post-synaptic activation (after applying non-linearity), $W_{0,j}$ is the weight of the bias unit, and $W_{i,j}$ are the weights of the other connections. Figure 2.1 shows a graphical representation of the operations performed by a neuron where: $x_1, x_2, ...x_n$ are the units, having weights of $w_1, w_2, ...w_n$, with incoming connection; b_0 is the bias unit (generally set to 1); "Σ" sums the product of the incoming units with their weights; and "$\curvearrowright$" is the non-linear operation applied.

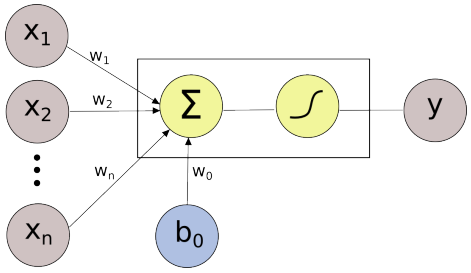


Figure 2.1: Diagram of an artificial neuron

2.1.2 Deep multilayer perceptrons

In a fully-connected feedforward multilayer perceptron model, these neurons are organized in different layers: input, hidden, and output layers. Each neuron connects to all other neurons—hence the term fully-connected—in the preceding (except for those in the input layer) and succeeding layers while being disconnected to neurons in the same layer (see Figure 2.2). The connections are directed—hence the term feedforward—in a way that units in a succeeding layer are fed by the outputs of the units in the preceding layer. Goodfellow et al. (2016, pp. 186–191) summarize functions used in the non-linear operations performed by the neurons. Two of the most common functions used in an MLP are the sigmoid (see equation 2.2) and hyperbolic tangent functions (see equation 2.3).

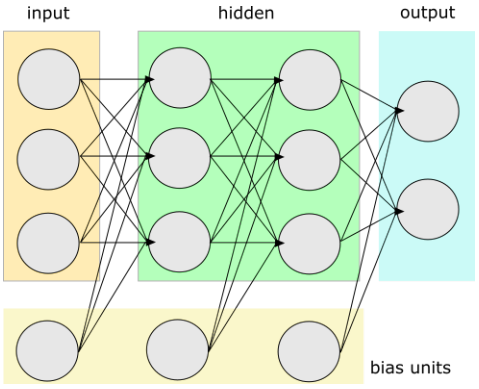


Figure 2.2: Simplified structure of an MLP

We train the network by minimizing a cost/lost function, which is a function of the weights of the network. Changing the weights of the connection between the neurons should then steer the network to decrease the value of the cost function. In a supervised learning setting, the training

phase involves showing a set of labeled samples (training set) to the network and updating the latter's weight such that the loss function is minimized. One possible choice of a loss function (e.g. for binary classification problems) is the Bernoulli negative log-likelihood (equation 2.4) function

$$E(w) = -\sum_{n=1}^{N} \{t_n \log(y_n) + (1 - t_n) \log(1 - y_n)\} \tag{2.4}$$

where t_n is the desired output and y_n is the network's prediction for sample n (Bishop, 2006, pp. 232–236). Learning the weights of the network to minimize the loss function occurs in two stages: firstly, we compute the partial derivatives (gradients) of the loss function with respect to the weights; and secondly, we adjust the weights using these gradients. Rumelhart et al. (1986) popularized an efficient way of doing the first stage by propagating the errors from the output layer back (hence the term backpropagation) to the preceding hidden layers. In the original backpropagation algorithm, they used a simple gradient descent: where the weights are adjusted by iteratively applying small proportions of the negative gradient. See Bishop (2006, pp. 245–246) for a simple example of propagating error using backpropagation.

We can also use the same loss function we used in training to evaluate how the network performs on an unseen test data (we call this the testing phase). For classification problems, we often count the proportion of correct predictions (often called overall accuracy) as a single measure of performance. Furthermore, we can investigate the confusion matrix to see these prediction accuracy measures for each of the classes.

The learning rate η defines the proportion of the negative gradient to be applied in adjusting the weight. Two additional terms, defined by momentum and proportional factors α and γ, can further improve the learning process (Du and Swamy, 2014, pp. 85–90). The momentum term effectively smooths the steps taken by gradient descent while speeding up convergence of the iteration. This three-term weight update ΔW applied at each epoch τ,

$$\Delta W(\tau) = -\eta \frac{\partial E(\tau)}{\partial W(\tau)} + \alpha \Delta W(\tau - 1) + \gamma E(W(\tau))\mathbf{1} \tag{2.5}$$

where $\frac{\partial E(\tau)}{\partial W(\tau)}$ are the gradients computed by backpropagation and $\mathbf{1}$ is a matrix of ones with the same size as the weight matrix W, provides a robust method for minimizing the loss function of an MLP. An epoch is usually defined as the number of iteration/s when all (or most) of the samples in the training set were successively used to compute the gradients. We call these constants (η, α, γ) hyperparameters: parameters outside of the model definition that are integral in selecting an appropriate model. These hyperparameters need to be chosen appropriately via a procedure called model selection (hyperparameter tuning).

Initialization presents a separate problem in learning the weights of a neural network. A poor choice of initial values of the weights may result to suboptimal optimization—both in the sense of the time required to approach convergence and its quality. In practice, we commonly start with small random absolute values or with small numbers having a mean of zero (Du and Swamy, 2014, pp. 108–110). Other methods suggest to initialize weights in the magnitude inversely proportional to the number of hidden units in a layer. Another more involved way to initialize weights is to perform an unsupervised pre-training of the network—an idea that played a significant role in the deep learning paradigm. This method employs unsupervised learning algorithms (discussed in the succeeding sections) usually trained in a greedy layer-wise manner. Bengio (2009) argues unsupervised pre-training helps both optimization (of the loss function) and regularization (discussed in the succeeding paragraphs) of a deep neural network.

For the sake of clarity, we consider deep multilayer perceptrons to be feedforward fully-connected artificial neural networks with more than one hidden layer. The number of layers defines the

depth of the network and the number of units in a layer defines its width. As the network grows, i.e. increases in depth and/or width, the risk of overfitting increases. Overfitting happens when an excessively complex statistical model reports exceptionally high performance during training; but then performs poorly when presented with new data, e.g. a test or validation dataset. We say: "the model overfits the training set" or " the model poorly generalizes on the test set."
To address ovefitting, we apply regularization methods to our model. One method is to introduce a regularization (weight decay) term λ in the loss function. In the case of an L2 regularization, the new loss function $J(w)$ becomes of the form:

$$J(w) = E(w) + \lambda \|w\|^2. \tag{2.6}$$

The regularization term penalizes the weights learned by the model—usually excluding the weights of the bias units—thereby inhibiting large absolute values of weights. We can also regularize our model by utilizing an early stopping method: where we prematurely stop the training based on some criterion (usually evaluated from a cross-validation set). Srivastava et al. (2014) recently introduced a kind of regularization method called dropout. In this method, hidden units in the network (along with their connection) are randomly "dropped out" (temporarily removed) with a probability of $(1 - p_r)$ in every epoch—where p_r is set to be the probability of retaining a unit during training. But during testing, all units are present and their outcoming weights are scaled by a factor of p_r. Dropout has been empirically proven to improve the generalization performance of neural networks in different tasks.
An MLP can be applied in a variety of remote sensing problems; and has been empirically compared by Benediktsson et al. (1990) against Bayesian approaches in this context. In a land cover classification task, the vectorized representation of the original bands of the image will form the input layer of a deep MLP. Consequently, the vector containing land cover labels will form the output layer of the same MLP. A similar architecture can tackle regression problems in remote sensing such as estimating certain indices, e.g. leaf area index, from satellite images.

2.1.3 Convolutional neural networks

Most of the structure an MLP uses also applies to a convolutional neural network: input layers, output layers, activation functions, and feedforward connection. The main difference of the two architectures is the density of connections between neurons. With the dense connection (i.e. every unit in the preceding layer is connected to all the units in the succeeding layer) employed by the MLP, directly learning spatial-contextual features (e.g. taking a neighborhood of pixels as an input for image classification) exponentially increases the number of its parameters. Thus, applying such an approach can be very expensive and will likely have poor generalization. On the contrary, a CNN utilizes differently structured hidden layers promoting sparser connection when compared to a fully-connected MLP of the same dimension. This architectural design of CNN with sparse and local connection was conceived incorporating several knowledge about images such as: stronger correlation between nearby pixels and reusability of local features from one region of the image to another. The hidden layer of a convolutional neural network consists of: convolutional, detector (non-linearity), and pooling layers. We often attach a fully-connected architecture (same as the hidden layers of an MLP) at the end of a stacked series of the three layers mentioned thus forming a deep architecture.
Many applications in computer vision and image recognition found convolutional neural networks to be highly effective (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; LeCun et al., 1998). So in the following discussion, we consider an image volume (2D image with say 3 bands/channels) as an example input to our convolutional neural network. The convolutional

layers applies the convolution operation to the input (a 2D convolution in the case of images); utilizing (often) multiple learnable filters thus producing a 3-dimensional volume of activation units. The depth of the volume being the number of filters. The length and width of the convolutional layer depends on the length and width of the input and the size of the kernel (sometimes also called the receptive field). The receptive field defines the region of the image from which a single feature (hidden unit) of a hidden layer is computed from. For a hidden layer directly following the input layer, it is equivalent to the kernel size. But for deeper (succeeding) hidden layers utilizing the same kernel size as the previous layers, the receptive field equivalently increases. To achieve sparse and local connection, we use a receptive field of size smaller than the input. With a smaller receptive field, we constrain an activation unit to be only affected by a local region in the input; and with the filters being shared across the whole input, we constrain different units to have the same parameters (sometimes called parameter tying) thus considerably decreasing the number of connections. In Figure 2.3 we highlight a hidden unit, together with the inputs connected to it, when formed by convolution with smaller receptive field (left) and by full connection (right).

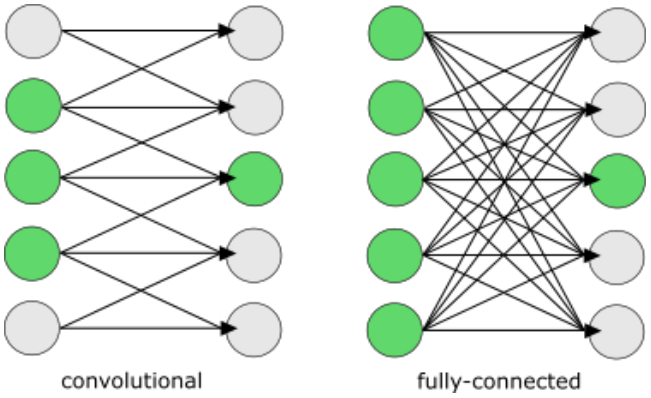


Figure 2.3: Locality and sparsity of a CNN

After producing activation values from the convolutional layer, a non-linear operation follows. We usually call this part the detector stage where we use a non-linear activation function, such as the rectified linear unit ($relu$)

$$h_2(a) = max(0, a) \tag{2.7}$$

to convert the activations from the convolutional layer. This operation retains the size of the volume of activation units. A pooling layer usually succeeds the convolutional layer and detector stage. Pooling further converts the activation units from the detector stage by using an aggregation function such as taking the maximum or average value over a region. We also often apply downsampling to reduce the size of resulting activation units (see Figure 2.4). Pooling promotes learning of representations invariant to small translations (Goodfellow et al., 2016, pp. 331–337)—one characteristic of features we want when performing object detection in images.

In contrast with an MLP of the same dimension (and taking the same region of pixels to extract spatial-contextual features from), convolutional neural networks are much easier to train. We could attribute it to the sparsity of connection of the network making it faster train, thus allowing users to perform more experiments. Goodfellow et al. (2016, pp. 337–339) further give insight by comparing a CNN to a fully-connected MLP with infinitely strong prior on the weights. They explain such a prior constrains the weights to be the same but shifted in space (parameter sharing) and all other weights, except for a local region in the input, are zeros (locality due to smaller receptive field).

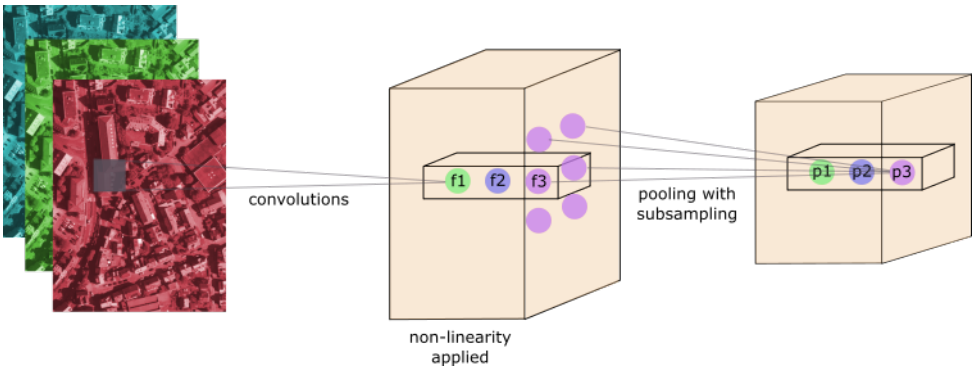


Figure 2.4: Pooling with subsampling of a CNN

In a remote sensing context, convolutional neural networks can play a role in learning representations able to efficiently capture contextual information for a classification or regression task. Similar to MLP, such tasks could be land cover classification or land cover index estimation from remote sensing images. However, only recently have there been interest on using these networks in a remote sensing context. Most of the unique properties of CNN were not strongly investigated in a remote sensing application.

2.1.4 Autoencoders

An autoencoder, auto-associator, or Diabolo network (Bengio, 2009) aims to learn compact and distributed representation of an input data. An autoencoder consists of two parts: the encoder—transforming the input into some form of representation—and the decoder—reconstructing the input from the codes produced by the encoder. They practically have the similar architecture as an MLP with one hidden layer, with the hidden units being the codes learned by the autoencoder. The same gradient-based optimization with backpropagation can be applied with training autoencoders. But, since the target values are the input values themselves, autoencoders are trained in an unsupervised manner.

Autoencoders can form deep architectures by being stacked and trained like an RBM and can also be deep by itself like an MLP with multiple hidden layers. Other variants exist such as denoising autoencoders (Vincent et al., 2008) and sparse autoencoders (Ranzato and LeCun, 2007). Instead using the original input, denoising autoencoders use corrupted version of the former as input. On the other hand, sparse autoencoder forces its feature representations to contain values that are mostly zero (sparsity constraint).

Applications in dimensionality reduction benefits from autoencoders. We can also use autoencoders for unsupervised feature learning in relevant classification and regression remote sensing problems. Autoencoders can play a role in unsupervised pre-training of other neural networks such as MLP or CNN as well. And unlike RBM, autoencoders are easier to train (Bengio, 2009).

2.1.5 Restricted Boltzmann machines

A Boltzmann machine (Ackley et al., 1985) is an energy-based undirected graphical model (Markov random field) having symmetric links between its visible (response) and hidden (latent) variables. By forbidding connections between variables in the same layers, we obtain a restricted Boltzmann machine (see Figure 2.5 and notice the absence of arrows). Same as the Boltzmann machine, an

RBM models the joint probability distribution

$$P(\mathrm{v} = v, \mathrm{h} = h) = \frac{1}{Z} \exp(-E(v, h)) \tag{2.8a}$$

$$E(v, h) = -b^T v - c^T h - v^T W h \tag{2.8b}$$

of its visible v and hidden variables h using an energy function E parametrized by the connections between the variables and their bias units (b, c, W). The constant Z, known as the partition function, sums exponential values of the energy function over all states. Naively performing this summation of exponential energy values poses a computational problem; hence, several clever algorithms arise to train RBM in a computationally tractable manner Goodfellow et al. (2016, pp. 598–620).

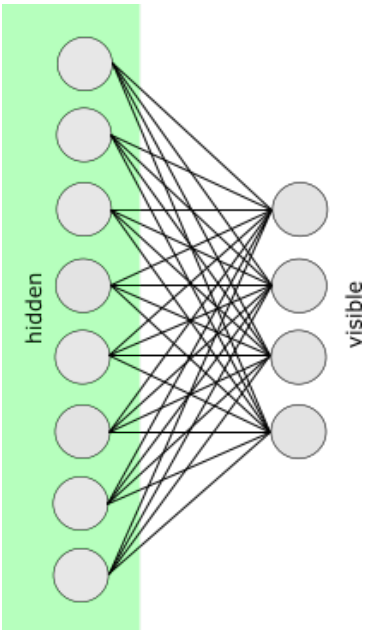


Figure 2.5: Restricted Boltzmann machine

Being a probabilistic model, we can apply maximum likelihood to estimate the parameters of an RBM. For this estimation, we can use a gradient-based optimization to maximize the log likelihood of the data (Goodfellow et al., 2016, pp. 597–598). Hinton (2002) proposed an efficient method, called contrastive divergence (CD), to approximate the gradient of this log likelihood function. Instead of maximizing the likelihood directly, CD minimizes the the difference between two Kullback-Leibler (KL) divergence values: comparing the equilibrium distribution against the empirical data distribution and the distribution of reconstructed data generated from a k-step Gibbs sampling. Murphy (2012, pp. 989–990) provides a pseudocode of the a CD-1 (k=1) algorithm for training an RBM. An alternative algorithm proposed by Tieleman (2008), called persistent CD, where instead of restarting the Markov Chain on the data after every parameter update—as what CD does—persistent CD initializes the Markov Chain for the next update on the last state of the previous update. Murphy (2012, pp. 990–991) also provides a pseudocode of the persistent CD algorithm. Both algorithms operate in an unsupervised manner.
Several variants of RBM exist each dealing with a different kind of variable. Murphy (2012, pp. 985–987) summarizes a number of them. We can stack restricted Boltzmann machines to construct deep architectures such as a deep belief network (DBN) and deep Boltzmann machine (DBM). A deep belief network (Hinton et al., 2006a) is a partially directed and undirected graphical model

formed by stacking a series of RBM. The bottom, starting from the visible units, part until the second to the last layer forms a directed graph while the top two layers forms an RBM. On the other hand, a deep Boltzmann machine (Salakhutdinov and Hinton, 2009) is basically a Boltzmann machine—hence fully undirected—composed of multiple hidden layers, i.e. units in the same layer are disconnected. Murphy (2012, p. 998) briefly describe a greedy layer-wise fashion for training both deep RBM-based models.
Similar to the autoencoder, the restricted Boltzmann machine and its variants can provide unsupervised feature learning methods for similar regression and classification problems described in the previous sections. We can also use such models for pre-training other deep neural networks in an unsupervised manner.

2.1.6 Other algorithms

Here are some other noteworthy algorithms in deep learning:

- Deep Q-Networks—Mnih et al. (2015) train a deep network using the q-learning algorithm (a form of reinforcement learning) to play Atari.
- Deep Convex Networks—Deng and Yu (2011) introduce a network where the weights can be learned by solving a convex optimization problem with closed solution.
- Multilayer Kernel Machines—Cho and Saul (2009) applies an iterative kernel mapping forming a deep architecture.
- Memory Networks—where some architectures of recurrent neural network fall. These networks integrate external memory modules in their structure.

2.1.7 Relevant benchmark results

Several deep learning algorithms currently holds record in a number of general image classification benchmark (Benenson, 2014). Lagrange and Saux (2015), and Paisitkriangkrai and Sherrah (2015) also recently submitted classification results of the ISPRS 2D semantic labeling benchmark dataset—which is also used in this thesis—utilizing a convolutional neural network in their classifier. But, unlike the architecture implemented in this study, the classifier made by Lagrange and Saux (2015) still rely on segmentation. On the other hand, Paisitkriangkrai and Sherrah (2015) used handcrafted features and post-processing to improve accuracy. Furthermore, both ignored—or at least did not discussed—other architectural variations and tuning techniques (changing connection density/sparsity, dimensions, and loss functions; applying pre-training, recurrence; etc.), and their effect to the performance of the classifier.

2.2 CHOSEN ALGORITHM AND DESIGN EXPERIMENTS

As discussed in chapter 1, contextual information is relevant when dealing with very high resolution airborne images. Filters learned by convolutional neural networks can capture contextual information. Furthermore, the hopefully translation-invariant features produced by the pooling layer are intuitively helpful to the problem of this study. Therefore, we choose convolutional neural networks to be the core algorithm for this research. In the next chapter, we discuss how we implemented the classifier exploiting the CNN architecture and how we perform several design experiments exploring different architectural variations and tuning techniques (as mentioned in the previous subsection 2.1.7) to improve the classifier's performance.

Chapter 3
Design and Implementation

This chapter discusses the relevant design decisions and implementation details taken to build the deep learning based classifier for the purposes of this study. The first section describes the architecture of the the artificial neural networks employed in the implementation of the classifier—hence, providing a set of common terms and notations used in the rest of the paper. The next section discusses the experimental analysis performed in search of an optimal configuration of the classifier. The chapter postpones and leaves specific architectural details of the deep learning based classifier of our primary interest to the next chapter.

3.1 NETWORK ARCHITECTURES

3.1.1 MLP

The MLP takes one pixel from an image as an input. Hence, the input layer consists of b units where b is equal to the number of bands of the input image. The layers have full connection from their preceding and to their succeeding layers as described in section 2.1.2 and illustrated in Figure 2.2. The units in the hidden layers utilizes the same activation function (varied in the experiments) while the final output layer utilizes the softmax function:

$$h_3(a_i) = \frac{e^{a_i}}{\sum_{j=1}^{J} e^{a_j}} \tag{3.1}$$

where $\sum_i h_2(a_i) = 1$ and $h_2(a_i) > 0$ to produce softmax scores for each class labels (Goodfellow et al., 2016, pp. 162–169). The classifier then assigns the label with the highest score to the pixel being classified. The output layer therefore consists of J units equivalent to the number of classes in the classification problem. With this structure of the output layer, the class labels are transformed into vector encoding where all elements of the vector are zero except for the element with index corresponding to the class label. For example: labels A, B, and C are transformed into (1, 0, 0), (0, 1, 0), and (0, 0, 1) vectors respectively.
The network minimizes a categorical cross-entropy objective function:

$$E_n(w) = -\sum_i t_i \log(y_i) \tag{3.2}$$

where t_i is the i^{th} element in the vector encoding of the true label and y_i is the i^{th} element of the output layer, computed for n samples. Applying L2 regularization, the equivalent objective function takes the form of equation 2.6. A stochastic gradient descent with momentum approach, discussed by Goodfellow et al. (2016, pp. 288–293) is used to optimize the loss function. Similar to the normal (sometimes called batch) gradient descent method, stochastic gradient descent updates the weights proportional to the negative gradient; with the proportion defined by the learning rate η introduced in section 2.1.2. But instead of updating the weights after computing the gradients

for all the training cases, stochastic gradient descent applies the update after computing the average gradient over a batch of randomly drawn training samples.
Aside from η, we need to set additional hyperparameters for this optimization method: momentum α, size of the batch, and criteria when to end the optimization. To improve the generalization error of the model, we also need to set the regularization hyperparameters: type of regularization applied (usually L2 or L1 parameter norm penalty), regularization term λ, the dropout rate (probability of dropping out a unit), and the early stopping parameter. The width, depth, and activation function used in the hidden layers can also be varied to find the optimal configuration of the model.

3.1.2 CNN+MLP

The convolutional neural network performs a pixel-wise classification of an image by taking a square patch of pixels P_{xy}, centered on the pixel (x, y) being classified, from the image as an input. The first layer forms a 3D input volume (tensor) of size i_s x i_s x b where i_s is the dimension of the input patch. The network then performs a series of 2D convolutions over the input, a unit-wise non-linear operation (activation function), and pooling with subsampling.
For this architecture, a "full convolution" is applied where the resulting output dimension from a convolution is calculated by the following equation

$$o = i + f_s - 1 \tag{3.3}$$

where o is the output dimension, i is the input dimension, and f_s is the kernel size. In "full convolutions", input images are automatically padded by appropriate number of zeros at their borders. The pooling operation is defined by a square region of dimensions p_s x p_s where the maximum value is taken to be the output (max pooling). We, by default, set the strides of the convolution to 1 pixel (but can be varied); while the pooling stride is usually set to p_s pixels—hence, producing non-overlapping pooling regions and downsampling the previous layer (in the first two dimensions) by a factor of p_s. Figure 3.1 illustrates the hyperparameters of the convolutional neural network, while Table 3.1 provides a short summary of these hyperparameters. Applying a "full convolution", the size of outputs from convolutional layers will only be affected by the pooling size and stride applied—hence, other hyperparameters (such as network depth) in model selection can be conveniently chosen.
We then flattened the last output volume produced from the series of convolution, non-linearity, and pooling into a one-dimensionl vector and connect it to layers with dense connection—similar to the hidden layers of a fully-connected MLP described in the previous section. This flattened vector contains the spatial-contextual features extracted by the CNN for the central pixel of the patch. The dense hidden layers uses the same activation functions as the non-linearity applied in the previous convolutional stage. When only one pixel is being classified at a time, the same output layer as described in the previous section can be used; and we center the input patch on this same pixel.

3.1.3 CNN (without MLP)

We also experimented with convolutional neural network architectures without utilizing dense layers at the end. For this CNN variant, we used "valid convolutions" where the resulting output dimension is calculated by the following equation

$$o = i - f_s + 1 \tag{3.4}$$

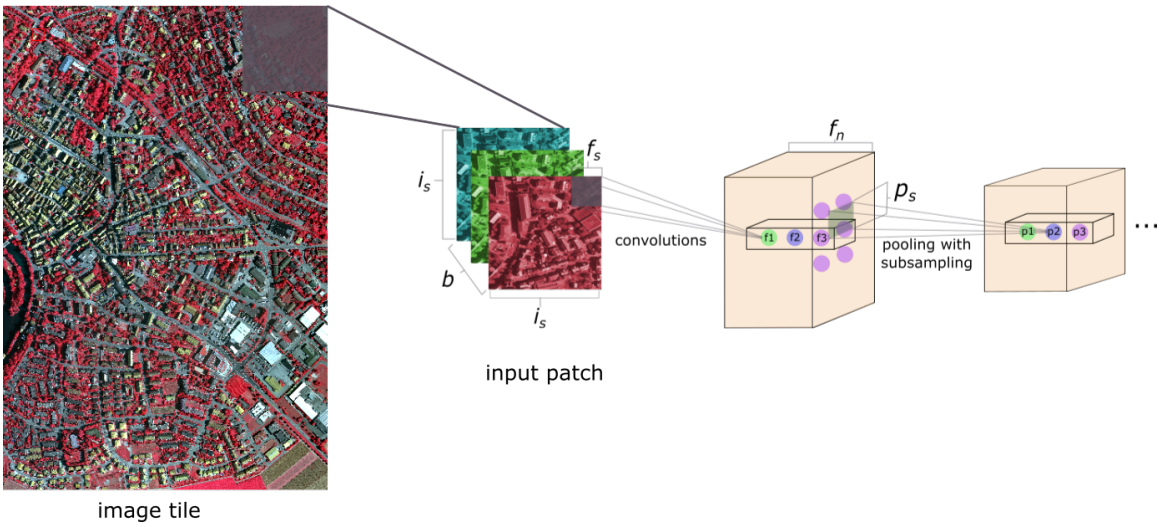


Figure 3.1: CNN hyperparameters, the kernel and pooling strides are defined by how much the the squares with f_s and p_s dimensions moves when performing convolution and pooling respectively.

Table 3.1: Summary of CNN hyperparameters

Hyperparameter	Description
Patch size i_s	The dimension of the square patch that is fed as an input to the CNN. It defines the extent of the region in an image where contextual information is being extracted for the classification of the central pixel of the patch.
Kernel size f_n	The dimension of the square kernels used in the convolutional layers of the CNN. Intuitively, this defines the extent of the local patterns that will be learned by the CNN as equivalent spatial-contextual features. The kernel stride (usually set to 1 pixel) defines how the kernels will be moved across the image to compute the features.
Number of filters f_s	The number of kernels/filters applied by the convolutional layers. Equivalently, this sets the maximum number (variety) of local patterns that can be learned by each layer of the CNN.
Pooling size p_s	The dimension of the pooling region where the maximum value will be taken as the output. The pooling stride, usually set to p_s, defines how the subsequent layer will be downsampled. It also equivalently sets the degree of invariance of the features that will be learned by the classifier to small translations.

where o is the output dimension, i is the input dimension, and f_s is the kernel size. For this kind of convolution, padding the input with zeros is unnecessary. We then chose hyperparameters (such as filter f_s and pooling p_s sizes) affecting the size of the output units for each layers such that the final output volume is a 1 x 1 x n (number of classes in the problem). Flattening the last output vector will then be unnecessary. Hence, with this variant, the (2D) spatial property of the features obtained from the convolutional layers are intuitively preserved. In this variant, the CNN directly returns the label of the central pixel of the input patch without the need of dense hidden layers of

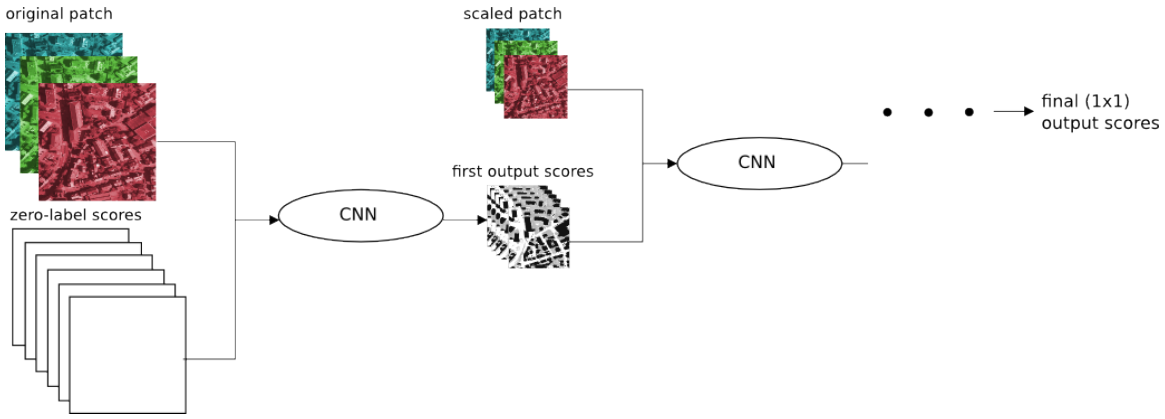


Figure 3.2: Diagram of a recurrent convolutional neural network (RCNN) architecture where each CNN instance uses the same set of weight values (parameter tying/sharing).

an MLP at the end.

3.1.4 Recurrent convolutional neural network

We also experimented with a recurrent convolutional architecture (Pinheiro and Collobert, 2014). The same components of the previously described convolutional architecture (without final dense layer) is used with few subtle differenes. Firstly, for this recurrent architecture, the output label scores of the network are fed back as an input to itself—hence, the term recurrence. So instead of having i_s x i_s x b (number of bands) input volumes, the recurrent CNN receives i_s x i_s x b+n (number of bands and classes) input volumes. We initialize the 0^{th} instance of the recurrent network with zero label score maps. A recurrent CNN is then formed by stacking instances of convolutional neural networks having the same (shared) weights. Figure 3.2 shows a graphic illustration of a recurrent convolutional neural network architecture.

In this recurrent architecture, succeeding CNN must then be fed with a scaled version of the original patch size with the same dimension as the output scores (downsampled by the "valid convolution" and pooling operations) from the previous instance. For scaling the patch, an approximate nearest neighbor method is applied. Similar to the previous architecture, hyperparameters (including the number of network instances) affecting the size of the output were chosen such that the final output volume is 1 x 1 x n.

Weight sharing across CNN instances in an RCNN allows us to increase the depth of the network (hence, increasing the number of features and model capacity as well) while maintaining the same number of parameters as compared to a standard CNN. Feeding output scores of a previous instance to the succeeding one also allows the model to learn contextual label dependency—similar to what an MRF-based classification is doing. In this way, the classification of a sample is not only conditioned on the raw pixel values of itself and nearby pixels but also on the label score values (of the neighborhood) as well.

3.2 DESIGN OF EXPERIMENTS

3.2.1 Experimental setup

Data on-hand

A subset of the ISPRS 2D semantic labeling benchmark data (Cramer, 2010) was used in the experiments. The dataset consists of 33 very high resolution (9 cm) airborne images of Vaihingen, Germany covering an area of about 7.4 x 4.7 square km. These 33 image tiles include an orthophoto, with 1 near infrared and 2 optical (Red and Green) bands, and a digital surface model (DSM). Sixteen out of the thirty-three tiles also have ground truth image containing the land cover classes used as labels in the classification. The land cover classes available in the ground truth images are: 1) impervious surface, 2) low vegetation, 3) buildings, 4) trees, 5) car, and 6) clutter. Figure 3.3 shows the orthophoto, DSM, and ground truth of image tile 1 (see ISPRS (2015) for tiling details).

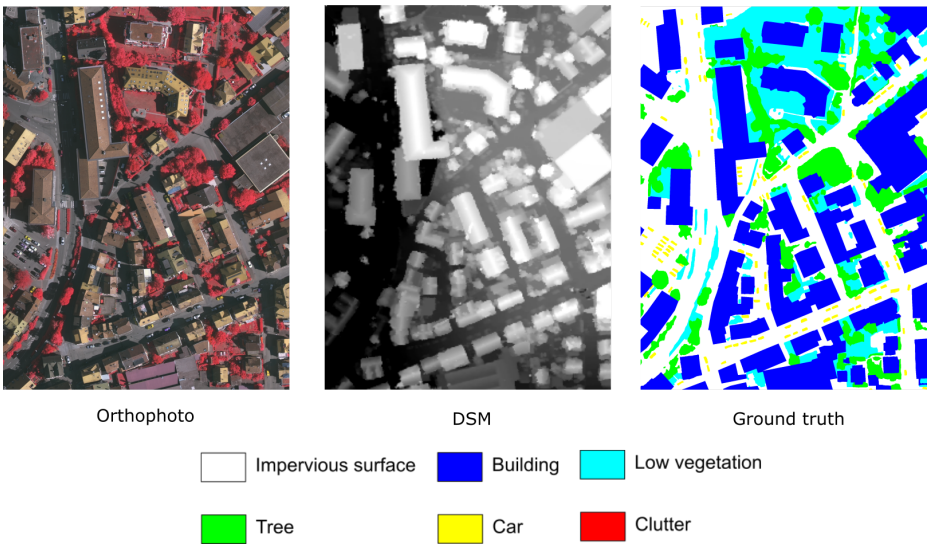


Figure 3.3: An example data showing the orthophoto, DSM, and ground truth of image tile 1.

Sampling

A stratified random sampling (based on class frequency) from 3 labeled tiles is used to build the training and (sparse) test samples. In the stratified random sampling, number of samples taken from each class are in linear proportion to their occurrence in a tile: e.g. if 25 % of pixels in tile 1 are building pixels then, 500 out of the 2000 sample points from tile 1 will be of the building class. In the domain adaptation analysis, test sets were separately taken from tiles where no training point was taken.

Testing

Sparse testing (only classifying sampled pixels from the image) is mainly used throughout the experiments—especially in the sensitivity analysis of hyperparameters—as full testing (classifying all the pixels in the image) can be computationally expensive. The classifier was evaluated over whole image tiles to print classified maps.

Normalization

Sample points/patches were normalized such that a set taken from an image tile has zero mean and unit standard deviation. For fully testing sampled tiles, the same normalization parameters used during the sampling were used. However, when fully testing unsampled tiles (in the domain adaptation analysis, see subsection 5.2.2), the average of the normalization parameters of all the sampled tiles were used. We applied this normalization scheme for all the experiments except for one experiment explained in subsection 3.2.4.

3.2.2 Initial experiments

We performed some initial experiments by training an MLP with 128 units in 2 hidden layers. We trained the network using stochastic gradient descent with momentum over 2668 training samples and evaluated using 2640 samples unseen during training taken from tiles 1, 3, and 5. In both setups, proper model selection was deliberately disregarded (since tuning was not the goal of this experiment).

On activation functions

Three activation functions were compared: the sigmoid, hyperbolic tangent ($tanh$), and the rectified linear unit ($relu$) functions (see equations 2.2, 2.3, and 3.1). We compared the overall classification accuracy of the MLP on the unseen data as the training epoch increases.

On regularization

The network was also evaluated with and without any regularization method applied. This experiment is done to show that neural networks (with a considerably large number of parameters, 17664 in this setup) are prone to overfitting. Hence, there will be a need of regularization techniques in training them.

On initialization

We also conducted a experiment on initialization. Here we compared an initialization technique proposed by Glorot and Bengio (2010) and when the weights of the network is initialized using values of 1. Due to time constraints, initializing the network using unsupervised pretraining was not studied.

3.2.3 Sensitivity to hyperparameters

The holdout validation method was used to select optimal configuration of the hyperparameters of the model. This method further splits the training set into another training set and a validation set–where the validation set is used to evaluate the criteria for selecting the "optimal" value of the hyperparameter/s. We use the architecture described in subsection 3.1.2 and the overall accuracy as the criteria for the following sensitivity analysis experiments. Each sensitivity experiment was designed independently, i.e. sets of samples were different in each experiment, thus absolute comparison of between overall accuracy values obtained from different experiments (e.g. OA from patch size experiment vs. OA from kernel size experiment) is irrelevant.
To address overfitting and improve the generalization (performance on unseen test data) of our networks, we apply 3 forms of regularization: an L2 weight decay term λ (see equation 2.6), an early stopping criteria, and dropout (Srivastava et al., 2014). For early stopping: we take a validation set and monitor the value of the loss function on this set, save the best value among all the

past epochs, and prematurely stop the training if the best value of the validation loss function does not change for e_n number of epochs after it has been saved. For dropout, the rate of "dropping out a unit" $(1 - p_r)$ should be set.

Patch size

The effect of the i_s hyperparameter was investigated by varying the sizes of input patches to a convolutional neural network with the following fixed configuration in Table 3.2. We trained the network using stochastic gradient descent with momentum over 1186 training samples while applying learning and regularization hyperparameters values in Table 3.3 tuned over 593 held-out validation samples. The network's overall accuracy was evaluated over 889 test samples. The following i_s values were used for this experiment: 9, 19, 33, 65, and 129.

Table 3.2: Patch size experiment: CNN configuration

Hyperparameter	Value
Layers [a]	I-C-A-P-D1-C-A-P-D1-F-D2-O
Nonlinearity used in A, F, and O [b]	$relu$-$softmax$
Width of F	128
Patch size i_s	(9, 19, 33, 65, 129)
Number of filters f_n	16
Kernel size f_s	3
Pooling size p_s	2

[a] Layer notation: I=Input, C=Convolution, A=Activation (1^{st} stage), P=Pooling, F=MLP hidden layer (fully-connected), O=Output, D1=Dropout in 1^{st} stage, D2= Dropout in 2^{nd} stage.
[b] $relu$ is used in A and F, while $softmax$ is used in O.
The weights (in the filters) are initialized using normalized initialization proposed by Glorot and Bengio (2010); the convolution stride is set to 1; and pooling stride is set to 2.

Table 3.3: Patch size experiment: learning and regularization hyperparameters

Hyperparameter	Values
Learning rate η [a]	(0.1, 0.001, 0.0001)
Momentum α	0.9
Learning rate decay η_d [a]	(0.01, 0.001)
Early stopping patience e_n	(10, 50, 150)
Max number of epoch	1000
Weight decay λ [b]	(0.01, 0.001)
Dropout rate in (D1, D2)	((0.7, 0.4), (0.5, 0.25), (0.3, 0.1))

[a] The learning rate decreases after each epoch defined by the function: $\eta(e) = \frac{\eta_0}{1+\eta_d * e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
[b] L2 parameter norm penalty is used.
The proportional factor γ is not used.
A fixed mini-batch size of 128 was used.

Kernel size

Aside from the patch size, we also investigated the effect of the kernel (filter) size f_s used by the convolutional layers of a CNN with the following fixed configuration in Table 3.4. We trained the network, using the same stochastic gradient descent method used in the previous patch size experiments, over 1329 training samples; while applying learning and regularization hyperparameters values in Table 3.5 tuned over 1329 held-out validation samples. The network's overall accuracy was evaluated over 2657 test samples. The following f_s values were used for this experiment: 3, 5, 9, 17, and 25.

Table 3.4: Kernel size experiment: CNN configuration

Hyperparameter	Value
Layers [a]	I-C-A-P-D1-C-A-P-D1-F-D2-O
Nonlinearity used in A, F, and O [b]	$relu$-$softmax$
Width of F	128
Patch size i_s	33
Number of filters f_n	16
Kernel size f_s	(3, 5, 9, 17, 25)
Pooling size p_s	2

[a] Layer notation: I=Input, C=Convolution, A=Activation (1^{st} stage), P=Pooling, F=MLP hidden layer (fully-connected), O=Output, D1=Dropout in 1^{st} stage, D2= Dropout in 2^{nd} stage.
[b] $relu$ is used in A and F, while $softmax$ is used in O.
The weights (in the filters) are initialized using normalized initialization proposed by Glorot and Bengio (2010); the convolution stride is set to 1; and pooling stride is set to 2.

Table 3.5: Kernel size experiment: learning and regularization hyperparameters

Hyperparameter	Values
Learning rate η [a]	(0.1, 0.001, 0.0001)
Momentum α	0.9
Learning rate decay η_d [a]	(0.01, 0.001)
Early stopping patience e_n	(5, 25, 75)
Max number of epoch	1000
Weight decay λ [b]	(0.01, 0.001, 0.0001)
Dropout rate in (D1, D2)	((0.7, 0.4), (0.5, 0.25), (0.3, 0.1))

[a] The learning rate decreases after each epoch defined by the function: $\eta(e) = \frac{\eta_0}{1+\eta_d * e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
[b] L2 parameter norm penalty is used.
The proportional factor γ is not used.
A fixed mini-batch size of 128 was used.

Number of filters

The effect of the number of filters f_n used by the convolutional layers of a CNN was also investigated. The network with the following fixed configuration in Table 3.6 was trained, using the

same stochastic gradient descent method used in the previous patch size experiments, over 1334 training samples; while applying learning and regularization hyperparameters values in Table 3.7 tuned over 4000 held-out validation samples. The network's overall accuracy was evaluated over 2666 test samples. The following f_n values were used for this experiment: 4, 8, 16, 32, 64, and 128.

Table 3.6: Number of filters experiment: CNN configuration

Hyperparameter	Value
Layers [a]	I-C-A-P-D1-C-A-P-D1-F-D2-O
Nonlinearity used in A, F, and O [b]	$relu$-$softmax$
Width of F	128
Patch size i_s	19
Number of filters f_n	(4, 8, 16, 32, 64, 128)
Kernel size f_s	3
Pooling size p_s	2

[a] Layer notation: I=Input, C=Convolution, A=Activation (1^{st} stage), P=Pooling, F=MLP hidden layer (fully-connected), O=Output, D1=Dropout in 1^{st} stage, D2= Dropout in 2^{nd} stage.
[b] $relu$ is used in A and F, while $softmax$ is used in O.
The weights (in the filters) are initialized using normalized initialization proposed by Glorot and Bengio (2010); the convolution stride is set to 1; and pooling stride is set to 2.

Table 3.7: Number of filters experiment: learning and regularization hyperparameters

Hyperparameter	Values
Learning rate η [a]	(0.5, 0.1, 0.01)
Momentum α	0.9
Learning rate decay η_d [a]	(0.01, 0.001)
Early stopping patience e_n	(40, 10)
Max number of epoch	1000
Weight decay λ [b]	(0.1, 0.01, 0.001)
Dropout rate in (D1, D2)	((0.7, 0.4), (0.5, 0.25), (0.3, 0.1))

[a] The learning rate decreases after each epoch defined by the function: $\eta(e) = \frac{\eta_0}{1+\eta_d * e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
[b] L2 parameter norm penalty is used.
The proportional factor γ is not used.
A fixed mini-batch size of 128 was used.

Depth

We also studied the effect of the number of convolutional layers C_n and number of dense/fully-connected layers F_n of a CNN with the following configuration in Table 3.8; trained using the same stochastic gradient descent method used in the previous patch size experiments over 5319 training samples; applied learning and regularization hyperparameters values in Table 3.9 tuned over 5319 held-out validation samples; and evaluated over 5332 test samples. The following C_n values were used for this experiment: 3, 4, 5, while fixing F_n to be equal to 1; and the following F_n values were used: 1, 2, 3, while fixing C_n to be equal to 3.

Table 3.8: Depth experiment: CNN configuration

Hyperparameter	Value
Layers [a]	I-(C-A-P-D1)xC_n-(F-D2)xF_n-O
Nonlinearity used in A, F, and O [b]	$relu$-$softmax$
Width of F	128
Patch size i_s	33
Number of filters f_n	16
Kernel size f_s	9
Pooling size p_s	2

[a] Layer notation: I=Input, C=Convolution, A=Activation (1^{st} stage), P=Pooling, F=MLP hidden layer (fully-connected), O=Output, D1=Dropout in 1^{st} stage, D2= Dropout in 2^{nd} stage.
[b] $relu$ is used in A and F, while $softmax$ is used in O.
The weights (in the filters) are initialized using normalized initialization proposed by Glorot and Bengio (2010); the convolution stride is set to 1; and pooling stride is set to 2.

Table 3.9: Depth experiment: learning and regularization hyperparameters

Hyperparameter	Values
Learning rate η [a]	(0.1, 0.01)
Momentum α	0.9
Learning rate decay η_d [a]	0.01
Early stopping patience e_n	40
Max number of epoch	1000
Weight decay λ [b]	(0.01, 0.001)
Dropout rate in (D1, D2)	((0.5, 0.5), (0.0, 0.0))

[a] The learning rate decreases after each epoch defined by the function: $\eta(e) = \frac{\eta_0}{1+\eta_d * e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
[b] L2 parameter norm penalty is used.
The proportional factor γ is not used.
A fixed mini-batch size of 128 was used.

3.2.4 Investigation of recurrence

We also investigated how recurrence in the architecture of the CNN can affect its classification performance. Two experimental setups were performed: one setup of a recurrent convolutional neural network (RCNN) utilizing input patch size i_s of 33 pixels and another utilizing 121 pixels. The first smaller patch size of 33 pixels was chosen as it showed good overall accuracy in the patch size experiment results (see Figure 5.5). However, such a small patch size has some limitations (discussed in the next subsection 3.2.4) when applied to an RCNN. Thus, we use another larger patch size of 121 pixels. We report both the overall accuracy and the classified map of tile 1 from these networks.

RCNN-33

We used a recurrent convolutional neural network comprising of 2 instances of a CNN with 3 convolutional layers applying "valid convolutions" for this experiment. Pooling drastically decreases

the output size of a convolutional layer, and thus is not feasible to be applied in this architecture with the given patch size. The first 2 convolutional layers comprise of 16 (9 x 9) kernels and use $relu$ as the activation function, while the last convolutional layer uses n (number of classes) 1 x 1 kernels and $softmax$ as the activation function. All the weights were initialized using the same initialization method used in the previous sensitivity analysis experiments.
We also trained this network using stochastic gradient descent with momentum method over 5319 training samples feeding mini-batches of 128 samples each iteration; applied learning and regularization hyperparameters values in Table 3.10 tuned over 5913 held-out validation samples; and evaluated over 5332 test samples aside from all the samples in image tile 1. Sixty columns and rows of (border) pixels on all sides of the tile were excluded. Because when using the next larger patch size, these pixels need to be padded by zeros; and from our observation, these zero-padded pixels were greatly misclassified during test.

Table 3.10: RCNN using patch size 33: learning and regularization hyperparameters

Hyperparameter	Values
Learning rate η [a]	(0.005, 0.0005)
Momentum α	0.9
Learning rate decay η_d [a]	(0.01, 0.001)
Early stopping patience e_n	(5, 50)
Max number of epoch	1000
Weight decay λ [b]	(0.01, 0.001)
Dropout rate	(0.0, 0.5)

[a] The learning rate decreases after each epoch defined by the function: $\eta(e) = \frac{\eta_0}{1+\eta_d * e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
[b] L2 parameter norm penalty is used.
The proportional factor γ is not used.
A fixed mini-batch size of 128 was used.
Dropout rates were uniform for all the convolutional layers.

RCNN-121

We also used a recurrent convolutional neural network comprising of 2 instances of a CNN with 3 convolutional layers applying "valid convolutions" for this experiment. The first 2 convolutional layers comprise of 25 (for the 1^{st}) and 50 (for the 2^{nd}) 8 x 8 kernels, applying a 2 x 2 max pooling, and using the $relu$ activation function. Same as the previous RCNN, the last convolutional layer uses n (number of classes) 1 x 1 kernels and $softmax$ as the activation function. Hence, aside from the patch size, this RCNN differ in 3 aspects from the first recurrent network described in the previous subsection 3.2.4: 1) uses a slightly smaller kernel (8 x 8 compared to 9 x 9 used by the previous one), 2) applies pooling layers, and 3) uses more filters to compensate the downsampling effect of pooling (25 and 50 compared to 16 of the first). All the weights were also initialized using the same initialization method used in the previous sensitivity analysis experiments.
The same stochastic gradient descent method (over 2667 training samples) was used with one implementation caveat—samples needed to be read from disk every iteration as they are (in total) too large to be loaded in memory (of the machine used in our experiments) for the whole training process. This caveat impractically prolongs training time; hence, to be logistically practical, proper holdout validation was not performed. With this implementation caveat, we also used a different

normalization scheme in contrast to what was explained in subsection 3.2.1. Instead of applying normalization for every set taken from a sampled tile, we apply the normalization process to the whole superset (containing sample sets from all tiles). We apply the following fixed learning and regularization hyperparameter values shown in Table 3.11.

Table 3.11: RCNN using patch size 121: learning and regularization hyperparameters

Hyperparameter	Values
Learning rate η [a]	0.005
Momentum α	0.9
Learning rate decay η_d [a]	0.01
Early stopping patience e_n	40
Max number of epoch	1000
Weight decay λ [b]	0.01
Dropout rate	0.2

[a] The learning rate decreases after each epoch defined by the function: $\eta(e) = \frac{\eta_0}{1+\eta_d * e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
[b] L2 parameter norm penalty is used.
The proportional factor γ is not used.
A fixed mini-batch size of 121 was used.
Dropout rates were uniform for all the convolutional layers.

For this specific experiment, we also compare an equivalent CNN architecture utilizing the same convolutional layers and input patch size (121). But for the CNN, the output score maps was not fed from the 3^{rd} to the 4^{th} convolutional layer. At the same time, weights of corresponding layers (in the RCNN) are not tied in the CNN architecture. We also applied the same fixed learning and regularization hyperparameters listed in Table 3.11.

3.3 FINAL IMPLEMENTATION

In this chapter we've shown how we investigate the components of our chosen core algorithm (CNN). In sum, we performed a total of 3 initial experiments on properties common to both MLP and CNN, 4 CNN hyperparameter sensitivity experiments (varying the patch size, kernel size, number of filters, and network depth), and 2 exploratory setups investigating the recurrent convolutional neural network architecture. We postpone the final implementation details of the deep learning based classifier compared with 2 other classifiers to chapter 4 and the results and discussions on these experiments to the 5^{th} chapter.

Chapter 4
Performance Comparison

This chapter compares the classification performance of our core deep learning based classifier—implemented based on the knowledge obtained in the design experiments described in the previous chapter—with other approaches. In the first section, we describe each approach that we compared. And in the succeeding section, we discuss how we compare each approach: which metrics were considered to be relevant in assessing all the approaches.

4.1 CLASSIFIERS

4.1.1 CNN+MLP

As discussed in section 2.2, convolutional layers in a convolutional neural network can capture spatial-contextual information. We therefore choose a convolutional neural network with a final dense layer (see subsection 3.1.2) as the classifier of our primary interest. Due to logistic reasons, we didn't perform a full model selection—rigorously searching all dimensions of the hyperparameter space—as doing so will take a considerably huge amount of time. But instead, we adopt the values of hyperparameters showing relatively good results in the sensitivity analysis (see subsection 5.1.2) with the constraint that the network should be relatively cheap to train.
We implemented the classifier of our primary interest as a CNN with 2 convolutional (applying "full convolutions"), and 1 dense layers accepting a 33 x 33 input patch. Each convolutional layer applies 16 9 x 9 kernels, followed by a $relu$ activation, then followed by a 2 x 2 max pooling operation. The convolutional stride is set to 1, while the pooling stride is set to 2. The resulting output volume from the 2 convolutional layers are then flattened into a one-dimensional vector and is connected to the succeeding dense layer. This single dense layer comprises of 128 units utilizing the same $relu$ function. The dense layer is then connected to the final output layer with n (number of classes) units utilizing the $softmax$ function.
The classifier is trained using stochastic gradient descent with momentum searching over the set of learning and regularization hyperparameters listed in Table 4.1.

4.1.2 Pixel-based MLP

We first compared the CNN+MLP classifier, described in subsection 4.1.1, with another classifier that does not take context into account. For this classifier, we use a multilayer perceptron (see subsection 3.1.1) accepting individual pixel values (RGB channels + DSM). For it to be comparable with the results of the CNN+MLP classifier, we use the same architecture as the final dense layer of the architecture described in the previous subsection 4.1.1. Hence, the network comprises of: an input layer with 4 units, fully-connected to a single hidden layer with 128 hidden units utilizing the $relu$ activation function, and finally connected to the output layer with n (number of classes) units utilizing the $softmax$ function. We trained the network using stochastic gradient descent with momentum searching over the set of learning and regularization hyperparameters listed in Table 4.2.

Table 4.1: CNN+MLP: learning and regularization hyperparameters

Hyperparameter	Values
Learning rate η [a]	(0.2, 0.1, 0.05)
Momentum α	0.9
Learning rate decay η_d [a]	(0.01, 0.001)
Early stopping patience e_n	(30, 10)
Max number of epoch	1000
Weight decay λ [b]	(0.1, 0.01, 0.001)
Dropout rate in (D1, D2)	((0.5, 0.25), (0.0, 0.0))

[a] The learning rate decreases after each epoch defined by the function: $\eta(e) = \frac{\eta_0}{1+\eta_d * e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
[b] L2 parameter norm penalty is used.
The proportional factor γ is not used.
A fixed mini-batch size of 128 was used.

Table 4.2: Pixel-based MLP and GLCM+MLP: learning and regularization hyperparameters

Hyperparameter	Values
Learning rate η [a]	(0.03, 0.003)
Momentum α	0.9
Learning rate decay η_d [a]	(0., 0.0001)
Early stopping patience e_n	(100, 20)
Max number of epoch	4000
Weight decay λ [b]	(0. 0.0001)
Dropout rate	(0.5, 0.0)

[a] The learning rate decreases after each epoch defined by the function: $\eta(e) = \frac{\eta_0}{1+\eta_d * e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
[b] L2 parameter norm penalty is used.
The proportional factor γ is not used.
A fixed mini-batch size of 128 was used.

4.1.3 GLCM+MLP

For this classifier, we used a similar architecture as the MLP used in the previous classifier. But instead of having 4 inputs (RGB channels + DSM), this classifier receives additional 7 handcrafted textural features calculated from different statistics of the gray level co-occurence matrix (GLCM (Haralick et al., 1973)). We applied the same approach as Onojeghuo and Blackburn (2011) used to produce the 7 textural features: mean, variance, homogeneity, contrast, dissimilarity, entropy, and second moment. And as Dorigo et al. (2012) did, we only compute the textural features for the band with the highest entropy.

Adopting the approach of Onojeghuo and Blackburn (2011) to determine the appropriate GLCM window size parameter: we observed the semivariograms of each class in image tile 1, and chose a window size that approximate the mean range (pixel distance/lag where the semivariance start to saturate) across all the semivariograms. See Figure 4.1 for the plots of the 5 semivariograms of each class. Based on these semivariograms, we chose the window size of 125 to calculate the

GLCM features. Gray level co-occurence matrices were computed using 4 offsets (directions): (1, 0), (1, 1), (0, 1), and (-1, 1). And the average of the GLCM statistic (e.g. average of entropy in the 4 directions) was used as the textural feature. Before computing all the GLCM statistics, we first compute the entropy for all bands in image tile 1. The near infrared band showed the highest mean entropy of 1.76 compared to the red band and green band with mean entropy values of 1.37 and 1.34 respectively. Hence, we only calculate the GLCM statistics of the near infrared band. Figure 4.2 shows the GLCM features extracted from the near infrared band of image tile 1.

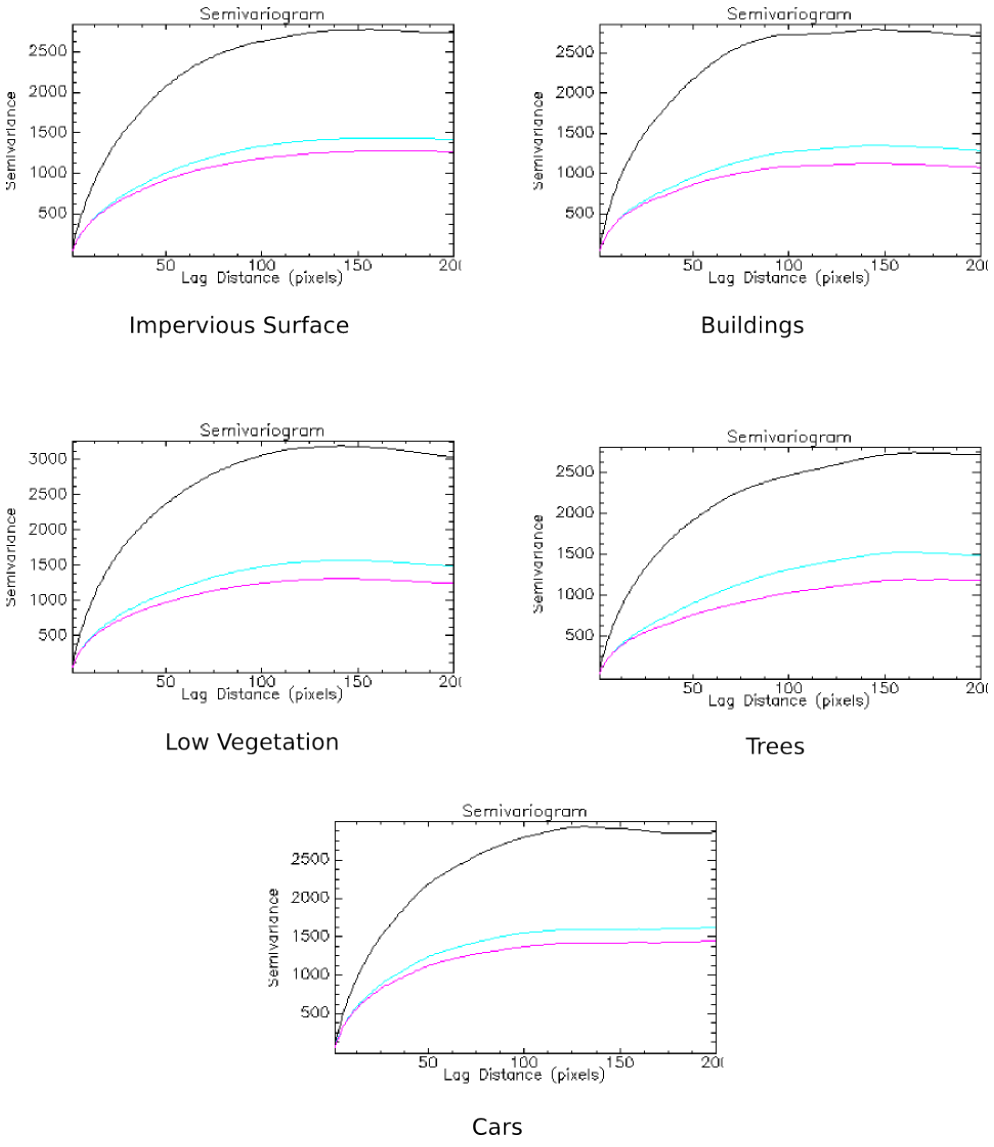


Figure 4.1: Plots of semivariograms of each class. The vertical and horizontal axes correspond to the semivariance and pixel lag respectively; while each curve corresponds to the 3 channels of the orthophoto. Samples used to estimate the semivariograms were taken from image tile 1.

We also trained the network using stochastic gradient descent with momentum searching over the set of learning and regularization hyperparameters listed in Table 4.2.

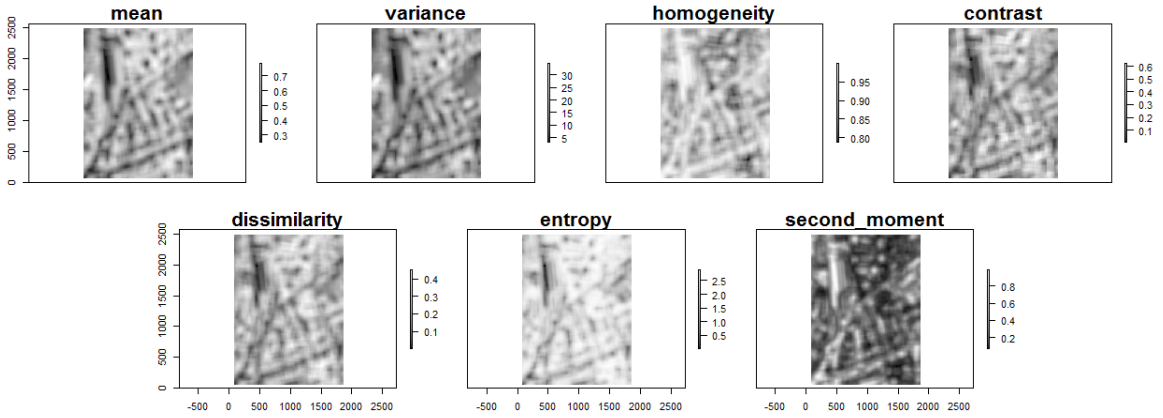


Figure 4.2: GLCM features of tile 1

4.2 METRICS

4.2.1 Accuracy gained from additional training samples

First, we studied the effect of training set size to the classification performance of our 3 classifiers. But instead of using sparse test samples (as we have done in the sensitivity analysis experiments), we evaluate the overall accuracies using all the "valid pixels"—pixels with enough information to provide the context required by the classifiers—in an image tile. Since we are capturing context within a specific contextual window (e.g. the GLCM window and the input patch) size, pixels within half the window size from the border of the tile will have "missing" information to provide this context. For our results to be comparable, the (larger) GLCM window size (compared to the CNN+MLP classifier's input patch size) was used to define the "valid pixels". Hence, we excluded 62 rows and columns of pixels from the border of each image tiles before evaluating the overall accuracies of the classifiers.
Only image tile 1 was used for this experiment as we expect the trend to carry over to the other tiles. For the evaluation of the next performance metrics, we use the classifier trained with the training set size that corresponds to the best overall classification accuracy over the image tile. As discussed in the next subsections, more than 1 tile was used to evaluate these other metrics.

4.2.2 Overall accuracy

As we have heavily used in the design experiments, we also used overall accuracy as a performance metric in the comparison of the classification approaches described in the previous subsections 4.1.1, 4.1.2, 4.1.3. We evaluated the classification accuracy of the classifiers across 2 "domains": (1) using tiles where training samples were taken, and (2) from tiles where no training samples were taken. With this setup, we can have an idea of the domain adaptability of the 3 classification approaches.

4.2.3 McNemar's test

We performed the McNemar's test (McNemar (1947) as cited in Bostanci and Bostanci (2013)) illustrated by Bostanci and Bostanci (2013) to test the significance of the difference in the classification accuracy of the classifiers. Foody (2004) also provides a good motivation of the use of the test in a remote sensing context. The test utilizes a 2 x 2 confusion matrix counting the samples which the

two classifiers (being compared) predicted correctly and incorrectly as shown in Table 4.2.3.

Table 4.3: McNemar's test: confusion matrix

	$C_1 =$ failure	$C_1 =$ success
$C_2 =$ failure	both fails (N_{ff})	1^{st} succeeds, 2^{nd} fails (N_{sf})
$C_2 =$ success	1^{st} fails (N_{fs}), 2^{nd} succeeds	both succeeds (N_{ss})

[a] $C_1 =$ failure/success signifies the incorrectly/correctly classified samples by the first classifier. Same applies to the 2^{nd} classifier C_2.

Following Bostanci and Bostanci (2013), the z-score is computed using equation 4.1:

$$z = \frac{|N_{sf} - N_{fs}| - 1}{\sqrt{N_{sf} + N_{fs}}} \tag{4.1}$$

where N_{sf} is the number of samples the first classifier correctly classified while the second classifier failed, and N_{fs} is the number of samples the first classifier failed to classify while the second classifier succeeded. Corresponding confidence levels from the resulting z-scores can be obtained using Table 4.4.

Table 4.4: McNemar's test: corresponding confidence levels for z-scores (Clark and Clark, 1999)

Z value	Two-tailed prediction	One-tailed prediction
1.645	90%	95%
1.960	95%	97.5%
2.326	98%	99%
2.576	99%	99.5%

The z-score approaches zero if the two classifiers perform equivalently and will diverge from zero as one classifier significantly outperforms the other. Two-tailed prediction is used two determine if the two algorithms' classification accuracies differ while one-tailed is used to test if a specific classifier performs better than the other. As was discussed in the previous subsection, this test was conducted separately within sampled image tiles and within the unsampled ones (domain adaptation setup).

4.2.4 Confusion matrix

Aside from the overall accuracy, we also investigated the resulting confusion matrices from classifying chosen image tiles using our 3 classification approaches. From the confusion matrix, we can compute the precision and recall of each classes using equations 4.2:

$$precision = \frac{tp}{tp + fp} \tag{4.2a}$$

$$recall = \frac{tp}{tp + fn} \tag{4.2b}$$

where tp is the number of true positives, fp is the number of false positives, and fn is the number of false negatives. We further compute the F1 scores using equation 4.3:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{4.3}$$

for us to have a single value to compare the performance of each classification approaches across our classes of interest. In this way, we can have a more detailed view on how the 3 classifiers perform. We also separately investigated the setups for sampled and unsampled image tiles.

4.2.5 Other metrics

Other performance metrics were also qualitatively investigated. One is the quality of the resulting classified map. Is the resulting map smoothly classified? Does the shapes classified objects resembles (e.g. straight edges for buildings and cars, while irregular rounded borders for trees, etc.)? Another metric we discussed qualitatively in the following chapter is the computational time of the classification approaches. Since various technologies (e.g. different programming languages, libraries, etc.) were used to implement the approaches, a quantitative analysis of computational time will be more complicated—as differences between the technologies should be taken into account. However, algorithmic complexity can be estimated; hence, we include a slight discussion on comparing the complexity of the 3 approaches.

4.3 PERFORMANCE RESULTS

In this chapter we've shown how we compared our core CNN+MLP classifier with other classification approaches—one depending on individual pixel values and the other using handcrafted features. In summary, we compared our 3 classification approaches using 6 metrics: overall classification accuracy, statistical difference in accuracy, class F1-scores, accuracy gained from additional training samples, map quality, and computational time and complexity. We postpone the results of these performance comparisons to the next chapter.

Chapter 5
Results and Discussions

In this chapter we report all the findings from the design experiments and performance comparison done as described in chapters 3 and 4. The first section discusses the results from the several architectural design experiments done. The succeeding section then presents the analysis done in evaluating our 3 classification approaches with the several performance metrics enumerated in section 4.2.

5.1 DESIGN ANALYSIS

5.1.1 Initial experiments

In this subsection, we discuss the results of the initial experiments performed to investigate the activation functions, regularization, and initialization methods used in an artificial neural network as explained in subsection 3.2.2.

Activation functions

In Figure 5.1, we can see that there is little or no difference with the overall classification accuracy of the MLP when using the 3 different activation functions. The sigmoid seems to slightly outperform the two functions (with the $relu$ having a slight advantage against the $tanh$ function). However, the sigmoid is considerably slower to train; as we can see in the graph that it reaches an upper bound at a much later epoch compared to the other two. So, for the next experiments, we chose to use the $relu$ function because it can be much faster to train compared to the sigmoid and it slightly performed better than hyperbolic tangent function.
We can also see, in the same figure, that the performance of the MLP decreases and stagnates after reaching a certain peak. Hence, there is clearly a need for an early stopping method—prematurely stopping the training before reaching a certain allowed maximum number of epoch.

Regularization

Without using any regularization method, a huge gap between the training and validation (unseen data) accuracy can be observed (as can be seen in Figure 5.2). This shows that the updates applied to the parameters of the network after some epoch only increases the accuracy on the training set, but actually hurts the classification performance of the network on unseen data—a clear sign of overfitting. But when each applying two of the regularization methods explained in subsection 3.2.3, we can see in Figure 5.3 that overfitting can be avoided. However, we can still see in this figure an increasing small gap between the training and validation accuracy (and at the same time stagnation of the latter). To address this small portion of overfitting that may occur, we apply an early stopping method explained in the same subsection 3.2.3.

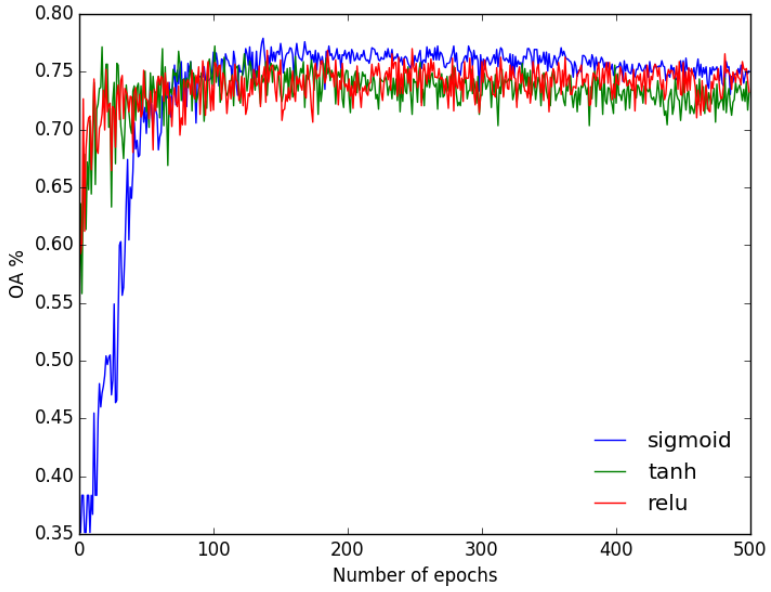


Figure 5.1: Activation function experiments

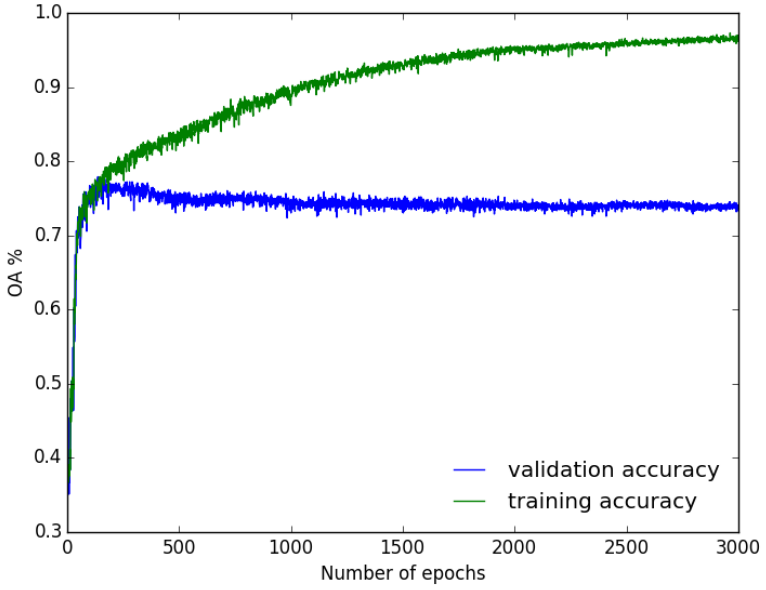
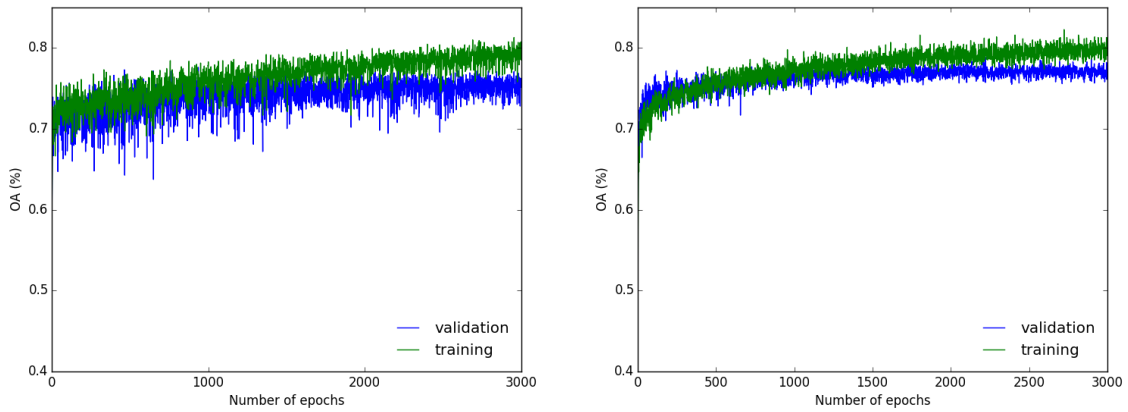


Figure 5.2: Regularization experiments: overfitting

Initialization

We can see in Figure 5.4 that a poor choice of initialization can prevent the network from learning anything. In this figure, we plot the training accuracy of the MLP trained in this experiment against the number of epoch. In the first setup the uniform initialization proposed by Glorot and Bengio (2010) was used; while for the second setup, values of 1 was used to initialize the parameters of the network. For the other experiments, we stick to using the uniform initialization.



(a) L2 weight decay (b) Dropout

Figure 5.3: Effect of applying L2 weight decay and Dropout as a regularizer.

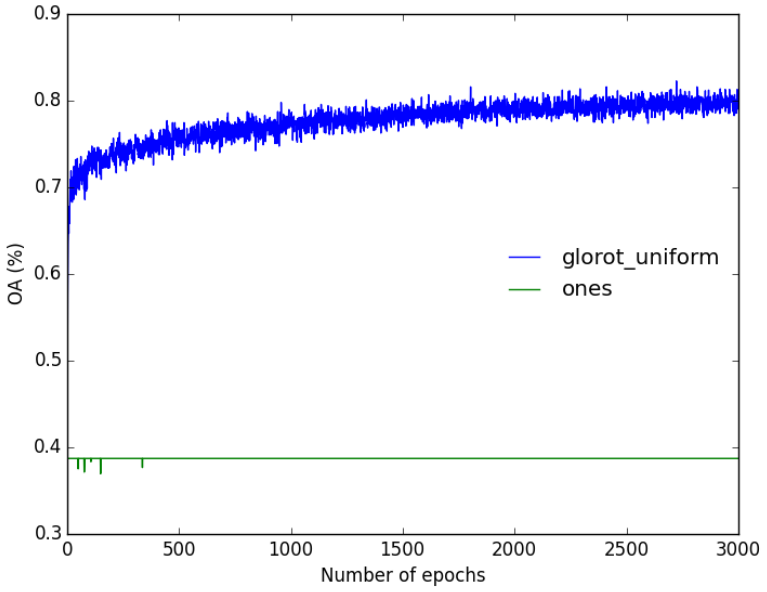


Figure 5.4: Effect of network initialization. Training accuracy of a network initialized with ones (in green) and the same network initialized using the uniform initialization proposed by Glorot and Bengio (2010)

5.1.2 CNN sensitivity analysis

In this subsection, we present the results of the sensitivity analysis done to a convolutional neural network with an architecture described in subsection 3.1.2. The overall classification accuracy of the network was measured as each of the 4 hyperparameters (enumerated below) were varied. All the hyperparameter search space configuration were reported in subsection 3.2.3.

Patch size

We can think of the patch size as the maximum span of contextual information being taken into account when the network classifies a pixel. Figure 5.5 shows how the classification performance of the CNN behaves as we varied the size of input patches fed to the network. As we increase the patch size from an input patch of 9 x 9 pixels, we can generally observe a trend of increasing overall accuracy from 75.58 % peaking at 78.97 % using a patch size of 33 x 33 pixels. However, further increasing the patch size beyond 33 x 33 degrades the overall accuracy of the network up to 73.57 % using an input patch of 129 x 129 pixels.

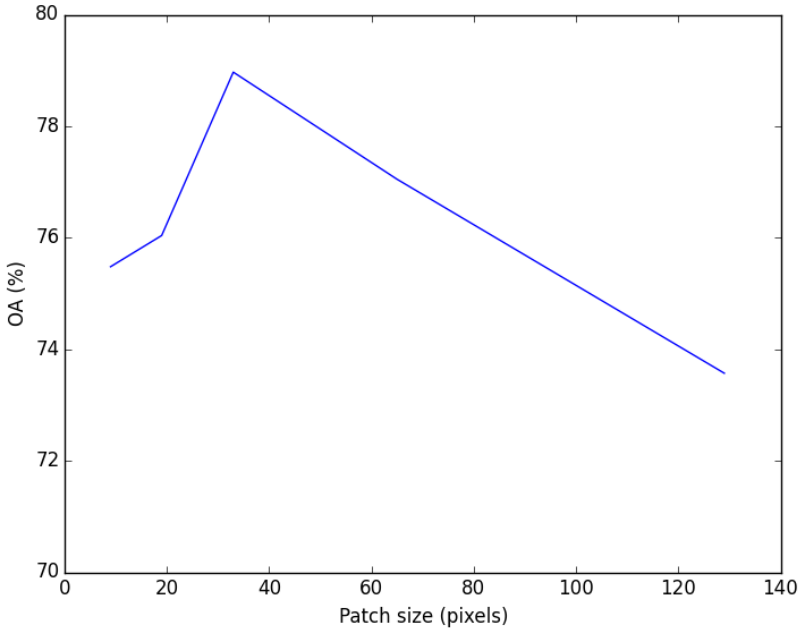


Figure 5.5: Varying the patch size of a CNN

The degradation in the overall accuracy could be caused by taking too much irrelevant context especially for the case of pixels near the border of an object (e.g. corner of a building). We can also attribute this effect to the relatively small training set size used for this experiment (1186). As a more complicated model (larger patch size, hence more features and more parameters for the dense layer) will definitely underperform compared to a less complicated one when training with an insufficiently small set of examples. An equivalent CNN setup using a larger patch size, discussed in subsection 5.1.3 will help to prove this second interpretation. For our CNN+MLP classifier, we stick to using the 33 x 33 input patch size showing the highest overall accuracy for this experimental setup.

Kernel size

The kernel size defines the maximum size of the patterns (e.g. an edge or a gradient) the classifier can learn to look for in order to discriminate the land cover classes of interest in this study. Figure 5.6 shows how the classification performance of the CNN behaves as we varied the kernel size used by the convolutional layers of the network. Similar to the patch size, we can generally observe a increasing overall accuracy (except for kernel size = 5) starting with 80.47 % using a 3 x 3 kernel, peaking at 81.71 % using a 17 x 17 kernel, and degrading to 77.98 % using 25 x 25 kernel.

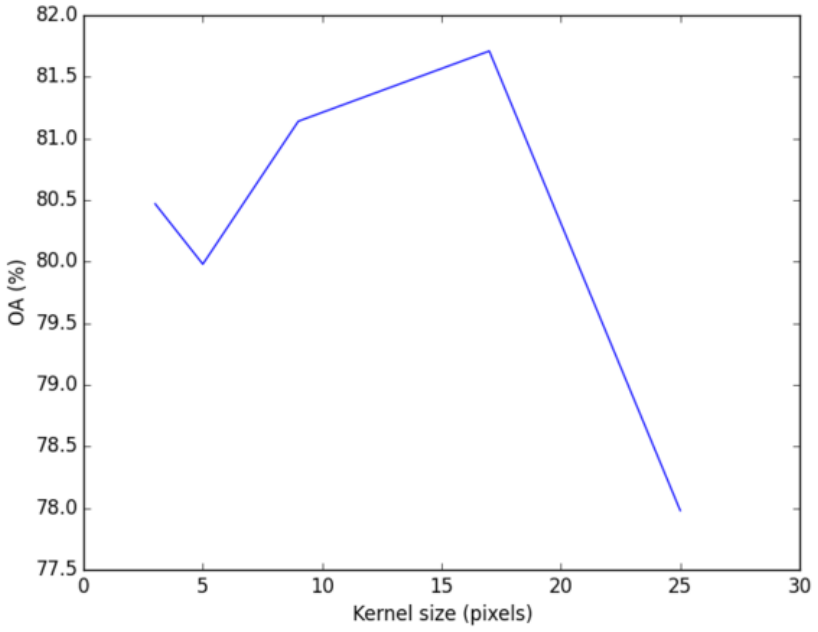


Figure 5.6: Varying the kernel size of a CNN

A larger kernel can theoretically learn all the patterns that can be learn by a smaller one; hence, increasing the kernel size should not degrade the accuracy. However, a network using a larger kernel size also quadratically increases the number of its parameters in the convolutional layers—and therefore, the network will be harder to train. This could be a plausible cause of the decrease in accuracy by using a larger 25 x 25 kernel. We can also observe a relatively small increase in overall accuracy (less than 0.6 %) observed from using a kernel size of 9 x 9 compared to an almost doubled size of 17 x 17 kernel. Thus, for the final CNN+MLP classifier, we strike a balance between overall accuracy and computational time by applying 9 x 9 kernels instead of 17 x 17 ones.

Number of filters

The number of filters sets the maximum number of spatial patterns the classifier can learn to look for in order to discriminate the classes of interest. Figure 5.6 shows how the classification performance of the CNN behaves as we varied the number of filters used by each convolutional layer of the network. Similar to the kernel size, we can generally observe a increasing overall accuracy starting with 76.78 % using 4 filters, peaking at 80.42 % using 16 filters, and going down to 77.79 % using 128 filters.

Analogously similar to the kernel size, a larger number of filters can also theoretically learn all the sets of filters that can be learned by a smaller number of filters. Increasing the number of filters, however, not only increases the number of features but also the number of parameters in all layers of the CNN—resulting to a much harder optimization problem (training). For our CNN+MLP classifier, we used 16 filters (showing the highest overall accuracy in this experiment) in each convolutional layer.

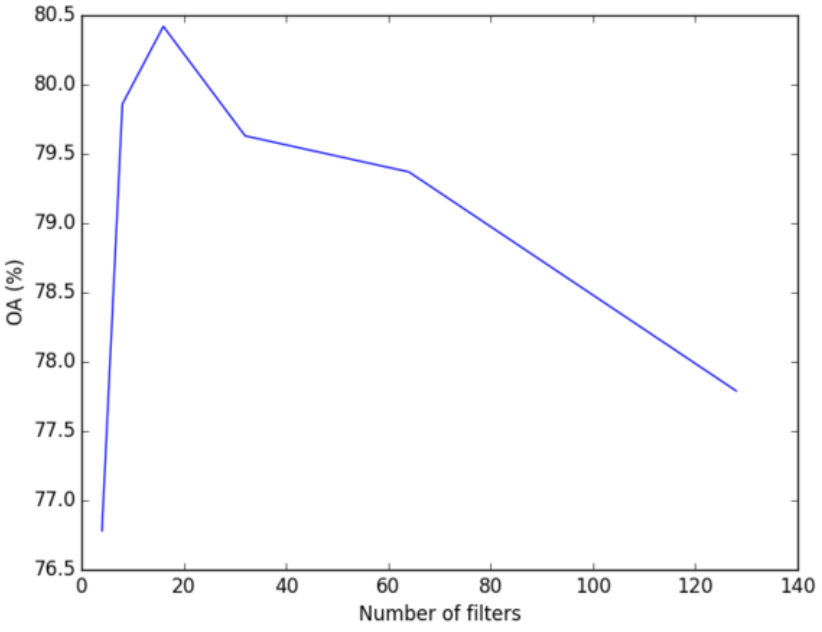


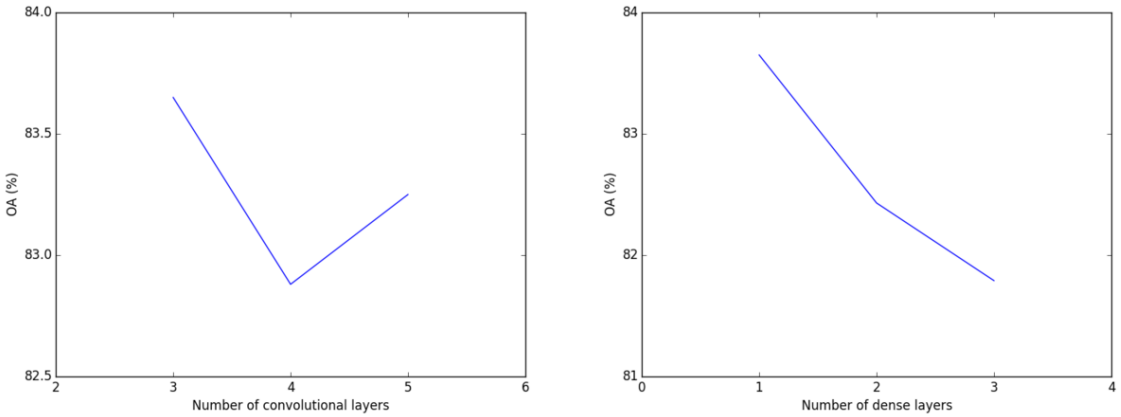
Figure 5.7: Varying the number of filters of each convolutional layer of a CNN

Depth

The depth of an artificial neural network is usually defined as the minimum number of operations separating the the input and output layers. For this experiment, however, we define 2 counts of network depth: one counting the number of convolutional layers and the other counting the number of dense layers at the end of the network. Figure 5.8 shows the effect of varying both the number of convolutional (left plot) and dense (right plot) layers of a CNN to its classification accuracy. Increasing both the number of convolutional layers and the number of dense layers does not seem to help the classification performance of the network.

Similar to increasing the number of filters, the deeper the network is, the more parameters and features it will have. Therefore, these deeper networks will suffer the same issue of difficulty in optimization (underfitting). But for this experiment we purposely use a relatively larger training sample set (5319) to address this said issue. We can also think of stacking more convolutional layers as allowing the network to learn features of increasing abstraction (e.g. from edges to shapes to objects). Assuming the sample size we used was "sufficient" to train networks in this experiment, then a possible interpretation of the classification performance not improving as the network depths were increased could be that that the complexity of the classification problem at hand does not require to learn features of higher abstraction. Hence, for our CNN+MLP classifier, we use a simpler architecture utilizing 2 convolutional layers and 1 dense layer.

Recently published work by Zhao and Du (2016), using a multi-scale convolutional neural network to classify 2 hyperspectral image datasets and 1 very high resolution image acquired using Worldview-II satellite, finds similar results. Such that further increasing the depth of a CNN (more than 2 or 3) does not considerably improve—and at times, could degrade—the overall classification accuracy of the network.



(a) Varying the number of convolutional layers (b) Varying the number of dense layers

Figure 5.8: Effect of varying the number of layers (convolutional—left, and dense—right) in a convolutional neural network.

5.1.3 Recurrent convolutional neural network

RCNN-33

The recurrent convolutional neural network using 33 x 33 input patches classified 81.98 % of the sparse test set correctly. But when evaluated over image tile 1 (excluding near-border pixels as explained in subsection 3.2.4), it only classified 72.61 % of the pixels correctly. Figure 5.9 shows the classified map of image tile 1 using this recurrent convolutional architecture.

RCNN-121

The recurrent convolutional neural network utilizing a larger input patch of 121 x 121 pixels shows more promising results. It classified 80.70 % of the sparse test set and 80.67 % of pixels in image tile 1 correctly. On the other hand, an equivalent CNN architecture using the same input patches performed slightly better (around 1 % for both tests) than the RCNN—classifying 81.99 % of the sparse test and 81.67 % of pixels in image tile 1 correctly. Figure 5.10 shows the classified maps of both the recurrent and standard CNN using input patch size of 121.
The 4 subtle differences between the two recurrent convolutional neural networks (aside from the size of the input patch used) are:

- A slightly larger kernel applied by the convolutional layers of the first network (9 x 9 compared to 8 x 8).
- A larger sample set used to train the first network.
- The absence of pooling layers in the first network.
- A larger number of filters in the second network to compensate the pooling.

The first two points (larger kernel and training sample size) should intuitively favor the first network. Hence, we could attribute the drop in classification performance (from sparse test to the whole image tile) of the first recurrent network to its smaller patch size, absence of pooling layers, and fewer filters used.

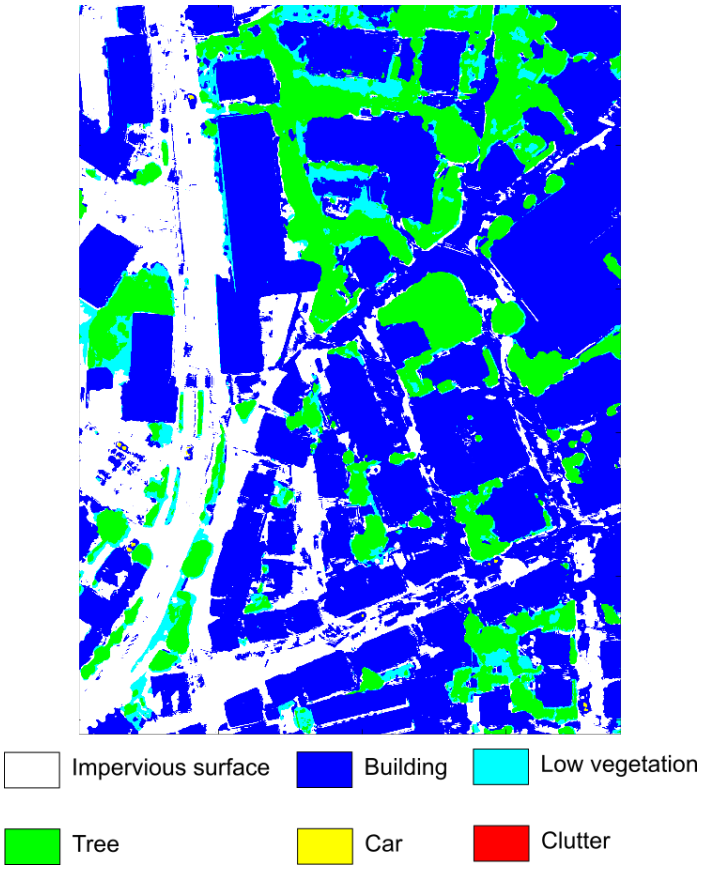


Figure 5.9: Classified map of a recurrent convolutional neural network using 33 x 33 input patches.

Both the recurrent and standard version of CNN using a 121 x 121 patch size shows promising classification accuracy results. Moreover, we can observe a smoother map resulting from the recurrent architecture compared the non-recurrent one as seen in Figure 5.10. This results confirms our intuition about the recurrence introduced in the network's architecture: by modeling contextual label dependency (as explained in subsection 3.1.4), the network is somehow performing a post-classification spatial regularization—resulting to a smoother classified map. But in this case, the smoothing also degraded the classification accuracy as car pixels were all smoothed out. We expect that with enough number of training samples, the network can learn an appropriate "smoothing factor" to correct this oversmoothing. Although with these interesting results at hand, we didn't have enough time to further explore this architecture (and use it as our primary classifier—performing model selection and performance analysis) mainly due to the implementation caveat explained in subsection 3.2.4. Thus for our primary classifier, we settled with a standard CNN (see subsection 4.1.1). We leave a more in-depth study of this recurrent architectures to future research.

5.2 PERFORMANCE ANALYSIS

5.2.1 Accuracy gained from additional training samples

Figure 5.11 shows how the 3 classification approaches performed over tile 1 using varying training set sizes. The CNN+MLP classifier generally outperforms the pixel-based MLP classifier for about almost 3% in average and also outperforms the GLCM+MLP classifier for more than 1% in

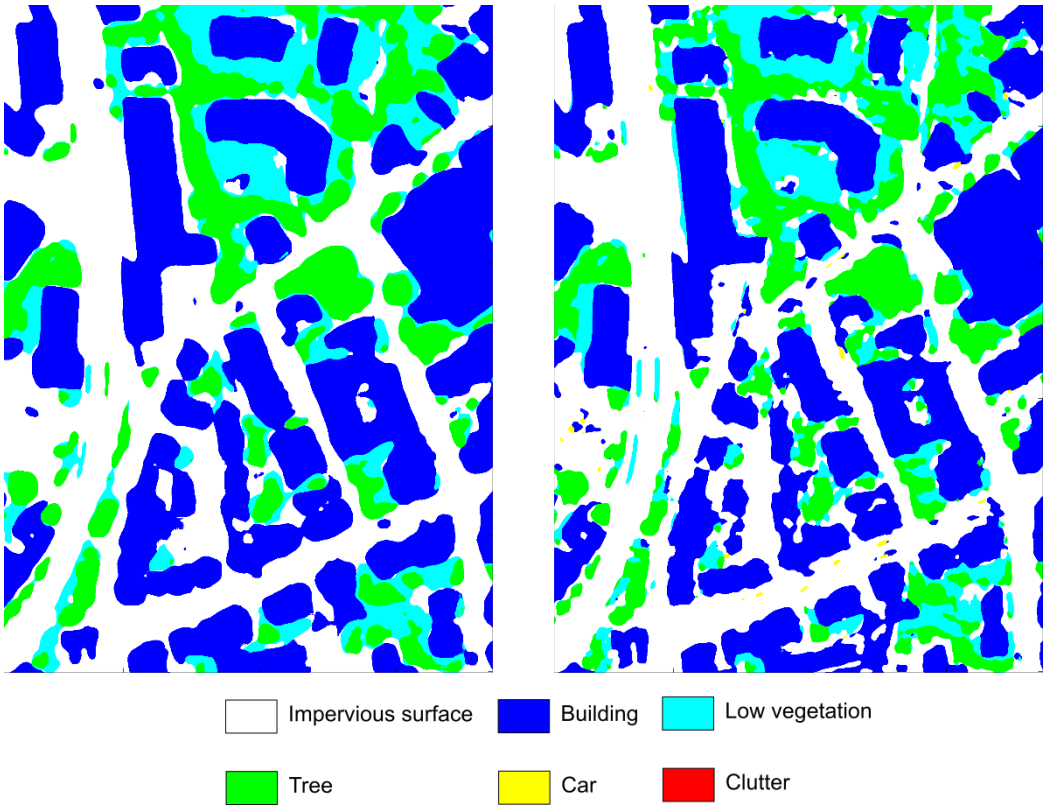


Figure 5.10: Classified maps of a recurrent (left) and standard (right) convolutional neural network using 121 x 121 input patches.

average (except for the case of the smallest sample size, where GLCM+MLP has 0.15% advantage). The CNN+MLP classifier also has relatively larger gains in overall accuracy as the sample size is increased—averaging 0.42% increase in overall accuracy for every 1000 training samples added compared to 0.01% and 0.17% average increase for the pixel-based MLP and GLCM+MLP classifiers respectively.

Increasing the training set size seems to be more beneficial to the CNN+MLP classifier than the other 2. The gap between the classification accuracies of the 3 classifiers even tends to increase as the as the the training sample size increases; with the classifiers having almost the same accuracy for the case of the smallest sample size used. This confirms the characteristics of deep neural networks of requiring and benefiting from large volume datasets. However, for very high resolution aerial images, one can conveniently obtain a sizeable set of samples as one object in an image (e.g. a building) could already be composed of thousands of pixels.

5.2.2 Overall accuracy

Sampled domain

Table 5.1 shows the overall classification accuracy results of the 3 classifiers over the 3 image tiles where training samples were taken. A similar trend as the results in the previous subsection can be observed for the all the 3 image tiles: the CNN+MLP classifier outperforming the other 2, with the pixel-based MLP classifier having the lowest classification accuracy.

Table 5.1: Overall accuracy results of the 3 classifiers on image tiles 1, 3, and 5 (where training samples where taken).

Image tile	CNN+MLP	Pixel-based MLP	GLCM+MLP
1	84.80%	80.43%	82.41%
3	82.95%	76.69%	80.49%
5	88.66%	84.96%	87.03%

This result presents two important points: for classifying image tiles where training samples are taken (sampled domain),

- the handcrafted GLCM features helps to improve overall classification accuracy on the problem at hand;
- but, moreover, the spatial-contextual features learned—automatically from the data—by the CNN+MLP classifier can further improve the accuracy.

Unsampled domain (adaptation)

Table 5.2 shows the overall classification accuracy results of the 3 classifiers over the 3 unsampled image tiles (where no training samples were taken). The CNN+MLP classifier continues to outperform the other 2, but now with a considerably larger margin compared to the results from classifying sampled image tiles. Moreover, the GLCM+MLP classifier performs poorly on image tiles 32 and 37 (where the pixel-based MLP classifier outperformed it).
This results continue to support the superiority (in terms of overall classification accuracy) of the CNN+MLP classifier. The drop in performance of the other 2 classifiers can be attributed to the

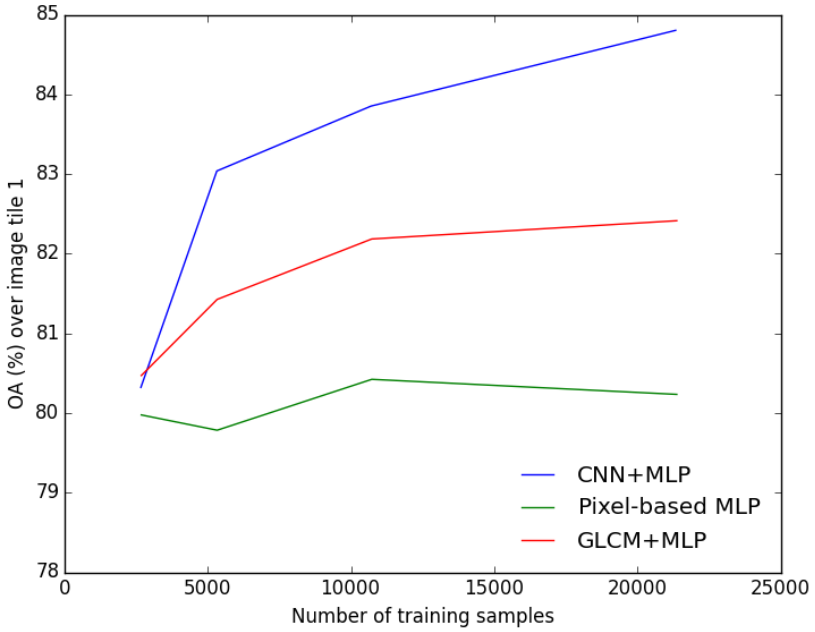


Figure 5.11: Performance of the classifiers on varying training set sample size.

Table 5.2: Overall accuracy results of the 3 classifiers on image tiles 7, 32, and 37 (where training samples where not taken).

Image tile	CNN+MLP	Pixel-based MLP	GLCM+MLP
7	78.05%	53.56%	55.27%
32	82.07%	65.08%	28.16%
37	72.26%	34.70%	28.18%

possible differences between the two unsampled and sampled domains: e.g. lighting when the raw images were taken, object (e.g. building) sizes and or density, etc. The features learned by the classifier seem to be more robust to adapting to this difference. However, one implementation detail should be noted that could change the results in this analysis. As explained subsection 3.2.1, normalization parameters used for fully testing the unsampled tiles were taken to be the average of the normalization parameters from the sampled tiles. This method seems to work (although there is still a clear drop in classification accuracy) for the CNN+MLP classifier, but it could not be the case for the other 2. The other normalization scheme used in the experiments using a recurrent convolutional neural network accepting 121 x 121 input patches could have been a better way to normalize the samples in this experiments. However, due to time constraints, we leave the proof of this claim for future works.
We perform the McNemar's test in the following subsection 5.2.3 to test the statistical significance of the classification performance reported in this subsection.

5.2.3 McNemar's test

GLCM+MLP vs Pixel-based MLP classifier

Table 5.3 shows the confusion matrix showing the corresponding confusion matrix of the McNemar's test comparing the performance of the GLCM+MLP and pixel-based MLP classifiers evaluated over the sampled domain. A z-score of 298.91 favoring the GLCM+MLP classifier can be derived from this confusion matrix. The corresponding confidence level (see Table 4.4) for this z-score suggest a strong statistical significance (more than 99% for both the one-tailed and two-tailed prediction). Hence, we reject the equivalent null hypothesis that there is no significant difference between the classification performance of the GLCM+MLP and pixel-based MLP classifiers over the sampled domain.

Table 5.3: Confusion matrix of the McNemar's test comparing the classification performance of GLCM+MLP and pixel-based MLP classifiers over the sampled image tiles 1, 3, and 5.

	GLCM+MLP = failure	GLCM+MLP = success
Pixel-based MLP = failure	1759400	1009759
Pixel-based MLP = success	627305	10707496

[a] GLCM+MLP/Pixel-based MLP = failure/success signifies the samples classified by the GLCM+MLP/pixel-based MLP classifier incorrectly/correctly.

Table 5.4 shows the confusion matrix showing the corresponding confusion matrix of the McNemar's test comparing the performance of the GLCM+MLP and pixel-based MLP classifiers evaluated over the unsampled domain. A z-score of 842.43 favoring the pixel-based MLP classifier can be calculated from this confusion matrix. The corresponding confidence level (see Table

4.4) for this z-score suggest a strong statistical significance (more than 99% for both the one-tailed and two-tailed prediction). Hence, we reject the equivalent null hypothesis that there is no significant difference between the classification performance of the GLCM+MLP and pixel-based MLP classifiers over the unsampled domain.

Table 5.4: Confusion matrix of the McNemar's test comparing the classification performance of GLCM+MLP and pixel-based MLP classifiers over the unsampled image tiles 7, 32, and 37.

	GLCM+MLP = failure	GLCM+MLP = success
Pixel-based MLP = failure	4428680	1426019
Pixel-based MLP = success	3247137	3201991

[a] GLCM+MLP/Pixel-based MLP = failure/success signifies the samples classified by the GLCM+MLP/pixel-based MLP classifier incorrectly/correctly.

CNN+MLP vs Pixel-based MLP classifier

Table 5.5 shows the confusion matrix showing the corresponding confusion matrix of the McNemar's test comparing the performance of the CNN+MLP and pixel-based MLP classifiers evaluated over the sampled domain. A z-score of 526.98 favoring the CNN+MLP classifier can be derived from this confusion matrix. The corresponding confidence level (see Table 4.4) for this z-score suggest a strong statistical significance (more than 99% for both the one-tailed and two-tailed prediction). Hence, we reject the equivalent null hypothesis that there is no significant difference between the classification performance of the CNN+MLP and pixel-based MLP classifiers over the sampled domain.

Table 5.5: Confusion matrix of the McNemar's test comparing the classification performance of CNN+MLP and pixel-based MLP classifiers over the sampled image tiles 1, 3, and 5.

	CNN+MLP = failure	CNN+MLP = success
Pixel-based MLP = failure	1565137	1204022
Pixel-based MLP = success	513406	10821395

[a] CNN+MLP/Pixel-based MLP = failure/success signifies the samples classified by the CNN+MLP/pixel-based MLP classifier incorrectly/correctly.

Table 5.6 shows the confusion matrix showing the corresponding confusion matrix of the McNemar's test comparing the performance of the CNN+MLP and pixel-based MLP classifiers evaluated over the unsampled domain. A z-score of 1526.36 favoring the CNN+MLP classifier can be calculated from this confusion matrix. The corresponding confidence level (see Table 4.4) for this z-score suggest a strong statistical significance (more than 99% for both the one-tailed and two-tailed prediction). Hence, we reject the equivalent null hypothesis that there is no significant difference between the classification performance of the CNN+MLP and pixel-based MLP classifiers over the unsampled domain.

CNN+MLP vs GLCM+MLP classifier

Table 5.7 shows the corresponding confusion matrix of the McNemar's test comparing the performance of the CNN+MLP and GLCM+MLP classifiers evaluated over the sampled domain. A

Table 5.6: Confusion matrix of the McNemar's test comparing the classification performance of CNN+MLP and pixel-based MLP classifiers over the unsampled image tiles 7, 32, and 37.

	CNN+MLP = failure	CNN+MLP = success
Pixel-based MLP = failure	2181855	3672844
Pixel-based MLP = success	539968	5909160

[a] CNN+MLP/Pixel-based MLP = failure/success signifies the samples classified by the CNN+MLP/pixel-based MLP classifier incorrectly/correctly.

z-score of 224.83 favoring the CNN+MLP classifier can be derived from this confusion matrix. The corresponding confidence level (see Table 4.4) for this z-score suggest a strong statistical significance (more than 99% for both the one-tailed and two-tailed prediction). Hence, we reject the equivalent null hypothesis that there is no significant difference between the classification performance of the CNN+MLP and GLCM+MLP classifiers over the sampled domain.

Table 5.7: Confusion matrix of the McNemar's test comparing the classification performance of CNN+MLP and GLCM+MLP classifiers over the sampled image tiles 1, 3, and 5.

	CNN+MLP = failure	CNN+MLP = success
GLCM+MLP = failure	1293329	1093376
GLCM+MLP = success	785214	10932041

[a] CNN+MLP/GLCM+MLP = failure/success signifies the samples classified by the CNN+MLP/GLCM+MLP classifier incorrectly/correctly.

Table 5.8 shows the confusion matrix showing the corresponding confusion matrix of the McNemar's test comparing the performance of the CNN+MLP and GLCM+MLP classifiers evaluated over the unsampled domain. A z-score of 1957.21 favoring the CNN+MLP classifier can be calculated from this confusion matrix. The corresponding confidence level (see Table 4.4) for this z-score suggest a strong statistical significance (more than 99% for both the one-tailed and two-tailed prediction). Hence, we reject the equivalent null hypothesis that there is no significant difference between the classification performance of the CNN+MLP and GLCM+MLP classifiers over the unsampled domain.

Table 5.8: Confusion matrix of the McNemar's test comparing the classification performance of CNN+MLP and GLCM+MLP classifiers over the unsampled image tiles 7, 32, and 37.

	CNN+MLP = failure	CNN+MLP = success
GLCM+MLP = failure	1995464	5680353
GLCM+MLP = success	726359	3901651

[a] CNN+MLP/GLCM+MLP = failure/success signifies the samples classified by the CNN+MLP/GLCM+MLP classifier incorrectly/correctly.

All the results of the McNemar's tests show that the classification performance of the 3 classifiers compared to each other significantly differs. The CNN+MLP classifier continues to outperform the other 2 classifier in both the sampled and unsampled domains, while the GLCM+MLP classifier performs better in the sampled domain as opposed to performing worse in the unsampled

domain when compared against the pixel-based MLP classifier.

5.2.4 Confusion matrix

Sampled domain

Table 5.9 presents the confusion matrix and class F1-scores of the CNN+MLP classifier over the combined predictions on sampled image tiles 1, 3, and 5; with the building class and car class having the highest and lowest F1-score respectively. The average class F1-score is 0.7071.

Table 5.9: Confusion matrix, along with class F1-scores, of the CNN+MLP classifier evaluated over the sampled image tiles 1, 3, and 5.

		Predicted class					
		IS	B	LV	TR	C	NC
Actual class	IS	4419181	336362	162377	51401	2350	0
	B	361121	4804693	42164	9539	670	0
	LV	242479	121652	1313083	188867	77	0
	TR	84898	20693	296851	1465079	0	0
	C	120369	32178	3854	639	23381	0
	NC	0	2	0	0	0	0
F1-score		0.8665	0.9122	0.7128	0.8178	0.2260	Nan

[a] IS = impervious surface, B = building, LV = low vegetation, TR = tree, C = car, NC = clutter classes.
[b] Nan means not a number, undefined (division by zero).

Table 5.10 presents the confusion matrix and class F1-scores of the GLCM+MLP classifier over the combined predictions on sampled image tiles 1, 3, and 5; with the building class and car class having the highest and lowest F1-score respectively. The average class F1-score is 0.6624.

Table 5.10: Confusion matrix, along with class F1-scores, of the GLCM+MLP classifier evaluated over the sampled image tiles 1, 3, and 5.

		Predicted class					
		IS	B	LV	TR	C	NC
Actual class	IS	4232077	475657	217205	39131	7601	0
	B	534725	4584377	70156	17495	11434	0
	LV	270259	164721	1123162	303135	4881	0
	TR	45580	46004	408246	1367460	231	0
	C	107444	40648	3881	723	27725	0
	NC	0	2	0	0	0	0
F1-score		0.8329	0.8708	0.6090	0.7607	0.2387	Nan

[a] IS = impervious surface, B = building, LV = low vegetation, TR = tree, C = car, NC = clutter classes.
[b] Nan means not a number, undefined (division by zero).

Table 5.11 presents the confusion matrix and class F1-scores of the pixel-based MLP classifier over the combined predictions on sampled image tiles 1, 3, and 5; with the building class and car class having the highest and lowest F1-score respectively. The average class F1-score is 0.7034.

Table 5.11: Confusion matrix, along with class F1-scores, of the pixel-based MLP classifier evaluated over the sampled image tiles 1, 3, and 5.

		Predicted class					
		IS	B	LV	TR	C	NC
Actual class	IS	4333601	363743	200345	59896	14086	0
	B	458451	4670056	64146	18890	6644	0
	LV	217159	136198	1202247	305908	4646	0
	TR	61751	32922	300071	1472362	415	0
	C	111534	24526	4604	768	38989	0
	NC	0	1	1	0	0	0
F1-score		0.8536	0.8942	0.6610	0.7905	0.3180	Nan

[a] IS = impervious surface, B = building, LV = low vegetation, TR = tree, C = car, NC = clutter classes.
[b] Nan means not a number, undefined (division by zero).

For all the classifier, the building class seems to be the easiest one to classify; while the car class being the most difficult. We can also notice that the pixel-based MLP classifier outperforms the other 2 classifiers on the most difficult car class. Hence, when we compare the average class F1-scores, the pixel-based MLP classifier also outperforms the GLCM+MLP classifier and only has minute difference of 0.0036 from the CNN+MLP classifier (having the highest average score).

Unsampled domain

Table 5.12 presents the confusion matrix and class F1-scores of the CNN+MLP classifier over the combined predictions on unsampled image tiles 7, 32, and 37; with the impervious surface class and car class having the highest and lowest F1-score respectively. The average class F1-score is 0.6583.

Table 5.12: Confusion matrix, along with class F1-scores, of the CNN+MLP classifier evaluated over the sampled image tiles 7, 32, and 37.

		Predicted class					
		IS	B	LV	TR	C	NC
Actual class	IS	3487798	855408	78410	22153	2329	0
	B	244933	2818011	22072	13006	1269	0
	LV	110483	272119	1494255	425065	6	0
	TR	56104	35525	310723	1756760	0	0
	C	141453	81631	5948	1037	25180	0
	NC	17516	19815	168	37	4613	0
F1-score		0.8202	0.7848	0.7093	0.8027	0.1745	Nan

[a] IS = impervious surface, B = building, LV = low vegetation, TR = tree, C = car, NC = clutter classes.
[b] Nan means not a number, undefined (division by zero).

Table 5.13 presents the confusion matrix and class F1-scores of the GLCM+MLP classifier over the combined predictions on unsampled image tiles 7, 32, and 37; with the tree class and car class

having the highest and lowest F1-score respectively. The average class F1-score is 0.3791.

Table 5.13: Confusion matrix, along with class F1-scores, of the GLCM+MLP classifier evaluated over the unsampled image tiles 7, 32, and 37.

		Predicted class					
		IS	B	LV	TR	C	NC
Actual class	IS	1248568	3089114	57815	50339	262	0
	B	30287	3025171	5988	37814	31	0
	LV	12800	539686	197895	1551533	14	0
	TR	2889	120555	62352	1973316	0	0
	C	61737	181798	4774	2762	4178	0
	NC	10745	30606	661	137	0	0
F1-score		0.4296	0.5999	0.1504	0.6834	0.0322	Nan

[a] IS = impervious surface, B = building, LV = low vegetation, TR = tree, C = car, NC = clutter classes.
[b] Nan means not a number, undefined (division by zero).

Table 5.14 presents the confusion matrix and class F1-scores of the pixel-based MLP classifier over the combined predictions on sampled image tiles 7, 32, and 37; with the building class and car class having the highest and lowest F1-score respectively. The average class F1-score is 0.2599.

Table 5.14: Confusion matrix, along with class F1-scores, of the pixel-based MLP classifier evaluated over the sampled image tiles 7, 32, and 37.

		Predicted class					
		IS	B	LV	TR	C	NC
Actual class	IS	1005407	2363160	575878	638429	5302	0
	B	211232	2366793	131512	653849	898	0
	LV	242352	755430	98510	659993	501	0
	TR	235258	830647	86765	1156829	320	0
	C	50108	125382	50740	31351	471	0
	NC	3782	3382	2080	17466	0	0
F1-score		0.3173	0.4826	0.0729	0.4231	0.0035	Nan

[a] IS = impervious surface, B = building, LV = low vegetation, TR = tree, C = car, NC = clutter classes.
[b] Nan means not a number, undefined (division by zero).

In the case of classifying unsampled image tiles, each classifier has its own class with the highest F1-score: the impervious surface class for the CNN+MLP classifier; the tree class for the GLCM+MLP classifier; and, the building class for the pixel-based MLP classifier. The car class continues to be the most difficult class. There's also a huge drop (as seen in the overall accuracy results, subsection 5.2.2), comparing the sampled and unsampled domains, in average F1-scores of the pixel-based MLP and GLCM+MLP classifiers; as oppose to a relatively small drop of 0.0488 in the average F1-score of the CNN+MLP classifier.
The difficulty in classifying the car class can be both attributed to: 1) the stratified sampling method used using linear proportions of the frequencies of the classes found in each image tile,

and 2) the inherent difficulty of classifying this class given the data (near infrared, optical bands, and a DSM). The car class being the least occurring class will result in training sets with very few car samples. The near infrared and optical bands may not be that helpful in classifying the car class as cars could vary in color—hence, will also have varying responses in this bands. The regular DSM may also be not that helpful as a car on a lower elevation could have the same value as an impervious surface of higher elevation; same car objects will also have varying DSM values depending on their location. A relatively higher car class F1-score of the pixel-based MLP classifier in the sampled domain could be possibly due to it having less parameter—hence, it will be less prone to underfitting the undersampled car class as oppose to the more complicated CNN+MLP and GLCM+MLP classifiers. A normalized DSM, as used by Paisitkriangkrai and Sherrah (2015) and Lagrange and Saux (2015), (providing heights relative to the ground) could help in classifying this car class. Changing the sampling scheme, such that distribution of the training samples among the classes are more balanced, can also help to improve the classification of the car pixels. One way to implement such a sampling scheme is to take the proportions of the logarithm of the frequency of the classes. We leave the study of this possible improvements in the methods for future works.

5.2.5 Other metrics

Lastly, we report the performance of the 3 classification approaches on metrics with more qualitative inclination. We compare the quality of the resulting classified maps of each classifier; and briefly discuss the computational time and complexity of the algorithms that makes up the classification approaches.

Map quality

To compare the quality of the classified maps from the 3 classifiers, we present in Figure 5.12 a zoomed portion of the classified maps and ground truth of image tile 3.

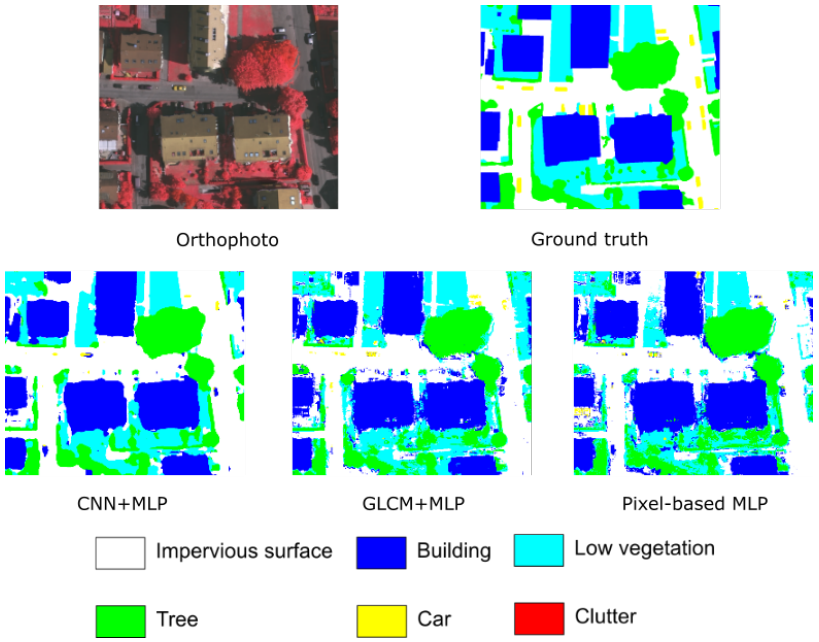


Figure 5.12: A subset of the resulting classified map of image tile 3 with the equivalent orthophoto and ground truth data.

We can clearly see that the CNN+MLP classifier presents a smoother more regularized classification. This figure also confirms the observation of low F1-scores of the car class, as most car pixels are greatly misclassified by all of the classifiers. Fully classified maps (along with the filters learned by the CNN+MLP classifier trained with the largest training sample set) of all the sampled and unsampled tiles are attached in appendix A.

On computational time and complexity

This is one metric for which the CNN+MLP classifier is clearly at a disadvantage. Training an instance of the CNN+MLP classifier, while only performing holdout validation over the learning and regularization hyperparameters, can take up to 2.5 days (when using the largest sample size). The training time could have been further prolonged drastically if a full model selection—varying also the hyperparameters defining the architecture of the network such as depth, etc.—was performed. However for both the pixel-based and GLCM+MLP classifiers, it only took maximum of 6 hours to train. For the case of GLCM, we can also add another extra hour for each tile to extract the handcrafted GLCM features. Both training (given a fixed batch size as applied in this study) and testing time approximately linearly scales with the number of training samples.
The resulting CNN+MLP classifier is also more computationally complex than the other 2. It has more than 300,000 parameters each applying multiplication followed by a summation (plus pooling and activation) in each unit of the succeeding layer. This could scale up to more than 40,000,000 operations for a single forward pass of classifying an example. On the other hand, the pixel-based MLP and GLCM+MLP classifiers only has around 18,000 parameters each applying multiplication followed by a summation in each unit of the succeeding layer. And would only scale to almost the same number of operations (more than 18,000) for single forward pass of classifying an example. For the GLCM+MLP classifier, calculating the GLCM features adds up (approximately) additional 500,000 operations. All in all, computational time and complexity is definitely a downside of using the CNN+MLP classifier.

5.3 FINAL RECAP

In summary, we present in this chapter the results of each design and performance analysis we have performed. In the initial experiments, we find the advantages of using $relu$ activation function over sigmoid and applying regularization and initialization techniques. We also observe and interpret the sensitivity of the CNN to several of its hyperparameters: patch size, kernel size, number of filters, and network depth. Generally, with the number of training and test samples (independently) fixed for each sensitivity experiment, we find that (except for the depth) the classification accuracy peaks—at a certain value of hyperparameter—as we increase the value of the hyperparameter; and the classification accuracy decreases from thereon, after further increasing values of the hyperparameter. We also find the superiority of the classifier of our primary interest (CNN+MLP) over the other 2 classifiers (pixel-based MLP and GLCM+MLP) in 5 of the 6 performance measure we considered—with only computational time and complexity being the disadvantage of CNN+MLP.
With the results of the research done discussed in this chapter, we conclude this thesis in the succeeding final chapter. The next chapter goes back to research questions posed in chapter 1; at the same time, addressing each questions based on insights gained from all the experiments performed, results analyzed, and interpretations discussed in this work.

Chapter 6
Conclusion and Future developments

6.1 CONCLUSION

In this study, we designed, analyzed, and evaluated a deep feature learning approach to the classification of very high resolution aerial images. The meaning and effect of several hyperparameters to the performance of classifier was investigated. From the knowledge obtain in these design analysis, we came up with the architecture of the classifier of our primary interest (CNN+MLP classifier). This deep feature learning based classifier automatically extracts and learns spatial-contextual features, from the data, (see the filters learned by the CNN+MLP classifier used to extract these spatial-contextual features in appendix B) useful for the classification problem at hand. We evaluated the CNN+MLP classifier against two other classification approaches: 1) an approach using individual pixel values (pixel-based MLP classifier), and 2) an approach using handcrafted spatial-contextual features (GLCM+MLP classifier). Several performance analyses—based on overall classification accuracy, statistical difference in accuracy, class F1-scores, accuracy gained from additional training samples, and map quality—shows that the CNN+MLP classifier outperforms the other 2 approaches. The only downside the CNN+MLP classifier is the greater computational time and complexity required when using it. This computational downside can be attributed to the fact that aside from classification, the CNN+MLP classifier is also learning and extracting useful spatial-contextual features for discriminating the classes of interest. Such additional task will definitely add up to the complexity and computational resources required by the classifier.
In this section, we also present our answers to the research questions posed in the first chapter:

1. How does deep learning algorithms (e.g. convolutional neural networks, autoencoders, and Boltzmann machine variants) work in a remote sensing context?

   In chapter 2, we discussed these most well-known deep learning algorithms. Convolutional neural networks (CNN) can be both used as a classifier in a supervised classification problem; and can also be used to learn features in an unsupervised manner, features that can be further used by another classification algorithm. The convolutional layers of these networks also allows us to capture contextual information relevant to the classification problem at hand. Autoencoders and Boltzmann machine variants can also be used to learn features in an unsupervised manner; and can also be used to initialize the weights of a supervised deep neural network. The (regression or classification) problems in remote sensing hardly differ, in a general sense, from those in other domains (computer vision, speech, natural language processing, etc.) where deep learning is applied. But comparing image classification specifically, the difference in scale between usual computer vision and remote sensing applications translates to difference in objects of interest—and hence, the features that needs to be learn and the classification level (i.e. labeling each image or each pixel) will be different.

2. Which deep learning algorithm is suited to classifying very high resolution airborne images of urban areas with sub-decimeter resolution?

With the properties of the popular algorithms concisely reviewed in chapter 2, we chose the convolutional neural network as the foundation of our CNN+MLP classification approach. Ground truth images of the dataset used in this study allowed us to train and evaluate these networks in a supervised manner.

3. What are the effects of varying the network architecture (e.g. feed-forward, recurrent) and dimensions (e.g. number of hidden layers, number of neurons in the hidden layers) to the performance of the classifier?

   Modifying some of the hyperparameters of the network such as input patch size, kernel size, and number of filters effectively alters the number of neurons in a hidden layer; while increasing the number of convolutional and dense layer of the CNN equivalently increases the number of hidden layers in the network. The results of the sensitivity analysis experiments (see subsection 5.1.2) shows the effects of these variations. In general, when the training set size is fixed, increasing the values of the hyperparameters affecting the number of neurons in a hidden layer also increases the overall accuracy until reaching a peak (with a certain hyperparameter value), then the accuracy drops afterwards. On the other hand, increasing both the number of convolutional and dense layers does not seem to improve the classification accuracy (similar to what Zhao and Du (2016) observed). Absolute comparison of the sensitivity of the CNN among its hyperparameters is irrelevant since training and test sets were independently fixed for each experiment. The choice of this hyperparameters for future implementations will mostly depend on the use case (classification problem) and the scale of the objects of interest in such a use case. For the problem discussed in this study, with sufficient number of samples, a patch size of about twice the dimension of the smallest object of interest (car width which is around 15 pixels) is sufficient to obtain good classification results (around 85% overall accuracy for the sampled domain).

   We showed in the experiment using a recurrent convolutional neural network accepting 121 x 121 input patches that recurrence in architecture can produce a smoother more regularized map (see Figure 5.10). The built-in contextual label and pixel dependency in the recurrent convolutional neural network can enable us, with sufficient training samples, to learn both the informative features and appropriate "smoothing factor" to produce an accurate and spatially regularized map.

4. What are the effects of initialization and regularization (e.g. dropout, early stopping, recurrence) techniques to the performance of the classifier?

   Results of the initial experiments in subsection 5.1.1 shows that a poor choice of initial weights of the network (e.g. initializing the weights with values of 1) can prevent the network from learning anything. The network weights must therefore be initialized in an appropriate manner. In this study, we adopted the uniform initialization technique suggested by Glorot and Bengio (2010). Same subsection 5.1.1 shows the effects and the need of regularization techniques to prevent the networks from overfitting.

5. What performance measures (e.g. classification accuracy, computational complexity, level of automation, training sample size) are relevant for assessing the classifier?

   In this study, we evaluated the CNN+MLP classifier along with the 2 other classification approaches using several performance measures: the training set size, overall classification accuracy, statistical difference in accuracy, class F1-scores, map quality, and computational time and complexity. The first 4 measures were assessed in a quantitative manner, while the last 2 were discussed with a qualitative inclination.

6. Can the classifier generalize well within a domain adaptation setting, where training and test samples are taken from different images but with similar characteristics?

   Several results in section 5.2 shows the superiority of the CNN+MLP classifier, when classifying unsampled image tiles, compared to the pixel-based MLP and GLCM+MLP classifiers. However, we still observe a considerable drop in overall accuracy and average class F1-score from the results of the 3 classifiers in this domain adaption experiment setup.

7. Which approach performs better and in which aspect of performance measure?

   Except for computational time and complexity, the CNN+MLP classification approach performed better than the other 2 approaches. The McNemar's test performed also have shown that the classification accuracy of CNN+MLP is statistically better (with more than 99% confidence level) than the classification accuracy of the other two classifiers.

8. How much does the performance of the feature learning and feature engineering approaches differ?

   For the overall classification accuracy results over image tile 1 while varying the training set size (see subsection 5.2.1), the CNN+MLP classifier generally outperforms the pixel-based MLP classifier for about almost 3% in average and also outperforms the GLCM+MLP classifier for more than 1% in average. See section 5.2 for the results of the other metrics.

We presented in this study how deep learning methods can play a role in the classification of aerial images with sub-decimeter resolution. Several architectural elements of the algorithms involved were investigated and interpreted. We also observed the gains in performance when using the deep learning based classifier and tackled the corresponding computational downside. All in all, we can see from this work the promise of deep learning that with enough training data—it can replace handcrafted rules for solving complex problems.

6.2 FUTURE DEVELOPMENTS

Finally, here is our list of recommendations for future research work:

- Further study of the recurrent convolution neural network (RCNN) architecture. We expect that when trained with a sufficient number of training samples and proper model selection, an RCNN can learn the appropriate "smoothing factor" resulting in a smooth and accurate classified map. A well-tuned RCNN can be substituted to the standard CNN as the classifier of our primary interest in the performance analysis experiments we have done. The initial score maps can be learned as well (similar to the network weights), instead of having zero values the whole time.

- Investigate how unsupervised pre-training of a (recurrent) convolutional neural network can affect its classification performance.

- Test a "better" sampling (see discussions in subsection 5.2.4) and normalization (see discussions in subsection 5.2.2) scheme.

- Performing the sensitivity analysis with varying (larger) training set size.

- Apply the classification approach to a problem dealing with classes of higher abstraction, e.g. classification of land use instead of land covers.

LIST OF REFERENCES

Ackley, D., Hinton, G., and Sejnowski, T. (1985). A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169.

Adamczyk, J. and Osberger, A. (2015). Red-edge vegetation indices for detecting and assessing disturbances in Norway spruce dominated mountain forests. *International Journal of Applied Earth Observation and Geoinformation*, 37:90–99.

Bekkari, A., Idbraim, S., Mammass, D., and El Yassa, M. (2011). Exploiting spectral and space information in classification of high resolution urban satellites images using Haralick features and SVM. *International Conference on Multimedia Computing and Systems -Proceedings*, pages 1–4.

Benediktsson, J., Swain, P., and Ersoy, O. (1990). Neural network approaches versus statistical methods in classification of multisource remote sensing data. *IEEE Transactions on Geoscience and Remote Sensing*, 28(4):540–552.

Benenson, R. (2014). What is the class of this image ? Retrieved from: `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html` at 2015-10-01.

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy Layer-Wise Training of Deep Networks. *Advances in neural information processing systems*, 19(1):153.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bostanci, B. and Bostanci, E. (2013). An evaluation of classification algorithms using mcnemar's test. In *Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)*, pages 15–26. Springer India.

Casado, R. and Younas, M. (2015). Emerging trends and technologies in big data processing. *Concurrency Computation Practice and Experience*, 27(8):2078–2091.

Chen, X., Xiang, S., Liu, C.-L., and Pan, C.-H. (2013). Vehicle Detection in Satellite Images by Parallel Deep Convolutional Neural Networks. *2013 2nd IAPR Asian Conference on Pattern Recognition*, pages 181–185.

Chen, X., Xiang, S., Liu, C.-L., and Pan, C.-H. (2014a). Aircraft Detection by Deep Convolutional Neural Networks. *IPSJ Transactions on Computer Vision and Applications*, 7(0):10–17.

Chen, Y., Lin, Z., Zhao, X., Wang, G., and Gu, Y. (2014b). Deep Learning-Based Classification of Hyperspectral Data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(6):1–14.

Cho, Y. and Saul, L. K. (2009). Kernel Methods for Deep Learning. *Advances in Neural Information Processing Systems*, pages 342–350.

Clark, A. F. and Clark, C. (1999). Performance characterization in computer vision a tutorial.

Cramer, M. (2010). The DGPF-Test on Digital Airborne Camera Evaluation — Overview and Test Design. *Photogrammetrie - Fernerkundung - Geoinformation*, 2:73–82.

Deichmann, U., Ehrlich, D., Zeug, G., and Small, C. (2011). Using High Resolution Satellite Data for the Identification of Urban Natural Disaster Risk. Retrieved August 9, 2015 from `http://ccsl.iccip.net/using_high_resolution_data.pdf`.

Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3(January):e2.

Deng, L. and Yu, D. (2011). Deep convex network: A scalable architecture for speech pattern classification. In *Interspeech*. International Speech Communication Association.

Department of economic and social affairs of the united nations population division (UNPD) (2014). World Urbanization Prospects: The 2014 Revision [highlights]. Technical report, United Nations, New York. Retrieved from: `http://esa.un.org/unpd/wup/Highlights/WUP2014-Highlights.pdf`.

Dewan, A. M. and Yamaguchi, Y. (2009). Land use and land cover change in Greater Dhaka, Bangladesh: Using remote sensing to promote sustainable urbanization. *Applied Geography*, 29(3):390–401.

Dorigo, W., Lucieer, A., Podobnikar, T., and Čarni, A. (2012). Mapping invasive fallopia japonica by combined spectral, spatial, and temporal analysis of digital orthophotos. *International Journal of Applied Earth Observation and Geoinformation*, 19:185 – 195.

Du, K.-L. and Swamy, M. N. S. (2014). *Neural Networks and Statistical Learning*. Springer-Verlag, London.

Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929.

Foody, G. (2004). Thematic map comparison: evaluating the statistical significance of differences in classification accuracy. *Photogrammetric Engineering and Remote Sensing*, 70(5):627–633.

Gleason, J., Nefian, A. V., Bouyssounousse, X., Fong, T., and Bebis, G. (2011). Vehicle detection from aerial imagery. *2011 IEEE International Conference on Robotics and Automation*, pages 2065–2070.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:249–256.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. Book in preparation for MIT Press. Version accessed from: `http://www.deeplearningbook.org/version-2016-01-30/index.html`.

Gressin, A., Vincent, N., Mallet, C., and Paparoditis, N. (2014). A unified framework for land-cover database update and enrichment using satellite imagery. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 5057–5061.

Han, J., Zhang, D., Cheng, G., Guo, L., and Ren, J. (2015). Object Detection in Optical Remote Sensing Images Based on Weakly Supervised Learning and High-Level Feature Learning. *IEEE Transactions on Geoscience and Remote Sensing*, 53(6):3325–3337.

Haralick, R., Shanmugan, K., and Dinstein, I. (1973). Textural features for image classification.

Hinton, G., Osindero, S., and Teh, Y. W. (2006a). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–54.

Hinton, G., Osindero, S., Welling, M., and Teh, Y. W. (2006b). Unsupervised discovery of non-linear structure using contrastive backpropagation. *Cognitive science*, 30(4):725–731.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

ISPRS (2015). 2D Semantic Labeling - Vaihingen data. Retrieved from: `http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-vaihingen.htmlat2016-02-15`.

Ju, J., Gopal, S., and Kolaczyk, E. D. (2005). On the choice of spatial and categorical scale in remote sensing land cover classification. *Remote Sensing of Environment*, 96(1):62–77.

Ke, Y., Quackenbush, L. J., and Im, J. (2010). Synergistic use of QuickBird multispectral imagery and LIDAR data for object-based forest species classification. *Remote Sensing of Environment*, 114(6):1141–1154.

Kemper, T., Mudau, N., Mangara, P., and Pesaresi, M. (2015). Towards an automated monitoring of human settlements in South Africa using high resolution SPOT satellite imagery. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-7/W3(May):1389–1394.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105.

Kumar, R., Jayaraman, V. K., and Kulkarni, B. D. (2005). An SVM classifier incorporating simultaneous noise reduction and feature selection: Illustrative case examples. *Pattern Recognition*, 38(1):41–49.

Lagrange, A. and Saux, B. L. (2015). Convolutional Neural Networks for Semantic Labeling. Technical report, ISPRS.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the 26th Annual International Conference on Machine Learning*, 2008:1–8.

Li, J. and Narayanan, R. M. (2004). Integrated Spectral and Spatial Information Mining in Remote Sensing Imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 42(3):673–685.

Li, M., Zang, S., Zhang, B., Li, S., and Wu, C. (2014). A review of remote sensing image classification techniques: The role of Spatio-contextual information. *European Journal of Remote Sensing*, 47(1):389–411.

Maire, F., Mejias, L., and Hodgson, A. (2014). A Convolutional Neural Network for Automatic Analysis of Aerial Imagery. In *Digital Image Computing: Techniques and Applications*.

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Mnih, V. and Hinton, G. E. (2010). Learning to detect roads in high-resolution aerial images. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6316 LNCS(PART 6):210–223.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. a., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Murphy, K. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Onojeghuo, A. O. and Blackburn, G. A. (2011). Mapping reedbed habitats using texture-based classification of quickbird imagery. *International Journal of Remote Sensing*, 32(23):8121–8138.

Pacifici, F., Chini, M., and Emery, W. J. (2009). A neural network approach using multi-scale textural metrics from very high-resolution panchromatic imagery for urban land-use classification. *Remote Sensing of Environment*, 113(6):1276–1292.

Paisitkriangkrai, S. and Sherrah, J. (2015). Effective Semantic Pixel labelling with Convolutional Networks and Conditional Random Fields. Technical report, ISPRS.

Pinheiro, P. and Collobert, R. (2014). Recurrent convolutional neural networks for scene labeling. *Proceedings of The 31st International Conference on Machine Learning*, 32(June):82–90.

Posner, I., Cummins, M., and Newman, P. (2009). A generative framework for fast urban labeling using spatial and temporal context. *Autonomous Robots*, 26(2-3):153–170.

Qi, L., Yong, D., Xin, N., Jiaqing, X., and Baoliang, L. (2014). Classification of land cover based on deep belief networks using polarimetric radarsat-2 data. *Geoscience and Remote Sensing Symposium*, pages 4679–4682.

Ranzato, M., Huang, F. J., Boureau, Y. L., and LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Ranzato, M. A. and LeCun, Y. (2007). A sparse and locally shift invariant feature extractor applied to document images. *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, 2:1213–1217.

Richards, J. A. (2013). *Remote Sensing Digital Image Analysis*. Springer, Heidelberg, 5th edition.

Rumelhart, D. E., Hinton, G., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Letters to Nature*, 323:533–536.

Saito, S. and Aoki, Y. (2015). Building and road detection from large aerial imagery. In *Proceedings of SPIE - The International Society for Optical Engineering*, volume 9405.

Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann Machines. *Artificial Intelligence*, 5(2):448–455.

Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1:1–14.

Somers, B. and Asner, G. P. (2013). Multi-temporal hyperspectral mixture analysis and feature selection for invasive species mapping in rainforests. *Remote Sensing of Environment*, 136:14–27.

Song, H., Xu, R., Ma, Y., and Li, G. (2013). Classification of ETM+ Remote Sensing Image Based on Hybrid Algorithm of Genetic Algorithm and Back Propagation Neural Network. *Mathematical Problems in Engineering*, 2013:1–8.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR)*, 15:1929–1958.

Tang, J., Deng, C., Huang, G.-B., and Zhao, B. (2015). Compressed-Domain Ship Detection on Spaceborne Optical Image Using Deep Neural Network and Extreme Learning Machine. *Geoscience and Remote Sensing, IEEE Transactions on*, 53(3):1174–1185.

Taylor, G., Hinton, G., and Roweis, S. (2007). Modeling human motion using binary latent variables. *Advances in neural information processing systems*, 19:1345.

Tieleman, T. (2008). Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient. *Proceedings of the 25th International Conference on Machine Learning*, 307:7.

Tuia, D., Flamary, R., and Courty, N. (2015). Multiclass feature learning for hyperspectral image classification: Sparse and hierarchical solutions. *ISPRS Journal of Photogrammetry and Remote Sensing*, 105:272–285.

Verheyden, A., Dahdouh-Guebas, F., Thomaes, K., De Genst, W., Hettiarachchi, S., and Koedam, N. (2002). High-resolution vegetation data for mangrove research as obtained from aerial photography. *Environment, Development and Sustainability*, 4(2):113–133.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1096–1103.

Wang, J., Song, J., Chen, M., and Yang, Z. (2015). Road network extraction: a neural-dynamic framework based on deep learning and a finite state machine. *International Journal of Remote Sensing*, 36(12):3144–3169.

Warner, T. A., Nellis, M. D., and Foody, G. M. (2009). *The SAGE Handbook of Remote Sensing*. Sage Publications Ltd, Los Angeles.

Yu, Y., Li, J., Member, S., Guan, H., Jia, F., and Wang, C. (2015). Learning Hierarchical Features for Automated Extraction of Road Markings From 3-D Mobile LiDAR Point Clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(2):709–726.

Zhao, W. and Du, S. (2016). Learning multiscale and deep representations for classifying remotely sensed imagery. *{ISPRS} Journal of Photogrammetry and Remote Sensing*, 113:155 – 165.

Appendix A
Classified Maps

In this appendix, we present the classified maps of the 3 classification approaches together with the corresponding orthophotos and ground truth data.

A.1 SAMPLED DOMAIN

Figure A.1 shows the maps of image tile 1.

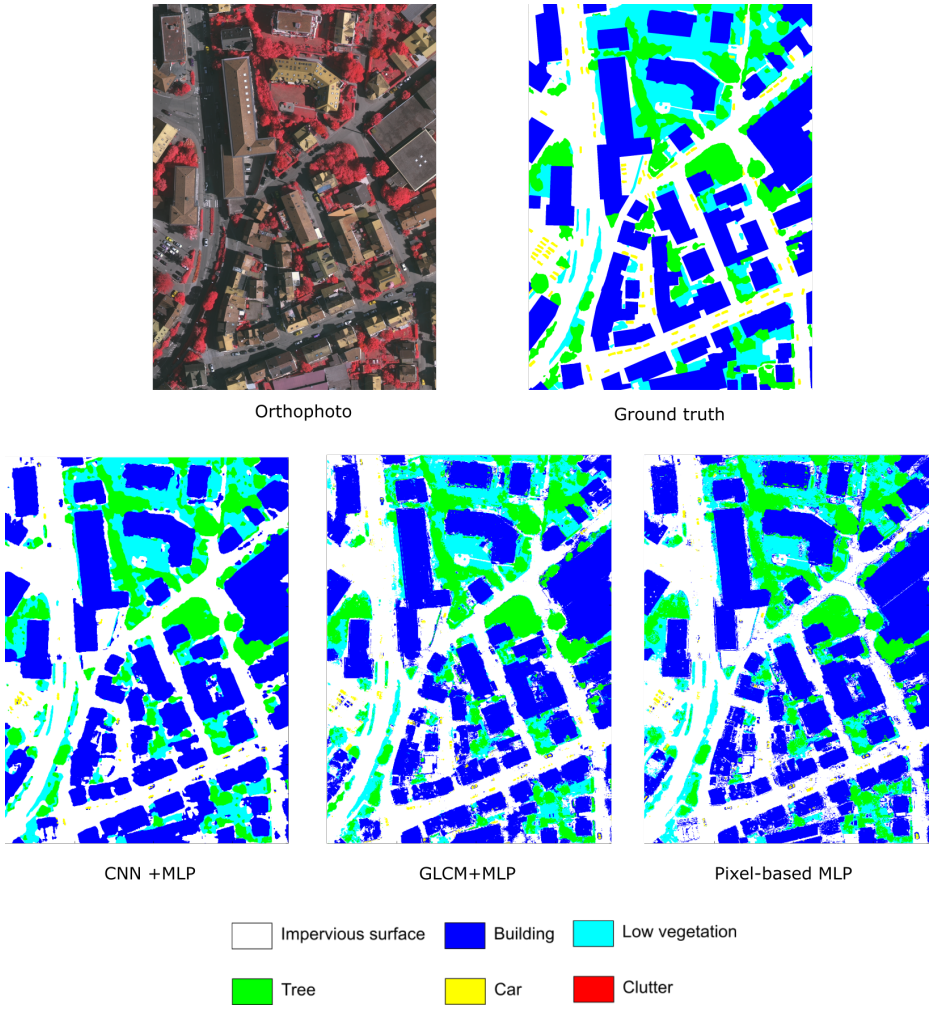


Figure A.1: Full classified map with orthophoto and ground truth image of image tile 1.

Figure A.2 shows the maps of image tile 3.
Figure A.3 shows the maps of image tile 5.

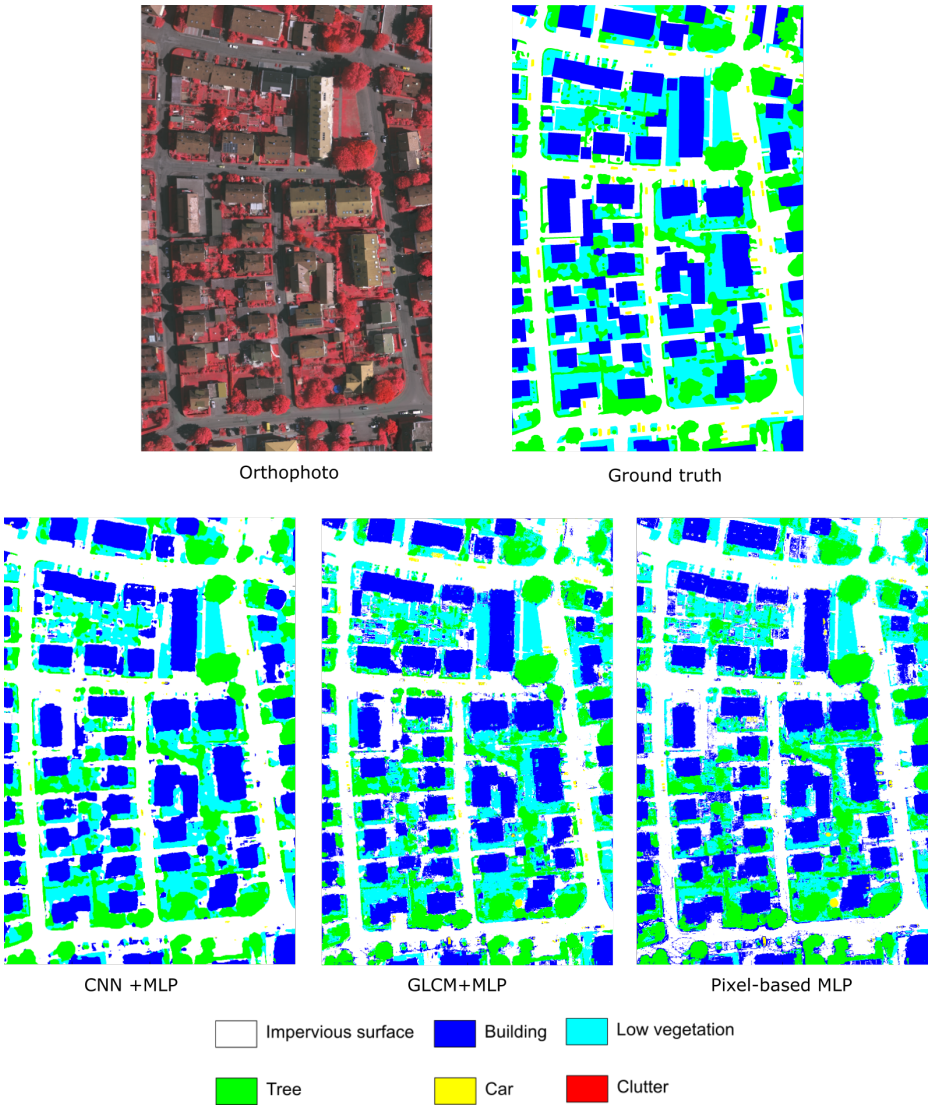


Figure A.2: Full classified map with orthophoto and ground truth image of image tile 3.

A.2 UNSAMPLED DOMAIN

Figure A.4 shows the maps of image tile 7.
Figure A.5 shows the maps of image tile 32.
Figure A.6 shows the maps of image tile 37.

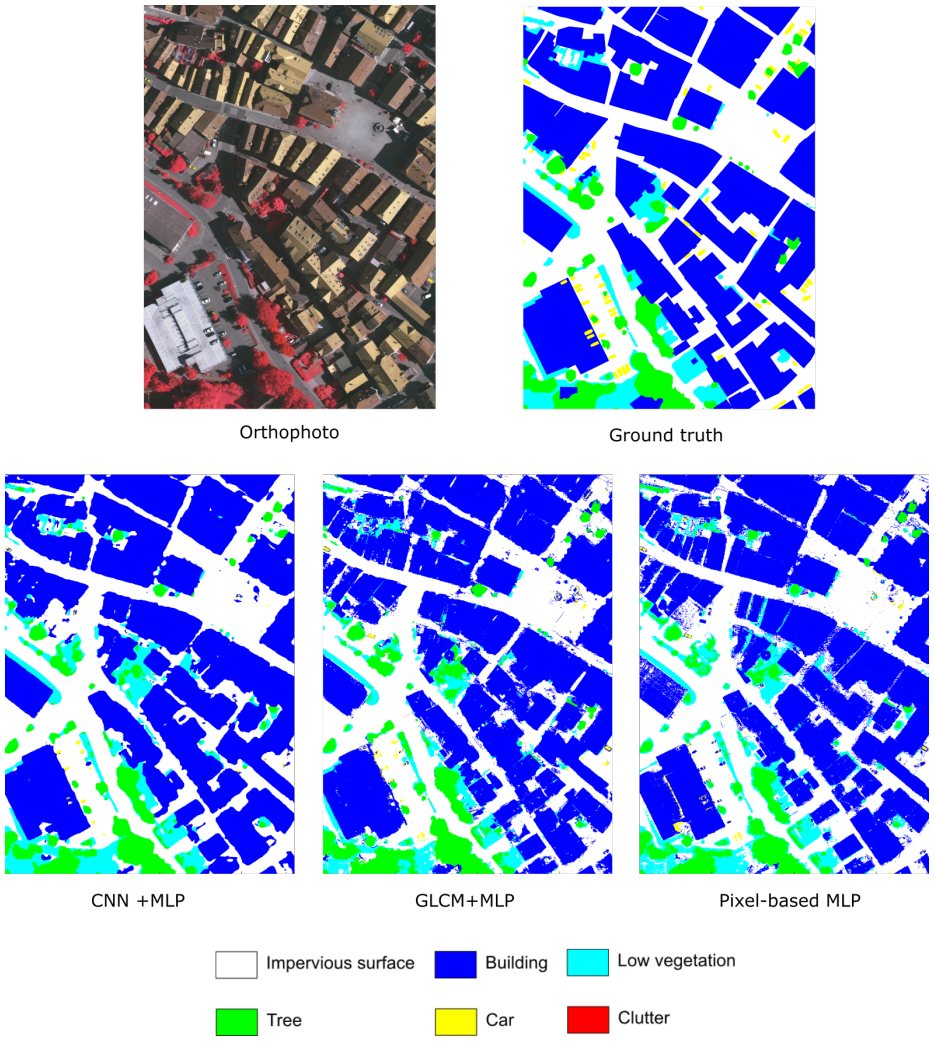


Figure A.3: Full classified map with orthophoto and ground truth image of image tile 5.

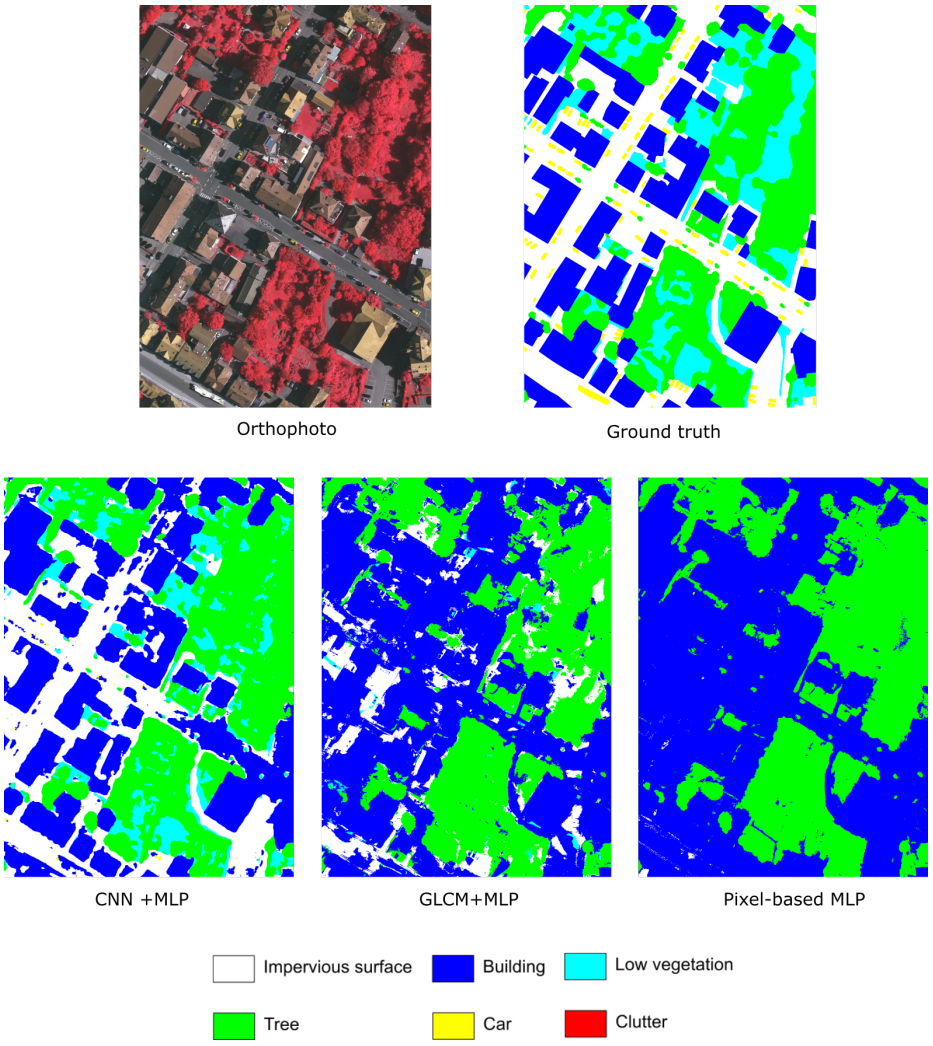


Figure A.4: Full classified map with orthophoto and ground truth image of image tile 7.

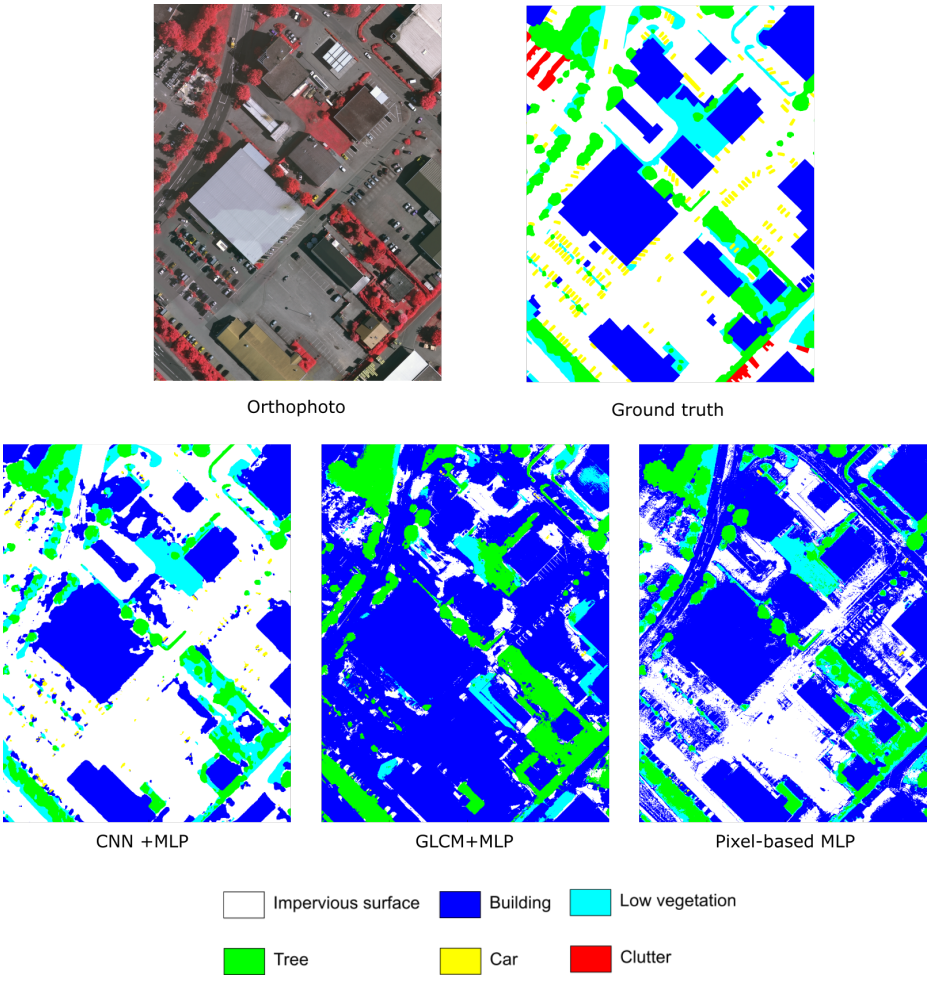


Figure A.5: Full classified map with orthophoto and ground truth image of image tile 32.

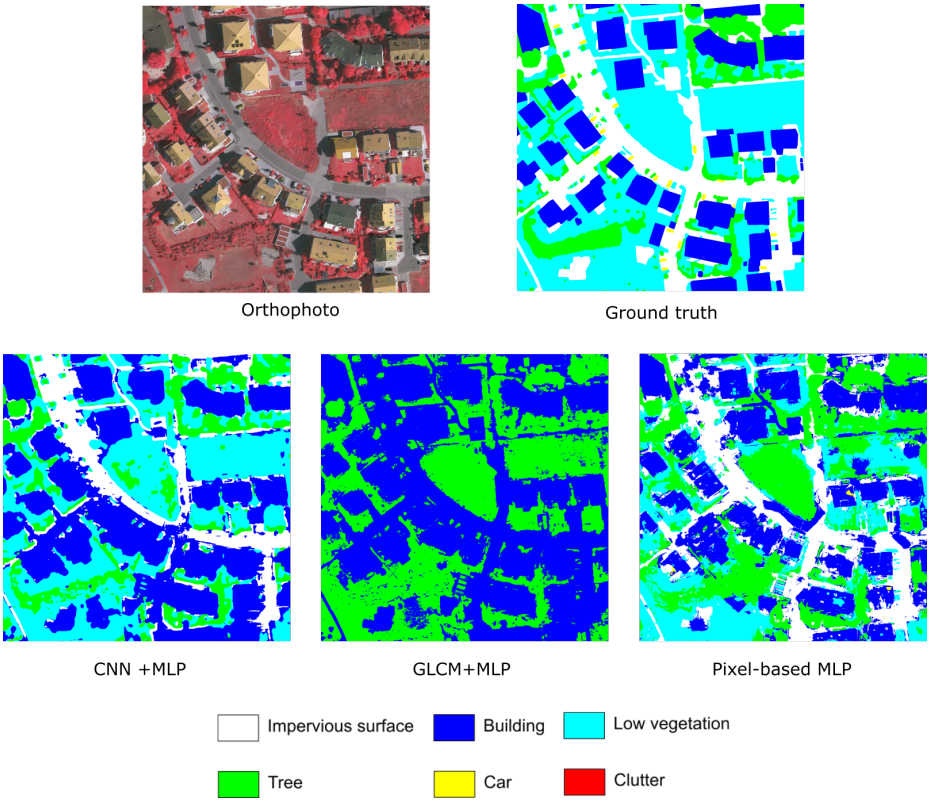
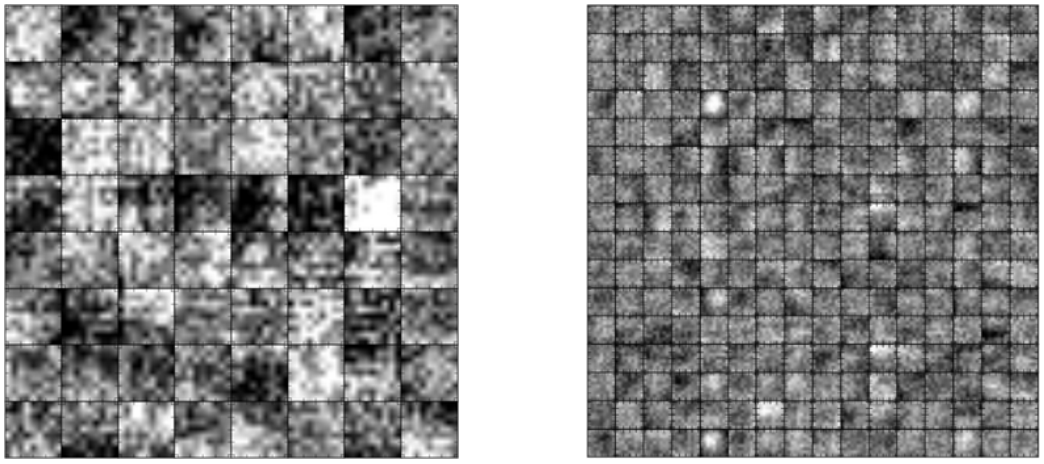


Figure A.6: Full classified map with orthophoto and ground truth image of image tile 37.

Appendix B
Filters Learned

In this appendix, we present the convolutional filters learned by the CNN+MLP classifier trained with the largest training set. Figure B.1 shows the filters learned of the 1^{st} and 2^{nd} convolutional layers of the CNN+MLP classifier.



(a) First layer (c) Second layer

Figure B.1: The filters learned by the first convolutional layer (a) and second convolutional layer (b) of CNN+MLP classifier trained with the largest training sample set.