Czech Technical University in Prague

&

Luleå University of Technology

Faculty of Electrical Engineering
Department of Cybernetics

MASTER'S THESIS

Semantic Segmentation of Satellite Images using Deep Learning

Shivaprakash Muruganandham

supervised by
Ing. Michal Reinštein, PhD.

16th *August*, 2016

Preface

First, I want to thank my thesis supervisor, Ing. Michal Reinštein, Ph.D for his guidance and feedback, as well as for providing me with the resources to carry out this work. His advice has been invaluable in helping me gain a deeper understanding of the subject. I also thank Vladimír Kubelka for his patience and assistance in answering my questions. Many thanks to Nikita and Abhishek for helping me proofread the thesis, and for their advise on designing it. Thanks also to Gabriel Blindell for making the document template used in this work publicly available.

Last but not the least, I thank my parents for their never-ending support throughout my studies, without which none of this would have been possible.

This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Abstract

A stark increase in the amount of satellite imagery available in recent years has made the interpretation of this data a challenging problem at scale. Deriving useful insights from such images requires a rich understanding of the information present in them.

This thesis explores the above problem by designing an automated framework for extracting semantic maps of roads and highways to track urban growth of cities in satellite images. Devising it as a supervised machine learning problem, a deep neural network is designed, implemented and experimentally evaluated. Publicly available datasets and frameworks are used for this purpose. The resulting pipeline includes image pre-processing algorithms that allows it to cope with input images of varying quality, resolution and channels.

Additionally, we review a computational graph approach to building a neural network using the TensorFlow [1] framework.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter lays the motivation for this thesis work, and the objectives therein. We then briefly glimpse at a high level overview of the report structure.

1.1 MOTIVATION

In order to make informed decisions pertaining to the environment, we are equipped with senses that allow us to observe it. This enables us to make effective changes around us as appropriate or desirable. By taking a step back and extending this general process to the large scale, we notice a need for understanding complex phenomena around the world such as urban growth, climate change, biodiversity studies and socioeconomic trends. This process, very generally is referred to as earth observation, and has applications in disaster response, resource management and precision farming among others.

Earth observation data is gathered by a range of techniques, and can be roughly categorized as remote and proximal (sometimes referred to as *in-situ*) sensing. The former is where "the distance between the object and the sensor far surpasses the linear dimensions of the sensor", while the latter is where this distance is comparable to the linear sensor dimensions [42].

Recent technological advances in microelectronics have also spiraled into the satellite manufacturing industry. The miniaturization of space grade components has resulted in the rise of small satellites, including a great number of remote sensing satellites. With diminishing launch and manufacturing costs, this has led to a democratized access to space. In turn, satellite imaging (a subset of remote sensing) has experienced an increase in interest and demand over the last few years, with imagery thus far available only to very few research communities becoming much more publicly available.[1] Adding to this, the rise of new markets has driven commercial satellite imaging away from mere pixel pushing to content providing - wherein the strength of the imagery lies in how insightful it is.

Historically, manual analyses of satellite and aerial imagery was feasible primarily because the volume of images available was quite low - but that is not the case now. Relevant information extraction from images thus becomes a problem with the high volume of data we deal with today. A major component of these problems is annotation (or labeling), wherein one identifies the structures and patterns visible in a satellite image.

Over the years, research in the computer vision community has addressed this problem of automating the analysis of large scale data in different ways, briefly touched upon in Chapter 2. Machine learning techniques have proven to be strong candidates here, especially in the last few years. At the time of writing, the state of the art in the automation of visual labeling tasks is seen in the deep learning research community, and that is where this thesis picks up at.

Machine learning research stems from the idea that a computer can be given the ability to learn, as a human would do, without being explicitly programmed.[36] Deep learning is a subset of machine learning, and refers to the application of a set of algorithms called neural networks, and their variants. In such methods, one provides the network (or model) with a set of labeled examples which it learns, or trains on. [2] Labeling these examples is done in many ways. Collaborative platforms such as OpenStreetMap and crowdsourcing marketplaces[3] are ideal for the annotation of images, and this existing volume can already be leveraged.

1.2 OBJECTIVES AND SCOPE

Given the above background, the main goal of this thesis is to design, implement and experimentally evaluate a deep neural network for the semantic

[1]Most prominently in 2008, when the US Geological Survey (USGS) made over 35 years of Landsat imagery publicly available

[2]This particular process is called supervised learning. Unsupervised learning, on the other hand, is characterized by training on unlabeled examples. In such cases, one deploys an autoencoder, beyond the scope of this work.

[3]Amazon's Mechanical Turk marketplace, for instance, has been used to identify and classify objects in satellite images

segmentation of man-made structures in satellite images. Semantic segmentation is one of different problems in computer vision, and is introduced in more detail in Chapter 2.

Understanding that urban growth and infrastructure expansion is highly correlated with roads and highways, we deal primarily with the segmentation of roads, as developed in [28]. The resulting pipeline shall include image pre-processing algorithms to cope with input images of varying quality, resolution, and channels. To this end, the objectives can be defined as:

1. Undertake a brief study on neural network techniques for computer vision.
2. Build a working deep network pipeline that takes in data to produce semantically segmented maps on the images.
3. Compare different neural network structures specified in existing literature. Build on existing models and fine tune them to the present problem (Transfer learning)
4. Evaluate the trained network on a different dataset to understand how well it generalizes.

1.3 STRUCTURE OF THESIS

The project is viewed in three sequential stages:

- Understanding
  Chapter 2 provides an overview of the state of the art in the semantic segmentation of images, along with a short review of how satellites are used in remote sensing. Chapter 3 dives deeper into what deep learning is, and gives the reader the necessary background for formulating a computer vision problem using neural networks. Chapter 4 presents the data used in this report and also details how annotated maps for roads were obtained.
- Development and Approach
  Chapter 5 builds on the understanding from Chapter 3, and reasons the approach taken to build the neural network models in this work. It also describes a procedure for assessing the models developed. Chapter 6 explains the technologies used to implement the models, and goes over the framework. Chapter 7 closes this section by presenting the results, both qualitative and quantitative by using the evaluation metrics introduced in Chapter 5.
- Concluding Remarks
  Chapter 8 recommends possible extensions on the present work. Chapter 9 concludes the thesis, and reviews the objectives set out above in Section 1.2.

I

Understanding

CHAPTER 2

Overview

This chapter takes a top down approach in presenting the current problem. Understanding the data is crucial in deciding the machine learning techniques to be used. Hence, the first section summarizes how one obtains satellite imagery today, and its different use cases. This is followed by a brief introduction to deep learning, where the most promising results in computer vision are currently being seen. Finally, a refresher on semantic segmentation and state of the art efforts made in the segmentation of satellite imagery is presented. A more thorough investigation of the same can be found in [25, 28]. In the interest of brevity, an attempt has been made to not delve into specifics, except where unavoidable. In such cases, revisiting Section 2.3 after a reading of Chapter 3 is advised.

2.1 SATELLITES AND EARTH OBSERVATION

Satellites launched into space are mission specific, among which Earth Observation is one.[1] The first spacecraft to have taken pictures of the Earth was Explorer 6, launched in 1959, and the number of Earth Observation (EO), or remote sensing satellites has only increased since then.

Figure 2.1 shows the number of civilian satellites launched each year, until 2013, with more and more small satellites going into orbit in the years following.[2]

[1]Other satellite missions include communications, navigation, exploration etc.
[2]Currently, 1,419 operational satellites are estimated to be in orbit.Obtained from [43]

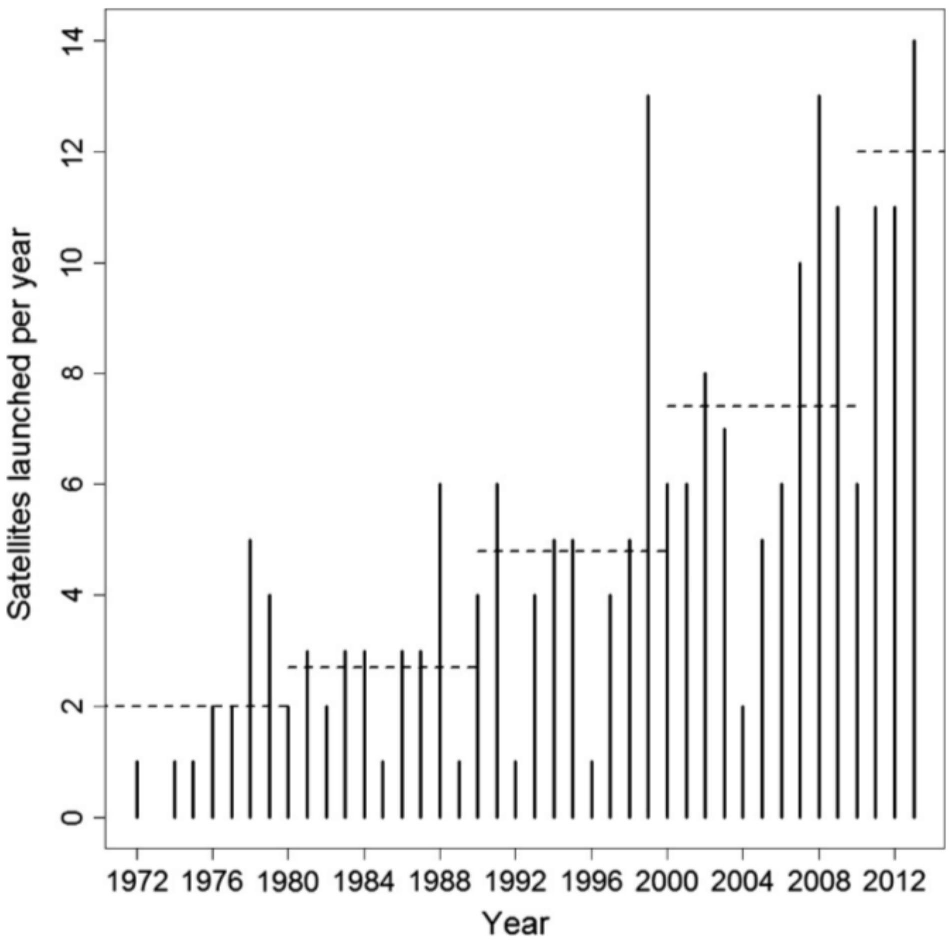


Figure 2.1 – Number of individual near-polar orbiting, land imaging civilian satellites launched per year. The horizontal dotted lines denote the average number launched per decade, which are 2, 2.7, 4.8, 7.4 and 12 respectively. Note that this graphic does not include private and commercial satellites launched. Adapted from [3].

Remote sensing satellites abound the Earth in Low Earth Orbit (LEO) and Medium Earth Orbit(MEO) [3]. EO satellites in most cases are in an application specific sun-synchronous orbit. A sun-synchronous orbit is one which ensures that the position of the sun with respect to the satellite and earth remains the same.

Chapter 1 introduced remote sensing as characterized by the distance between the sensor and its target object. The farther back one can go, the larger the coverage area that can be viewed, at the expense of spatial resolution. Hence, earth observation by remote sensing is primarily practiced using aircrafts, (high altitude) balloons or satellites alone or in different combinations, depending on the application. For imaging, a trade-off between spatial coverage area and spatial resolution can be reached by using both aerial and satellite imagery.

Earth Observation (EO) satellites have found applications in many fields, ranging from cartography, urban planning, disaster relief, real estate management to econometric/social trend analysis, military intelligence and climate studies. [18]

With the move towards automated drone delivery systems and autonomous vehicles, for instance, there is a greater demand for the use of satellite imagery that can be used as extraneous information for sensor fusion in the vehicles, to get clearer context of surroundings. Understanding of urban structures from images hence becomes important.

Sensors used in remote sensing are called so because they have the ability to *gauge (sense) interactions between earth surface materials and electromagnetic energy.* [4] These sensors are broadly categorized as active and passive sensors. Passive sensors use existing energy sources (commonly, the sun), while active sensors produce their own energy. [11]

Optical imaging from satellites/aircrafts is a form of passive remote sensing, where electromagnetic energy from the sun in the visible spectrum that is reflected off the earth is used to capture photographs. The visible spectrum here refers to electromagnetic radiation with a wavelength in the range of $380nm$ to $760nm$, sometimes also extending to include the near-infrared and ultraviolet regions.

[3]LEO ranges from 160 to $2000km$ altitude, while MEO ranges from $2000km$ to below the geostationary orbit.

[4]The term earth surface materials here is used loosely, since applications also see the sensing of artificial environments. The energy is also not limited to the electromagnetic spectrum. A thorough investigation can be found in [11].

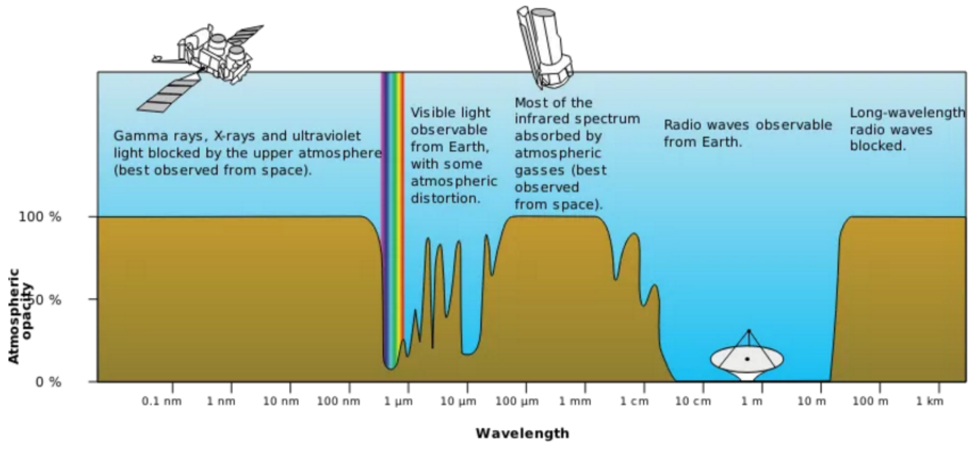


FIGURE 2.2 – A visualization of the different atmospheric electromagnetic windows. Adapted from [47].

Sensors are also capable of capturing other specific regions [5] in the electromagnetic spectrum. [48] Examples of prominent EO satellite programs include the LANDSAT program [22] from National Aeronautics and Space Administration (NASA), and the Indian Remote Sensing (IRS) [16] program, from the Indian Space Research Organization (ISRO).

Radar sensing is an example of active sensing, where the sensor includes a microwave emitter that emits radiation onto the target. The backscattered waves are then measured by the sensor to produce an image. The RADARSAT-2 satellite [32], launched by the Canadian Space Agency in 2007 is a prime example.

The reason for using the electromagnetic (EM) spectrum lies in the fact that each and every object *reflects, transmits and absorbs light differently, depending on its chemical composition.* [11] This property of an object is referred to as it's spectral signature, and is what makes remote sensing is possible. It is also important to take note of interference by the earth's atmosphere, which absorbs certain wavelengths in the EM spectrum. Hence, sensors are designed to measure specific ranges of wavelengths alone. These are termed atmospheric windows, as depicted in Figure 2.2.

The three channel (RGB) images used in this work are only a small subset of the imagery available from such satellites. Images obtained can also be hyperspectral or multispectral. Hyperspectral data contains a large number

[5] Also known as spectral bands, these include: microwaves(Radar), Infra-Red (IR), Near and Mid-IR, Visible light, Ultra-Violet. These regions span wavelengths $0.1cm - 0.4\mu m$.

of very narrow EM bands of $10 - 20nm$. [47] NASA's Hyperion[6] imaging spectrometer is one such example, producing $30m$ resolution images in 220 spectral bands. Multispectral imagery is similar, but contains fewer and wider bands, as obtained from the Landsat-8 sensors.

The quality of information in an image provided by EO satellites is characterized by its resolution. These are defined as the spatial resolution, i.e., the visible details in pixel space, spectral resolution, i.e., depending on the width of EM bands available in the image, and temporal resolution, which depends on the revisit time period of the particular satellite.[47]

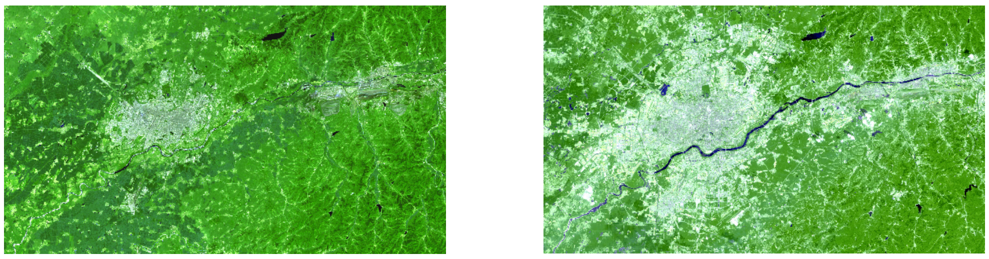


Figure 2.3 – Landsat imagery, depicting urban expansion in Shenyang, China, spanning 30 years from 1984 to 2014. Images reproduced at a lower resolution. Accessed at [22].

Most private imaging corporations provide very high resolution imagery, at less than $0.5m/pixel$, made possible by state of the art technologies. Digital Globe's WorldView-3, a commercial EO satellite launched in 2014 currently provides a very-high resolution (VHR) of $0.31m$.[7]

2.2 DEEP LEARNING

Neural networks, despite having been around for decades, have garnered much attention only in the last few years in the computer vision and machine learning communities. While the topic is covered in detail in Chapter 3, a brief introduction is provided here to make Section 2.3 readable.

One definition for an artificial neural network was provided first in [7], where the author stated that *"a neural network is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."*

[6] `https://eo1.usgs.gov/sensors/hyperion`
[7] `https://www.digitalglobe.com/resources/satellite-information`

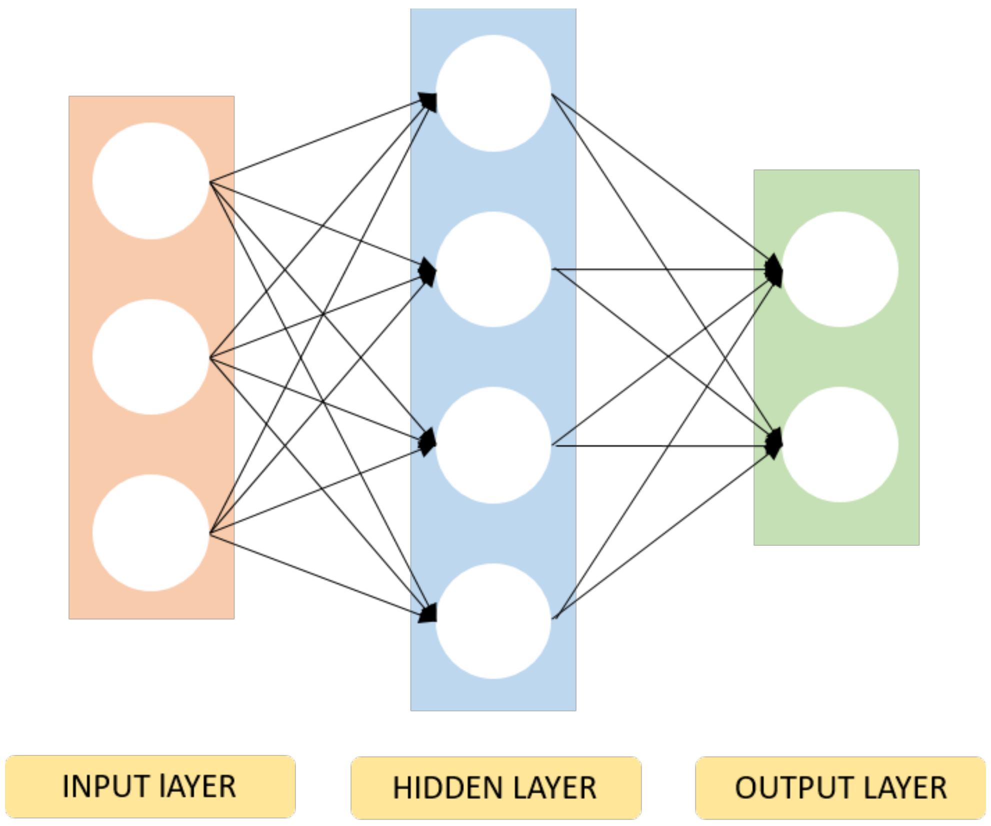


FIGURE 2.4 – Visualizing a 3 layer neural network architecture. Reproduced from [19].

In a very rough manner, neural network algorithms can be thought of to be modeled on the structure of neurons present in the brain. These networks are generally visualized as layers of *neurons* stacked on top of one another. Each layer consists of a number of units (or neurons), followed by an activation function.[8] A very basic 3-layered neural network is shown in Figure 2.4.

Neural networks can be thought of as classifiers that extract hierarchical features from raw data, (which in our case, is pixel values) and learn models for various vision related tasks, such as object recognition and semantic segmentation, among others.

The parameters of the model, i.e., intermediate neurons in the hidden layer in Figure 2.4 for instance, are trained and learned (i.e., updated) via classical optimization methods. The user defines a cost or loss function to

[8]These are modeled after the electrical stimulations at synapses present in a brain, where dendrites convey information from one neuron to another.

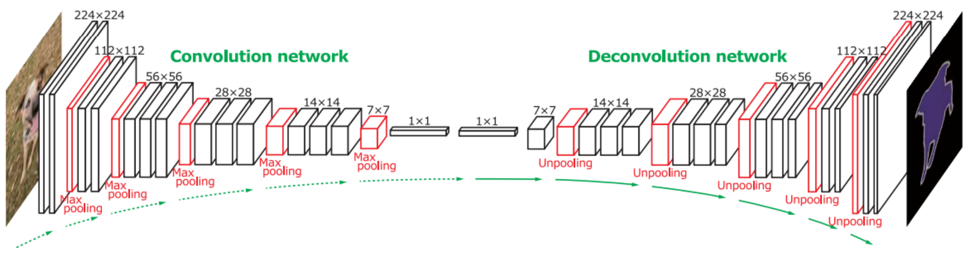


FIGURE 2.5 – Architecture of a convolutional neural network. Depicts the deconvolution network proposed in [30]. One can see the gradual progress of (de)convolutional layers and (un)pooling functions across the network. Adapted from [30].

be optimized for. This cost function encodes the probability of the neural network output being as close to the desired output (ground truth) as possible. The parameters of the neural network are then updated accordingly, to minimize/maximize the cost function. This process of updating is done via gradient methods, by "propagating" the error (mismatch between desired output and neural network output) back through the network units to the input. This algorithm is formally known as the backpropagation algorithm.

The layers in a neural network are of different types: convolutions, which consist of filters, pooling layers which introduce a translational invariance to the network, and more. These are dealt with in more detail in Chapter 3. Figure 2.5 shows a visualization of one such convolutional neural network.

2.3 SEMANTIC SEGMENTATION

The interpretation of visual information has been approached in several ways over the years, but the underlying process remains the same: examining images for the purpose of identifying objects and judging their significance.[9] The problem of learning from visual information is generally classified into image classification [37], object localization and detection [15], semantic segmentation and instance segmentation [44], among others.[19]

Semantic segmentation for images then, can be defined as the process of grouping parts of images so that each pixel in a group corresponds to the object class of the group as a whole. In the present work, the object classes correspond to roads and the background. In a multi-class setting, the classes can be further grouped into buildings, meadows, parking lots etc. The remainder of this section details recent advances in dealing with the semantic segmentation problem, along with literature on the same applied to satellite/aerial imagery.

State-of-the-art networks (or models) in this space are obtained by evaluating performance on large scale benchmark datasets, such as the Microsoft Common Objects in Context(MSCOCO) [23], ImageNet[34] and PASCAL VOC [12]. MSCOCO is an image recognition, segmentation and captioning dataset with more than $300,000$ images and 80 object categories, while ImageNet [9] is a large scale dataset with more than $14,000,000$ images and more than 1000 categories, with different subsets used for each benchmark task.

With deep neural networks currently enjoying a wave of success years on image recognition tasks, the rest of this work approaches the problem of segmenting satellite imagery with deep nets. Transfer learning, a technique wherein knowledge learned by a deep network in one context is used to improve its performance in a related but different context will be explored. [49]

Image interpretation includes as a subset the process of image examination with a specific purpose of identifying discriminative characteristics of objects of interest. In order to obtain total scene understanding from an aerial image several steps are needed. Given an image, a segmentation step is applied in order to divide the scene into regions of specific categories (such as residential areas, farmland, forest, roads etc.), i.e., to see the entire visual environment as an interconnected image of all categories.

Most state-of-the-art results were designed for the image classification task, wherein an object class is assigned to an image as a whole. [25] presented a modified version of VGGNet [37] - which originally took in images of a constant shape to classify them amongst a large group of classes. The VGGNet model itself improved upon previous image classification networks, notably AlexNet [21], by getting rid of local response normalization layers. The power of VGGNet also lay in its simplicity, with a very homogeneous structure as compared to previous models. The models introduced in [25], referred to as Fully Convolutional Networks (FCN) modified the original VGGNet designed for an image classification task, for the image segmentation task. Noting that spatial information in an image is paramount in semantic segmentation, the fully connected layers generally found in the final layers of a neural network were discarded in [25] and replaced by their equivalent convolutional layers.

Another recent semantic segmentation algorithm was introduced in [30], wherein a deconvolution network was stacked on top of the usual convolutional network. By deconvolution, we refer to taking the convolutional trans-

[9]ImageNet is the dataset behind the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

pose of the input. The models developed thus improved upon the FCNs in [25] by integrating the deconvolution network. A network ensemble approach was also proposed, combining the knowledge learned from the deconvolution network and the FCN methods in [25], to further improve performance. The models in [30] proved to be state-of-the-art when evaluated on the PASCAL VOC 2012 [10] segmentation benchmark dataset.

State-of-the-art results in aerial image labeling were given by [29] in 2013, along with benchmark datasets for the evaluation of deep networks dealing with satellite imagery. This followed a detailed study of previous works, ranging between both neural network (NN) methods and non-NN methods. One of the datasets released along with [29], the Massachussetts Roads dataset is used in the present work, and described in detail in Chapter 4. [28] detailed a study on the effect of tuning different parameters in a NN model, and thus provides a strong foundation for the current work. [28] used a patch-based labeling framework, and presented an end-to-end framework for learning to label aerial imagery, by addressing three key issues: learning from noisy data, learning discriminative features, and performing structured predictions to improve the quality of predictions.

In [35], multiple objects were extracted from aerial imagery by building upon the work of [28], and proposed additional models that performed better. While [28] dealt with binary classification (roads vs. background, or buildings vs. background), [35] combined the benchmark roads and buildings data sets to predict three labels on each image (road, building, background). In particular, they proposed a new function, the channel-wise inhibited softmax (CIS) to effectively train a neural net. State-of-the-art results were improved upon in [35] by introducing a model averaging technique, a type of ensemble learning [11] method.

Another promising work in this direction includes [17], which combined very-high-resolution (VHR) images with publicly accessible geocodes of specific locations to generate new ground truth labels for effective neural network training. Similar to [35] in that the author dealt with 3 class labeling (building, road, background), this work built on the FCN architecture introduced in [25] by modifying the base architectures described there, and introduced additional models with marginal performance improvement. [17] proved the notion that publicly available perspectival imagery can be used to train neural networks to produce semantic maps.

[10] `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/`

[11] Ensemble learning is the method of strategically using multiple models to improve the performance of a model (a specific metric)

CHAPTER 3

Convolutional Neural Networks

This chapter provides a short conceptual introduction to neural networks. It presents an overview of the different layers of a Convolutional Neural Network (CNN), and specifies its relationship to the Fully Convolutional Network (FCN). Following this, the backpropagation training algorithm is reviewed.

3.1 Architecture

A neural network fundamentally consists of learnable parameters like in a single linear classifier - termed weights and biases.

Consider a simple linear function of the form shown in Equation 3.1. An input tensor (the data input, a flattened image vector [1] when dealing with images) is multiplied with an appropriately sized variable weight matrix, and added with a bias vector. If x_i denotes the input vector and W_i denotes the corresponding weight matrix, b_i its bias vector, the linear function returns a score, y_i as:

$$y_i = W_i x_i + b_i \tag{3.1}$$

This function essentially mapped the data x_i from image space into a score, y_i for that particular image. In an image classification task, y_i can be understood to encode the classifier's confidence that the image x_i belongs to a particular label. For image segmentation, on the other hand, the problem is formulated

[1] Images can be represented as a matrix of numbers, specifying real values across different channels/bands. For an RGB image of size $a \times b$, this would mean a matrix of size $a \times b \times c$.

as a per-pixel classification. Hence, y_i would represent the score given to a particular pixel in the image, x_i as belonging to a particular class. [2]

This linear mapping can sometimes be followed by a non-linear *activation* function as in Figure 3.2, inserted to activate only for inputs of a specific range of input values. This is what is contained in a single layer of a neural network.

The same process is repeated multiple times, for each layer of a network, to obtain the final class scores. Scores thus obtained are fed into a final classifier layer, which converts them to class probabilities, i.e., a probability distribution for each input as belonging to different classes. The sum of all probability values across classes for a single input is always equal to 1. The entire network can thus be thought of as one differentiable function, mapping the raw image from the input to predicted class scores at the output.[3]

Convolutional neural networks take advantage of the underlying structure in images. Topological information, i.e., spatial information about the structure in an image, such as adjacency and rotations are also taken into account. We shall now look into the details of how the different layers of a neural network interact with each other.

A neural network consists of several layers defining different operations each of which are explained in the following subsections: convolutional layers in Section 3.1.1, pooling layers in Section 3.1.3, activation layers in Section 3.1.2, regularization layers in Section 3.1.6, fully connected layers in Section 3.1.4, and the final classification layer in Section 3.1.5.

A classifier is always the final layer, with the purpose of producing class probabilities as an output. A predefined cost function, defined in Section 3.2 is calculated out of the classifier outputs. Optimization techniques, most commonly, stochastic gradient descent, are then applied on the cost function to compute gradients of the constituent variables backwards in the network, and these parameters are accordingly updated.

3.1.1 *Convolutions*

Applied to RGB images (which is what we deal with in this work), convolutional networks take note of the fact that an image is a 3-dimensional matrix [4], and each of the layers are arranged similarly. This is depicted in Figure 3.1. Each such layer of a CNN consists of kernels (or filters) of a certain volume,

[2] A detailed visualization can be found at `http://cs231n.github.io/linear-classify/`
[3] http://cs231n.github.io/
[4] In the case of hyperspectral/multispectral images, this can be generalized to being an n-dimensional matrix.

viewed as a volume of units (also called neurons), sized $h \times w \times d$, with h and w being its spatial dimensions, and d the number of feature channels of the kernel. Every one of these filter is *convolved* with a corresponding volume of the input image, and slid through the entire image (of size $H_i \times W_i \times D_i$ with $H_, W_i$ being its spatial dimensions and D_i being the number of channels) across its spatial dimensions H_i, W_i.

Convolution here refers to a summation of the element-wise dot product of the neurons in each filter with the corresponding values in the input. Thus, the input image can be considered to be the first layer in a CNN. Based on this notion, a convolution with a single filter at each layer results in a 2-dimensional output, of a certain spatial size (decided by parameters such as stride and padding, defined below, used in the convolution step). This is the activation map for one filter on the input. At each layer of a CNN, N such filters are used, each one resulting in an activation map. These are stacked together across the 3^{rd} dimension to obtain the output of a single convolutional layer that consists of N filters. Figure 3.1 describes this procedure visually, showcasing a 2×2 filter applied on the input volume.

A single neuron in one filter of a certain layer can be mapped to its connected neurons in all preceding layers by following such convolutions. This is termed as the effective receptive field of that neuron. It is easy to see that convolutions result in very local connections, with the neurons in lower layers (closer to the input) having a smaller receptive field than those in higher layers. Lower layers learn to represent small areas of the input, while higher layers learn more specific semantics, since they respond to a larger subregion of the input image.In this way, a feature hierarchy is built from the local to the global.

The stride s of a filter is defined as the intervals the filter moves in each spatial dimension, and padding p_h, p_w refers to the number of pixels added at the outer edges of the input. The stride can hence be considered a means of subsampling [45] the input. These hyperparameters, along with N (the number of filters) and h, w (the spatial extent of the filter) help define the size of the output volume at each layer. Generally, square filters are used, i.e., $h = w = f$. The output volume of such a layer is given by

$$H_o = \frac{H_i - f + 2p}{s + 1}$$
$$W_o = \frac{W_i - f + 2p}{s + 1}$$
$$D_o = N$$

It is important to note that filters need not always be of the same size at different layers, nor be homogeneous. In [41], we learn that the Inception

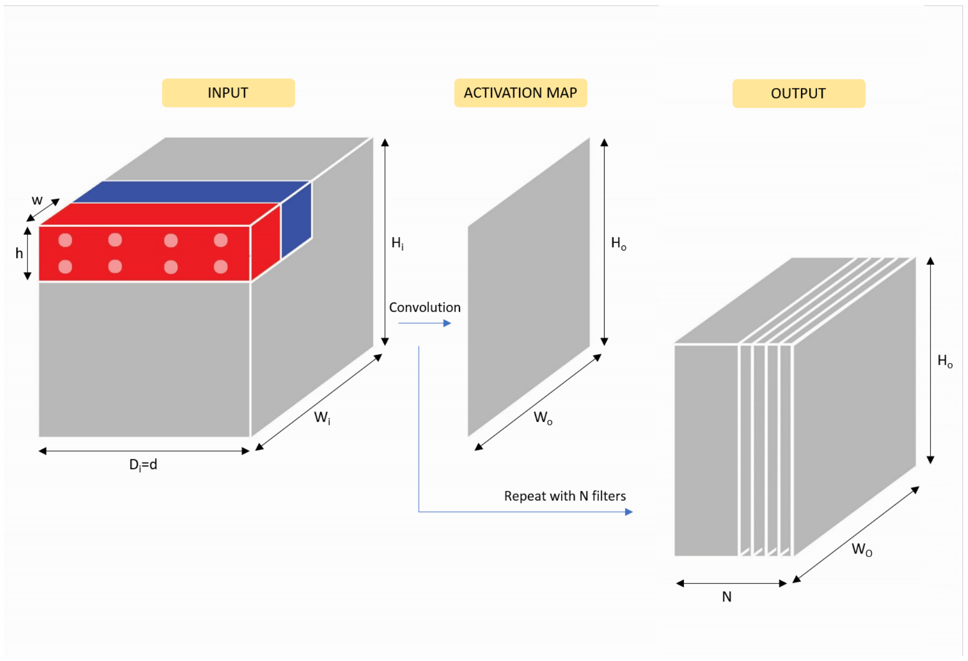


FIGURE 3.1 – An illustration of a single convolutional layer. The red and blue areas signify two positions of the same filter of size $h \times w \times d$ which is *convolved* across the input volume by sliding it across. Output volume is obtained by using N such filters. Considering the filter to be a 2×2 filter, we see that the stride parameter here, $s = 2$. For an RGB image input, $D_i = d = 3$.

architecture, designed for computational efficiency, used filters of varying sizes at different layers. 1×1 convolutions were used as dimension reduction modules to get rid of computational bottlenecks.

3.1.2 *Non-Linearity Functions*

Neural networks initially stemmed from biological theory on how neurons in a brain are connected, and allow for the processing of information. Non-linearity functions are used to model the firing (or activations) of specific neurons in a network layer, and are hence also referred to as activation functions.

In general, the outputs of a convolution step are fed into activation functions at each layer. A variety of such proposed non-linearity layers exist, notably the Sigmoid function, Tanh, Rectified Linear Unit (ReLU)[nair and hinton] or Maxout, among others. For most purposes, ReLU units have been found to be effective and are the preferred choice.[21] A ReLU unit activates by thresholding the negative inputs at zero, and passing the positive inputs unchanged, as in:

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

where z is the input to the ReLU unit.

Sigmoids, for instance, tend to saturate when initialized weights are too large. On the other hand, if the gradient is negligibly small, it might as well not exist, thereby being *killed*. This is the vanishing gradient problem. [19]. Another issue with the sigmoid is that outputs of the sigmoid function are not zero-centered. As seen in Figure 3.2, the *tanh* function is quite similar to the sigmoid, except in that it is zero-centered. Finally, ReLU units, proven to be computationally efficient [21], are also not as hindered as tanh and sigmoid functions by vanishing or exploding gradients.For these reasons, ReLU is the recommended activation function [19].

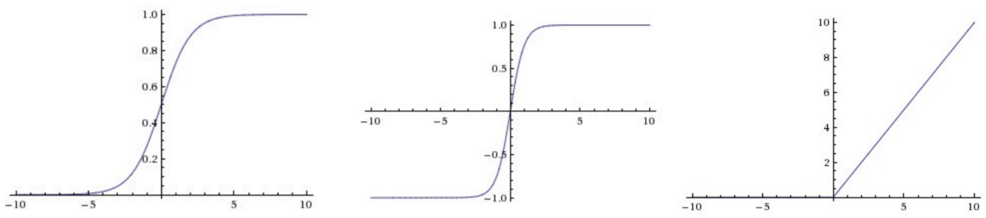


FIGURE 3.2 – Sample activation functions. On the left is a sigmoid function, that squeezes real numbers into the range [0,1]. In the middle, the *tanh* non-linearity and on the right, a ReLU function. Adapted from [19].

3.1.3 *Pooling layers*

Convolutional layers are commonly interspersed with pooling layers, which aid in down-sampling the input features spatially. The input information is aggregated by sliding a window across it, and feeding the output to a (non-linear) pooling function, so as to reduce its spatial resolution.

In a pooling operation, the input image is partitioned into (usually non-overlapping) sub-areas, and a single value from each sub-area is returned for each activation map in the depth-dimension. In the case of max-pooling, which is used throughout this work, the maximum value from each sub-area is returned. Pooling layers also have a stride specification that allows for control over the output dimensions. An alternative is the average pooling function, where the average value of the sub-area is returned instead.

Pooling provides a form of robustness to the network, by reducing the quantity of translational variance in the image.[4] Additionally, it also decreases the computational cost of the network by discarding redundant (or

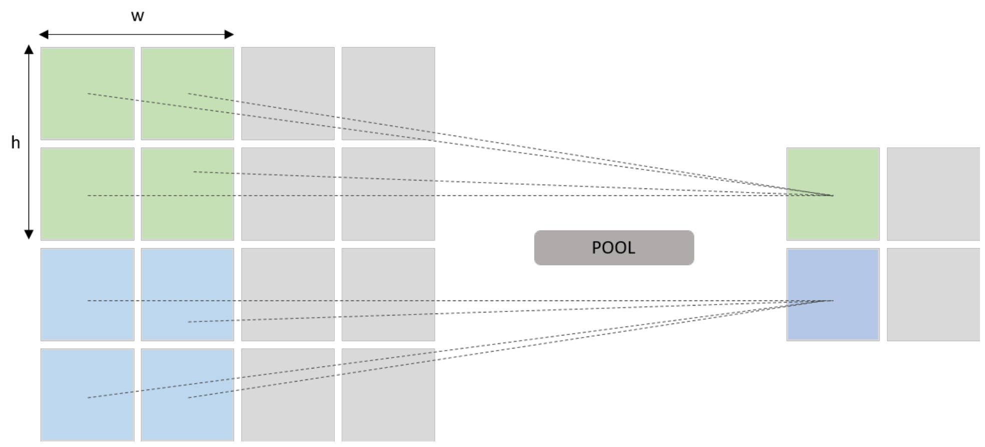


Figure 3.3 – Illustration of a 2×2 pooling layer. Note the reduction in spatial resolution in the output layer, thereby making it invariant to local transformations in the input. Image reproduced from [17].

unnecessary) information, decided by the particular use case, thereby making the network more efficient. Other forms of the pooling layer include the average pooling function, the L^2 norm of a rectangular neighborhood, or a weighted average based on distance to the central pixel. [4]

3.1.4 *Fully Connected Layers*

Once higher level features are detected from the preceding convolution and pooling layers, a fully connected layer is usually attached at the end of the network. Each neuron in this layer is connected to the entire input volume (from the preceding convolution, activation or pooling layer) that it receives. The intuition here is that by taking into account all the activations received, the neurons in this layer can determine which features correspond with which class the most. The activations of these neurons are computed via Equation 3.1.

Since a neuron in a fully connected layer receives activations from all input neurons, spatial information is lost. This is undesirable in a semantic segmentation problem, where spatial context is key in effective learning.

One way to overcome this is by looking at the fully connected layer as its equivalent convolutional layer representation. They can be viewed as 1×1 convolutions applied over the entire input (either in image space, or feature space) - with a full-connection mapping. The filters here can also be viewed as having spatial extent equal to that of the input layer. Thus, one can proceed

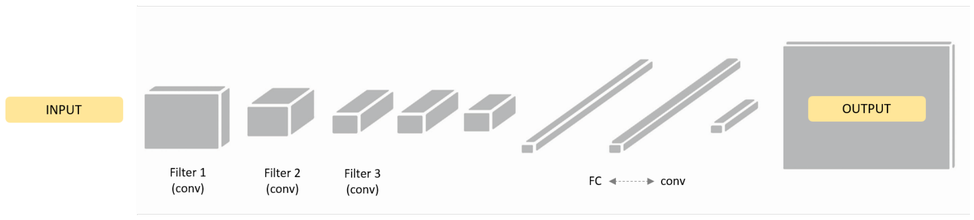


FIGURE 3.4 – Illustration of fully connected layers converted into 1×1 convolutions, depicted as the long, narrow convolution filters just before the output layer.

as in the convolutional layers 3.1.1 This is the basic intuition behind Fully Convolutional Networks, introduced in [25].

3.1.5 *Classifier*

A classifier is chosen by taking into account the problem at hand and the data being used. The softmax function is used in this work, since it allows for the prediction of one class out of mutually exclusive classes. For the binary class problem, this reduces to being a logistic regression. Here, the scores from the network are interpreted as unnormalized log probabilities [19], and the loss metric is defined as the negative log-likelihood of the softmax function, and is a cross-entropy loss. Here, the softmax function gives a probability value for a certain input, x_i belonging to a certain class, k as:

$$p(y = k|\bar{x} = x_i) = \frac{e^{(s_k)}}{\sum_j e^{(s_j)}} \tag{3.2}$$

where s is the score obtained for the particular class from the previous layers of the CNN.

Apart from the softmax function, it is (albeit less) common to see the SVM classifier, where the loss is defined as a hinge loss. The SVM classifier computes uncalibrated scores for each class, as opposed to the softmax above which returns interpretable results akin to probabilities. It is generally seen that SVM and Softmax return comparable results [19].

3.1.6 *Regularization*

Overfitting on training data is a major problem, especially when dealing with deep neural networks where the network is powerful enough to fit itself extremely well on the training set alone, at the cost of generalization ability. Overfitting is best avoided. Techniques developed to do just this are termed regularization techniques.

Dropout is a simple and effective regularization strategy that is included in the training phase. First introduced in [40], it is implemented as dropout layers, characterized by a probability value. Each neuron in a training step is kept active with a specified probability, p. Thus, the neural network is sampled iteratively by dropping out certain neurons or not. All edges connected to the dropped out neuron are removed at each training iteration, and restored before the next. Figure 3.5 showcases this step for a 3-layer neural network.

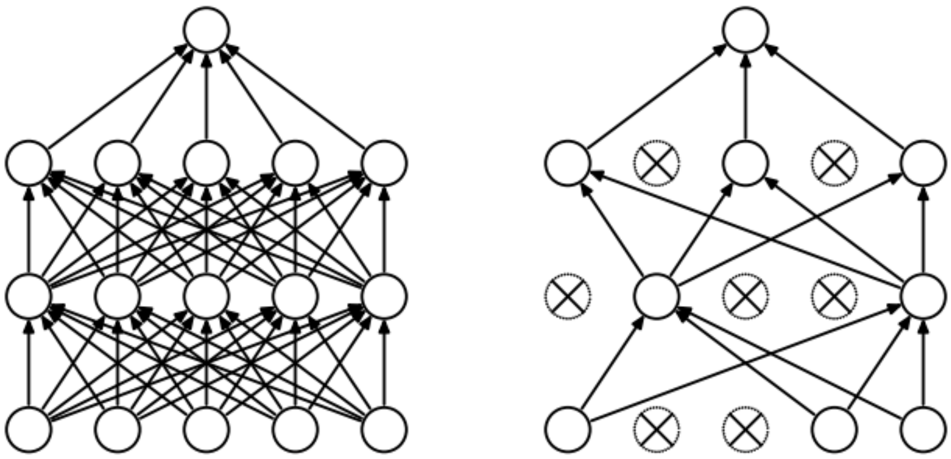


FIGURE 3.5 – An example of dropout in a network. On the left is a standard neural net, and on the right, the network after applying dropout. Adapted from [40].

In the prediction phase, all neurons are kept active. To account for the subsampled dropout networks during training, an approximation is done via scaling each neuron activation by p on the full network.

Another commonly seen regularization method is L2 regularization. The squared magnitude of all parameters) are added directly into the loss function defined in Section 3.2.2, and this "total loss" function is minimized as in the usual case. The intuition here is that this results in a *preference for certain weights over others*[19]. Termed the regularization penalty ($R(W)$), this L2 norm is calculated as:

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

where i, j span the size of the weight matrix W whose elements are $w_{i,j}$. This term is scaled by λ, the regularization strength, and added into the loss function, defined in 3.2.2.

3.2 TRAINING

The learning process in a neural network can be broken down into four foundational steps:

1. Forward computation
2. Error/loss optimization
3. Backpropagation and parameter updates

3.2.1 *Forward Computation*

The input image is first fed through a pre-processing pipeline, which generally includes a mean subtraction and normalization step.

$$x_i = x_i - x_{mean} \tag{3.3}$$

$$x_i = x_i / \sigma \tag{3.4}$$

where the input is x_i, with $i = 1, 2 ... N$ and σ is the standard deviation of the input vector.

This is then fed through the neural network architecture, which generally consists of the layers described in 3.1 in different combinations. It is usually the case that lower layers consist of alternative convolution and pooling layers, followed by fully connected layers higher up. The network returns the class score for the input, encoding its probability of belonging to a certain class. To note here is the fact that the scores returned from the network can be unscaled, as in the case of SVM classifier, or negative log likelihood confidences, as defined by the softmax classifier. As discussed previously, the former is less interpretable, while also being dependent on the margin. For semantic segmentation, a class score is provided for each pixel in the image. The kernels learned across the different convolution layers can be visualized to understand what a network might be learning to recognize, or segment. For this work, these are presented in Chapter 6.

3.2.2 *Loss Optimization*

The set of scores provided by the network need to be optimized by adjusting the values of the parameters being learned in the network, i.e., weight filters and biases. Such an uncertainty in determining which set of parameters are ideal is quantified by the loss function, which can be formulated as an optimization problem. For the softmax classifier, this turns out to be the cross-entropy loss for each vector of class scores s:

$$L_i = -log\left(\frac{e^{(s_k)}}{\sum_j e^{(s_j)}}\right) \tag{3.5}$$

with cross entropy:

$$H(p,q) = -\sum_{i=1}^{n} p(x) \log q(x)$$

where q represents the softmax function defined above. The final loss is hence defined to be:

$$L = \sum_{n}^{N} L_i + \lambda R(W) \tag{3.6}$$

where λ is the regularization strength, and L, the total loss. The loss optimization step is then defined as a minimization of L. Note that in the case of an SVM classifier, a hinge loss function is used instead, defined as:

$$L = \frac{1}{N} \sum_{i}^{N} \sum_{j \neq y_i} max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + 1) + \lambda R(W) \tag{3.7}$$

where x_i is the input, j is the correctly predicted label, and y_i, the incorrect labels. In the ideal scenario, the predicted label from the network is the same as the training label for each pixel, i.e., 0 loss is computed. For minimizing the calculated loss thus, the problem is formulated as an optimization step, and the loss function is minimized.

3.2.3 *Backpropagation*

Backpropagation is a fundamental concept in learning with neural networks. The objective here is to periodically update the initialized weight parameters. It is observed that the problem backpropagation helps solve is that of optimizing a cost function. This is exactly what is done in optimal control theory, where the problems are variational with constraints where a function that optimizes a loss function is sought out.

In the machine learning paradigm, this approach towards optimization is different primarily in that the functions are imagined to be a graph of interconnected units of computation [33]. A function is thought of as a dynamical object being evaluated by a graph of discrete elements. This interpretation lets us see that the backpropagation algorithm is akin to the chain rule in differentiation. Figure 3.6 provides a simple example of the backpropagation algorithm in action.

In the forward pass, the usual values at each node are calculated to be 3 and -12 respectively. During the backward pass, gradients are calculated as specified by the chain rule of differentiation:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \tag{3.8}$$

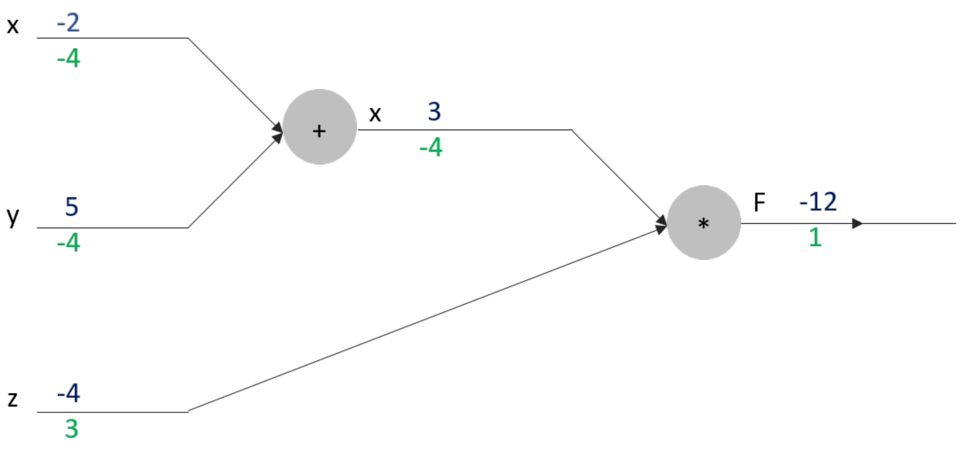


FIGURE 3.6 – Illustration of a simple computational graph for the equation $(x+y)\times z$ where $x=-2$, $y=5$, $z=4$ depicting backpropagation. Forward pass,values in blue, computes from left to right, while backward pass, values in green, iteratively applies chain rule to calculate gradients first for the output, and *flows* back to the input. This example has been completely reproduced from [19].

This backpropagation algorithm can be extended by formulating an update rule for a single neuron, as in Equation 3.9, where w_{ij}, the weight value between two neurons i, j in two proximal layers, ϵ is the learning rate, and L, the loss function.

$$w_{ij} = w_{ij} - \epsilon . \frac{\partial L}{\partial w_{ij}} \tag{3.9}$$

The optimization algorithm is generally understood to be gradient descent and its variants. In the present work, we use the ADAM optimization algorithm. A simple implementation of gradient descent might not work very well in a deep network, since it faces issues with navigating around local optima. This is rectified by introducing another parameter, the momentum, which helps to aid the gradient descent (GD) update process as necessary to reach an optimal point. Adam optimizer is one such implementation, short for Adaptive Moment Estimation,[20] where adaptive learning rates are calculated for each parameter. In the case of a noisy gradient parameter as occurs near a local optima, the x value is updated using an estimation of the first and second moments of the gradients.[2] These values correspond to the mean and variance of the gradients respectively. These are shown below as:

$$m = \beta_1 m + (1 - \beta_1) d_x \tag{3.10}$$

$$v = \beta_2 v + (1 - \beta_2) d_x^2 \tag{3.11}$$

$$x = x - \epsilon \frac{m}{\sqrt{v}} \tag{3.12}$$

where β_1 and β_2 are constants used to specify the decay rate, m is the mean, and v, the variance of the gradients. ϵ is suggested to be set at 10^{-8} [20]

3.3 HYPERPARAMETERS

An important part of developing a NN architecture is the selection of hyperparameters. Hyperparameters are variables set to specific values before the actual training (optimization) process. Different methods exist in choosing these values:

1. Manual: Hyperparameters are set by hand, usually by leveraging existing knowledge about the problem and guessing parameter values. Parameters are then modified as necessary, until a usable set of parameters are found.
2. Search algorithms: A grid search, or random search algorithm can be deployed to identify feasible ranges for the hyperparameters. The network is then trained on multiple models by using all combinations of parameters made available in these ranges. A random search algorithm is recommended here, as it has been shown to generally work better than other methods. [6].
3. "Hyper" Optimization: The idea here is to create an automated approach which can optimize the performance of the model according to the task. The generalization performance of a network is modelled in such a way that the choice of parameters chosen by the search algorithm following an experiment is optimized. [38]

One of three methods is usually followed to feed the neural network with data in the training phase:

1. Batch Gradient Descent : The cost function gradient is calculated over the entire dataset.
2. Mini-Batch Gradient Descent : A subset of the training dataset (called a mini-batch) is fed into the network and updates are made for each such mini-batch.
3. Stochastic Gradient Descent : Parameter updates are performed for each training example.

A list of best practices that is followed across the community was proposed in [5], and the same is listed below for completeness.

3.3.1 *Learning Rate*

The learning rate can be understood as the rate at which the gradient updates to the parameters occur in the gradient direction. When this rate, ϵ is too small, model convergence takes a long time. On the other hand, if ϵ is too large, the model diverges, and the loss might fluctuate indefinitely. To ensure optimal learning, an initial learning rate ϵ_0 is first defined (0.01 is a generally accepted standard here), following which the rate is updated by scaling with a decay factor periodically, depending on the mini-batch size and number of iterations. This updation of the learning rate can be formulated as:

$$\epsilon_t = \epsilon_0 \forall t < \tau \tag{3.13}$$

$$\epsilon_t = \epsilon_0 t^{\alpha} \tag{3.14}$$

where τ and α are ideally set to adapt depending on preset thresholds based on the loss function.

3.3.2 *Mini-batch size*

Mini-batch is chosen over batch or stochastic gradient descent updation rules, because it offers the advantages of both other options, while minimizing the disadvantages. (not as noisy as stochastic gradient descent, and not as inefficient as batch gradient descent). Following this, the size of mini-batch to be used is set based on the computational power available.

3.3.3 *Weight Initialization*

The local minimum reached by the training algorithm is highly dependent on the initialization scheme used for the weight matrices. [27]. While biases can be set to 0, oftentimes, weights are initialized to vary in a random 0 mean distribution, by taking into account the fan-in of a particular neuron.

3.3.4 *Regularization*

A validation set can come in handy in setting regularization strength λ and p for the dropout probability. The regularization strength is set by evaluating the model on the validation set during training. λ is usually dependent on the loss function, and ranges anywhere from 10^{-3} to 10^{4} [5]. Dropout is kept to a sensible default of 0.5, which has proven to be sufficiently effective [40].

CHAPTER 4

Data Review

This chapter presents the datasets used in this work, and briefly describes possible techniques of obtaining annotated ground truth data from publicly available sources.

4.1 DESCRIPTION

The advent of open sourced collaborative projects such as OpenStreetMap has made it possible for the computer vision and machine learning research community to get access to high quality ground truth data for training on satellite imagery. In [29], the authors released high quality publicly available datasets for this purpose. The current work makes use of one of these datasets - the Massachussetts roads dataset[1].

4.1.1 *Massachussetts Roads Data*

This dataset included 1500x1500 pixel images of the city of Massachusetts released by the state, at a resolution of 1 meter per pixel. Target maps used on these images were also readily available, in rasterized format. A description of how these were prepared is given in Section 4.2.

This work makes use of the Roads Dataset, from Chapter 6.2 of [28]. Each image in the original dataset comes at a 1500x1500 resolution, and split randomly into training, validation and test datasets as in table below.

[1] http://www.mass.gov/anf/research-and-tech/it-serv-and-support/application-serv/office-of-geographic-information-massgis/

Training	Testing	Validation
1108	49	14

Table 4.1 – Table presenting randomly split sets of the Massachusetts Roads dataset in [28].

Training	Testing	Validation
4440	294	84

Table 4.2 – Table presenting randomly split sets of the Massachusetts Roads dataset prepared from images in Table 4.1.

Hard binary labels were used in the original generation of this data. One point of note here is that the original labels downloaded for Massachusetts were three channeled RGB images with only two classes throughout. This was hence first converted into a sparse label representation and tested. Marked improvement in performance was noticed when a dense matrix of one-hot encoded vectors were used for the labels instead.

Due to computational constraints, the 1500×1500 images from Table 4.1 were cropped into non-overlapping segments of size 500×500, across the train, test and validation splits, and a random subset of 4440 images thus generated were chosen from the training set, while the number of examples in the test and validation sets were left untouched. This training included a wide range of urban, suburban and coastal regions. During training, the images were fed into the models in a minibatch of size 3. The training set size, is unfortunately not large enough to guarantee good production level performance. This was set with the constraint of limited computational budget available during the initial phases of the project. Representative samples from the Massachusetts Roads dataset can be seen in Figure 4.1. Thereby, the Massachussetts roads dataset, modified as above, is presented in 4.2.

No further randomized scheme was used, apart from the selection of the training set of 4440 images for the training set.

In the Massachusetts dataset, a wide range of urban areas were seen to include parking lots, usually at big box stores, thereby resulting in comparatively large gray areas, of the same RGB values as for roads. Additionally, a quick qualitative overview over randomized subsets of the 4400 images showed that tree cover in many areas resulted in blocking out of roads from the vantage point of the viewer. This was also observed in the final results, where predicted roads in suburban regions with a dense population of trees

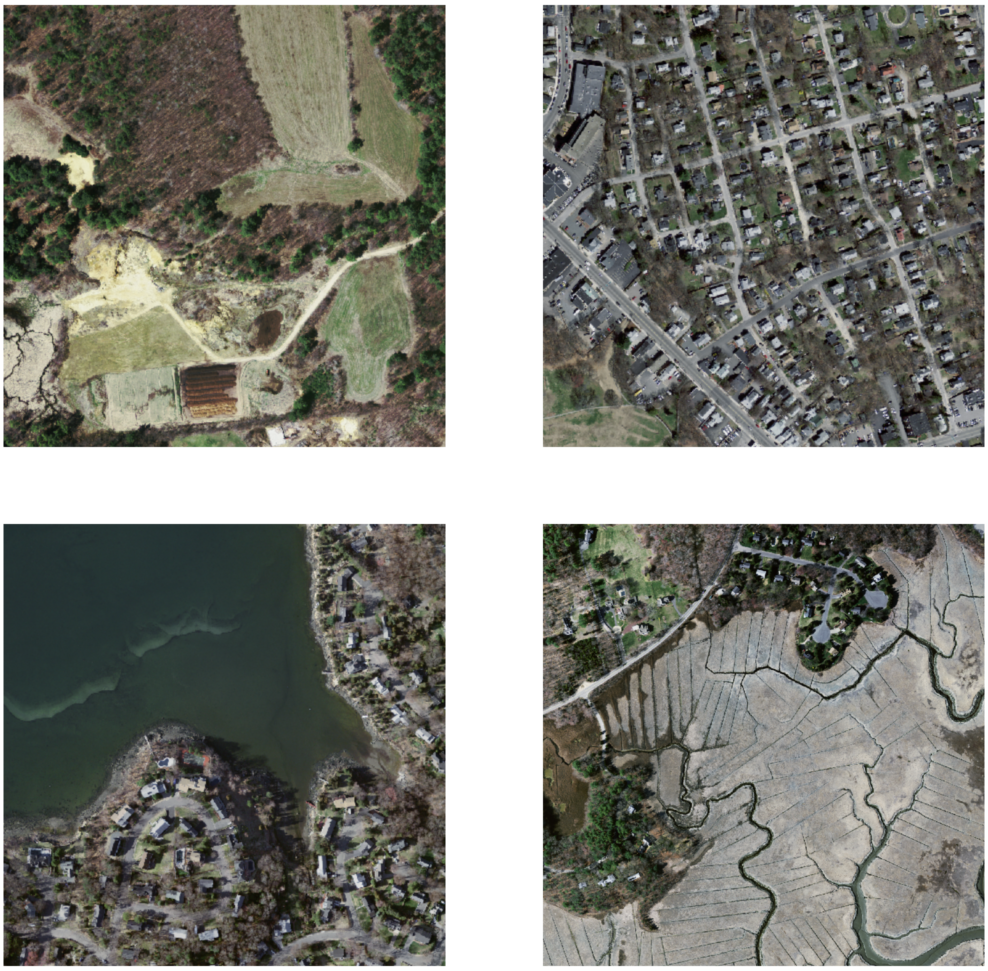


FIGURE 4.1 – Images from the Massachusetts Roads dataset, showing an urban, suburban and coastal region. Bottom right shows a region with waterways that could, in theory, be mistaken for roads in the case of simpler models.

near roads were broken lines, with many of them being discontinuous. This is discussed in detail in Chapter 7.

The testing set and validation set are distinguished by the purpose for which they are used. In a normal machine learning task, a model is trained on a training set, and evaluated by testing it on the test set. A validation set is used to ensure that the best possible model is obtained. This is done by using the validation set to tune the parameters of the model during training, by periodically evaluating on the validation set. For instance, observing the trends in training set accuracy and validation set accuracy together can tell us if the model is overfitting onto the training set, or underfitting. This can then lead to an informed tuning of the regularization strength and other hyperparameters presented in Section 3.3.

4.2 DATASET PREPARATION

While training and testing on one dataset, it is also imperative to understand how well the model performs on a completely different one. This section describes the procedure used to create a second, much smaller dataset used for evaluating the models in this work.

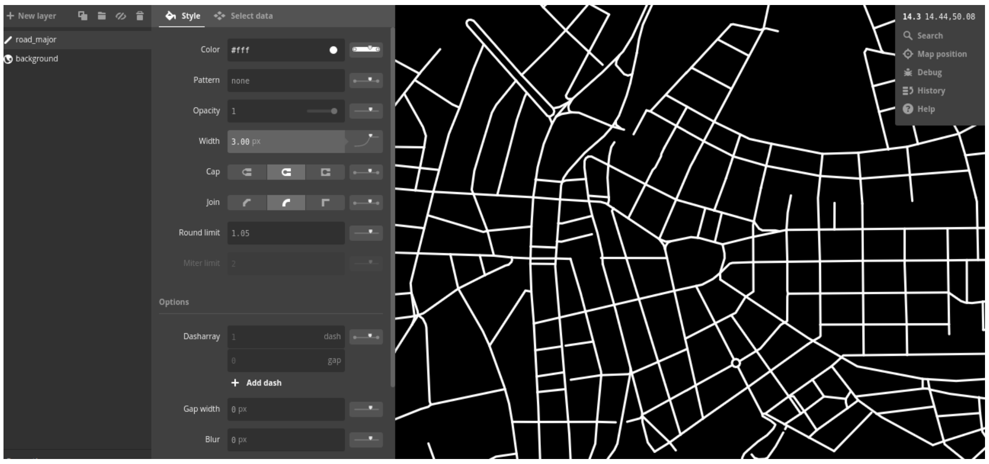
One major gap to overcome here is the generation of ground truth data. Labels in previous works related to learning from satellite/aerial imagery, including [28], [17] relied on using OpenStreetMap, an initiative to create and provide free geographic data (including street maps, among others), to anyone.[31].

In [28], per-pixel labels were generated by rasterizing vector graphic maps extracted from OpenStreetMap. It was noted that this conversion procedure used (thereby also in this work) was an arbitrary choice, also affecting the quality of predictions. The rasterization process used is beyond the scope of this work, and a detailed description can be found in Section 3.1.2 of [28]. A fairly similar procedure is used in [17], with target maps being generated by using highway tags provided for most roads in the OSM database.

To this end, the second dataset was prepared using the Google Maps static API [14]. Satellite images for different regions across the city of Prague were obtained from the SPOT satellites, made available via Google Maps.

For the ground truth, annotated label maps were prepared. A label image is an n-channel image, wherein each pixel on it is an n-dimensional vector. For each such pixel, the sum over all n elements is one, indicating the probability (soft, or hard) of its semantic group - buildings, roads, or background. In our case, the problem is a hard binary classification, and hence, each label pixel is mapped as a 1 dimensional vector, with a value of 0 or 1, indicating its classification as a road, or not.

Two possibilities of annotating the dataset were available - obtain the ground truths from OpenStreetMaps and extract per-pixel labels, or label them manually. Considering the small size of this second dataset, and ease of use of MapBox Studio[26], a mapping platform built on top of OpenStreetMaps data, MapBox Studio, a screenshot of which is shown in Figure 4.2, was used to extract a high quality ground truth dataset.



Color #fff
Pattern none
Opacity 1
Width 3.00 px
Cap
Join
Round limit 1.05
Miter limit 2
Dasharray 1 dash
0 gap
Gap width 0 px
Blur 0 px

14.3 14.44,50.08

FIGURE 4.2 – A screengrab of MapBox Studio, an online platform from MapBox [26] for the creation of customized maps. On the left, we see a space for different annotated layers that can be added onto the map. Here, it contains one such layer: roads, for the road segmentation task.

This was done by first ensuring that the latitude-longitude centering and zoom level of each satellite image from the Google Maps API was replicated using MapBox Studio, which runs on top of OpenStreetMaps.

In addition, a point of concern in the preparation of this dataset was the scale and pixel resolution of the imagery itself. The Google Maps Static API provides a user the option to obtain either satellite imagery, or aerial imagery, depending on the zoom level that one specifies to access the image.

It is a general standard that the 20 zoom levels used by these platforms, [14, 26, 31] correspond to different pixel resolutions of the earth's surface. Google Maps and OSM/MapBox present *mosaics* of 256×256 pixel sized tiles of the earth's surface at different zoom levels. The pixel resolution for the Prague dataset, while needing to be matched to the Massachusetts Roads dataset, presents a problem in that such a resolution is not readily available off the Google Maps API. Additionally, maps taken off these platforms use the Mercator projection [39], and the scaling/pixel resolution is latitude dependent. For this, one resorts to calculating the actual pixel resolution in *m/pixel* for an image, given by:

$$resolution = \frac{156543.03392 \times \cos(\theta)}{2^z} \tag{4.1}$$

where θ is defined as the Latitude of the location in *radians*, and z is the zoom level set on the API, calculated with the assumption that the radius of Earth is

$6378.137km$. This provides a fairly accurate estimate of the pixel resolution of the images obtained from the Static Maps API, and the appropriate zoom level (found to be 15) can be calculated.

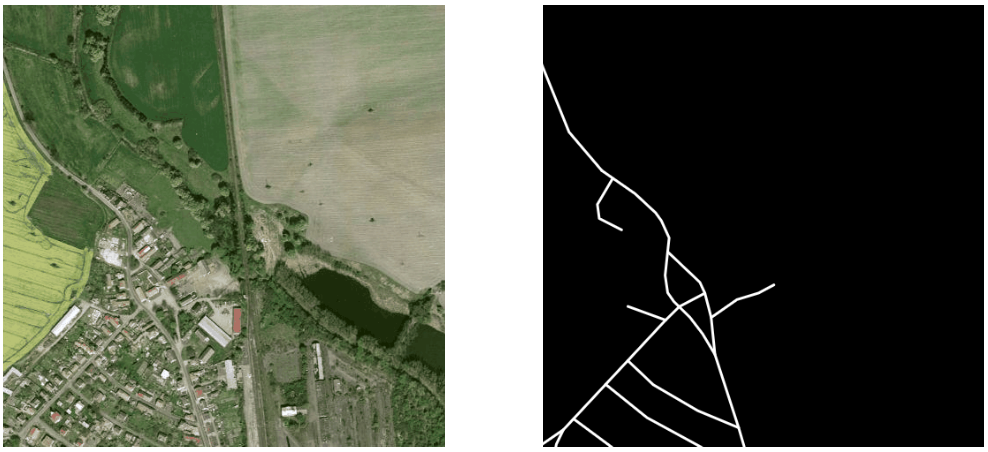


FIGURE 4.3 – Suburban sample of images and labels prepared for the Prague dataset. To note here is the variance in mean color value to the Massachusetts dataset, also visible qualitatively on comparison with the images in Figure 4.1.

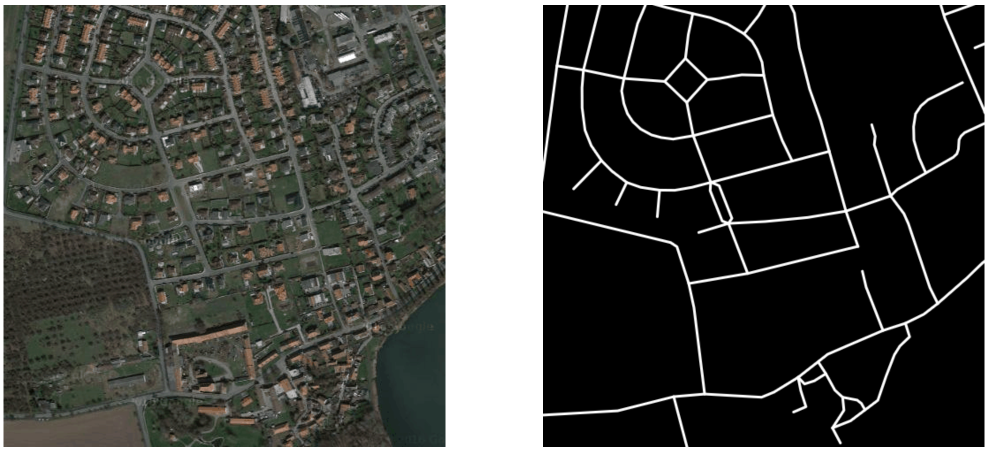


FIGURE 4.4 – Urban sample of images and labels prepared for the Prague dataset.

Following this procedure, a second dataset for the purpose of model evaluation was prepared. The image size was rescaled to 500×500 for convenience, but images of other pixel sizes may also be used in evaluation, so long as the resolution of the earth's surface is kept within an acceptable range of the original training set resolution. 25 such images from across regions in and around the city of Prague, Czech Republic were thus put together to create the second dataset.

In this Prague dataset, one key indicator of a difference was the color of roofs on buildings. Most of the urban structures in the Prague images were roofed with tiles, with a red/brown tinge, noticeably different from the white/gray roofs prevalent in the Massachusetts dataset. This did not play a major role in the model's performance, except for the fact that this could explain a reduced difference in accuracy of predictions between the Prague and Massachusetts test sets.

4.3 PRE-PROCESSING

The above described datasets were augmented by introducing random flips and jitter shifts. By exploiting the invariant features in the dataset, we can artificially expand it. For instance, in the case of road segmentation from satellite imagery, it does not matter which way the image is oriented. Introducing flips and rotations can only help expand the training set, with no possible downside.

Further augmentations can also be done by taking into consideration the orbital parameters of the particular satellite image provider, for instance. In the present case, the training set of 4440 images from Table 4.2 were randomly flipped and jittered to increase the training set size to 9000 instances.

The dataset obtained thus is fed through a pre-processing pipeline to remove unnecessary information. This involved a simple mean-centering, and normalization of the training set, wherein R-G-B image mean values were subtracted from across the entire training set, and divided by the standard deviation before being fed into the pipeline.

II

Development and Approach

CHAPTER 5

Methodology

A primer on the current approach is provided here. The first section presents a naive CNN implementation, followed by a description of the Fully Convolutional Network architecture, a part of which was used to extract features from a pre-trained model.

5.1 PREVIEW

Understanding an image and classifying its content into semantic groups translates into formulating a per-pixel classifier, where we predict a class for each pixel in the image, and extract a semantic map of the entire image.[8] The same idea can be extrapolated into serving for a multi-class classification, where one would consider semantic groups such as buildings, meadows and rivers.[46]

For the present work, we deal with the problem of road segmentation. On one level, there are two primarily different approaches in segmentation: patch-wise labeling [28], or whole image learning [25]. In the former, predictions are made on a central patch of smaller spatial size than the actual image, and many such predicted patches are stitched together to obtain the predictions for the entire image. This is done away with in whole image learning, where no such patches are used, and instead, predictions are made for the entire input image size. In [25], it is shown that whole image learning is akin to a patch-based approach with each batch using its effective receptive field. As it is observed there that whole image training is just as effective as a patch based approach in the segmentation task, it is the preferred method here.

For the present dataset, using images smaller than 500×500 pixels images does not lend itself well for effective training. When smaller image sizes were experimented with on a small scale, a 13% reduction in the accuracy for a basic 4 layer architecture (on the same images) was observed. The performance was drastically lower when tested with images in suburban areas, where roads are sparse and farther apart. On the other hand, computation time in such cases also saw a 3% decrease in this case, due to the change in spatial size. Despite this, images of size 500×500 were eventually chosen for training.

5.2 NAIVE APPROACH

The initial model developed was a simple 4 layer architecture, with the primary aim of understanding how to implement a NN and get it running. For this, the model was first used on a subset of the roads dataset, and later trained on the entire 9000 strong training set.

It is common to increase the number of feature maps produced in each layer as we go higher and higher.[5] The network architecture hence consists of 4 convolutional layers with 32, 32, 64 and 64 feature maps respectively at each layer, followed by a dropout of 0.5. ReLU activations were used in all the models. These feature maps here were finally passed through a softmax function to obtain our probability heatmaps. All filters sizes followed the simplest possible structure: 3×3 sized kernels at each layer, at a stride and padding of 1 throughout. A graph visualization of this network is provided in Figure 5.1.

5.3 TRANSFER LEARNING

Many recent developments in the machine learning/computer vision circles have been driven primarily by the use of common benchmarks - models trained and tested on standard datasets of high variance, that generally lend themselves well to powerful features. [49] The use of transfer learning allows one to use an existing model that has learned fairly generalizable weights trained on a large dataset such as ImageNet [34], and fine tune the network to suit the particular use case. Convolutional networks learn features hierarchically. The generic descriptors that are obtained from a ConvNet hence provide a powerful starting point for fine-tuning existing models to a more specific task.

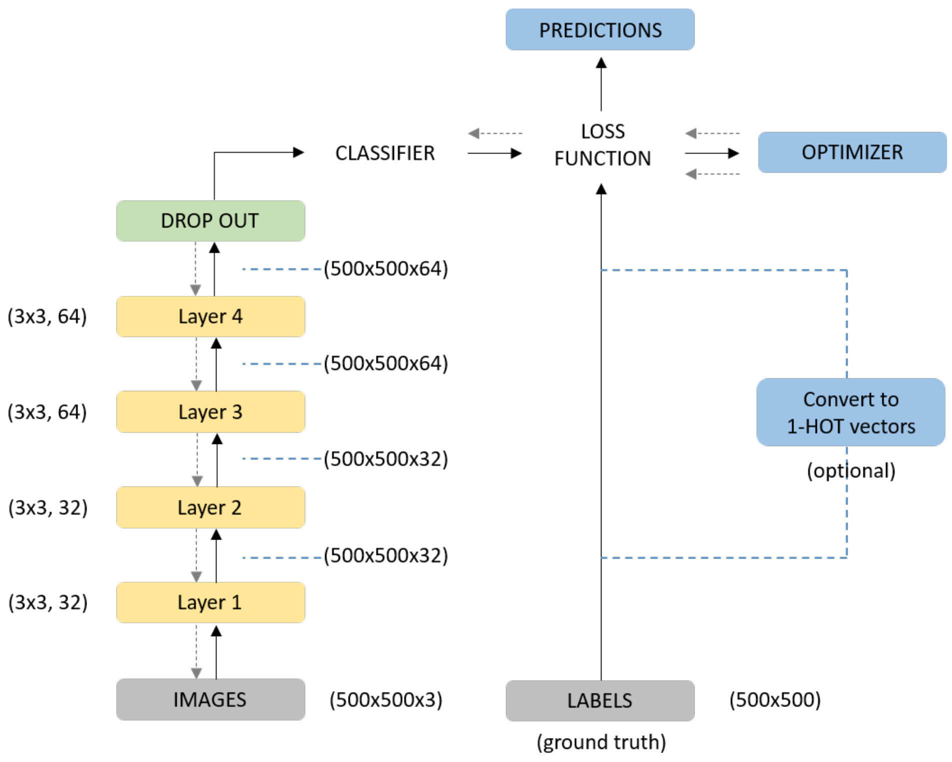


FIGURE 5.1 – Architecture for the 4 layer neural network. Dotted arrows indicate backpropagation pathways, stemming from the Adam optimizer module. Labels were modified to 1-hot vectors depending on the usage of sparse/dense matrices for the loss function.

For this work, the VGG16 model was chosen as the baseline fixed feature extractor. Specifically, the FCN32s and FCN8s architecture, presented in [25] one of which was based on the VGG16-D variant, introduced in [37]. The advantage of the VGG over other networks (that had marginally better performances in some cases [41]), is its simplistic architecture, with homogeneous 3x3 convolution kernels and 2x2 max pooling throughout the pipeline, shown in Figure 5.2.

With an error of 8.5% on ImageNet data from the ImageNet Large Scale Visual Recognition Competition (ILSVRC), it is among the stronger candidates from the different architectures in vgg paper. At this time, the state-of-the-art performance was obtained by the Resnet model [15], with an error of 3.5%. Despite this clear difference, VGG is chosen for the simplicity in structure as outlined above.

The 16 layers of the network are divided into 5 convolution stages, grouped

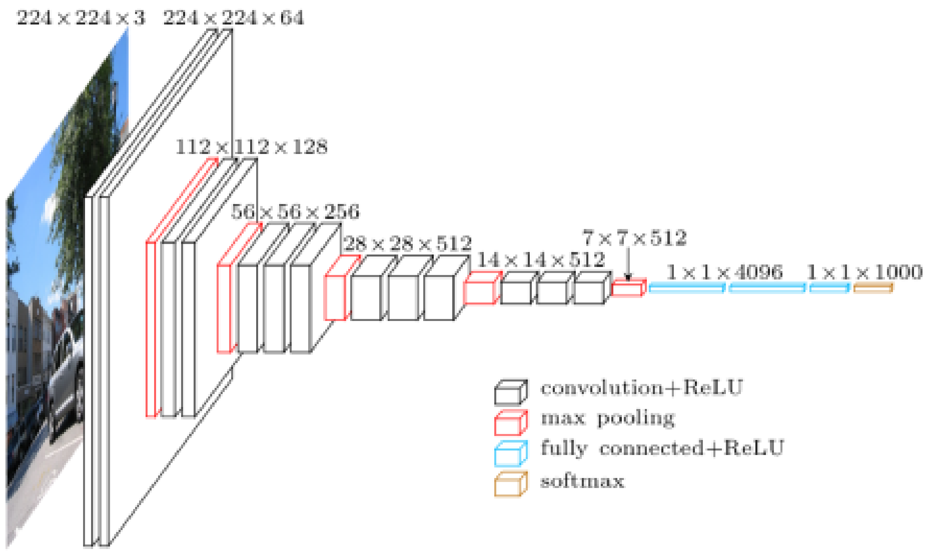


FIGURE 5.2 – The base VGG-16 architecture used in this work. The figure shows the original 16 layer VGG pipeline for images of size $224 \times 224 \times 3$, classified into 1000 classes, as seen in the last softmax layer on the right. The five stages in the layer are seen as convolution and pooling layers grouped together. These are followed by fully connected final layers, before the classifier is applied. Adapted from [10].

in pairs of 2 or 3 convolution layers, followed by 3 final fully connected layers before the softmax classifier. Figure 5.3 shows interpolated weights from the first layer of the VGG network in our case. We see here that the first layer primarily encodes color and direction information. In fact, the same is true for much of the lower layers, all the way up to the 10^{th} or 13^{th} layers.

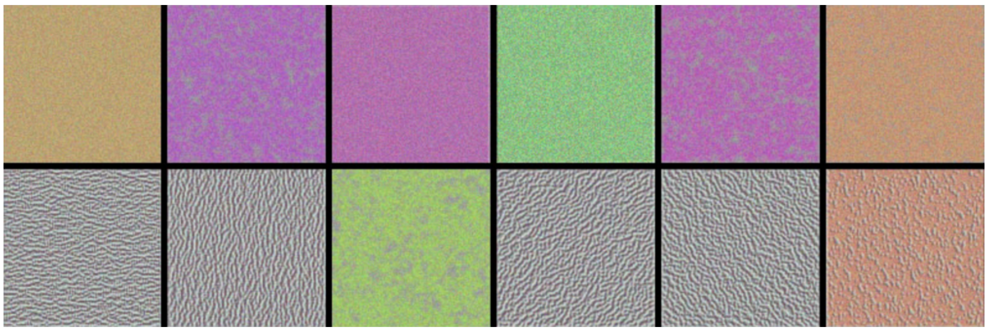


FIGURE 5.3 – A sample of the first layer weights for the VGG-16 model, pretrained on ImageNet. Adapted from [24].

The FCN networks modified the VGG by converting the fully connected layers to their equivalent 1x1 convolutional layers. A novel feature of these models was the introduction of skip connections, which resulted in the 3 models described in [25]. The motivation here is that at higher layers (layers on the right side of Figure 5.4) the network learns from a very sparse feature input, due to the multiple pooling steps through the network. To hint the network in the right direction, features extracted from lower pool layers with a denser structure are added to the final layer features, and the classifier makes predictions on these aggregated features. The use of skip connections resulted in 3 model architectures proposed in [25]: FCN-8s which included skip connections from the *Pool*3 and *Pool*4 layers, FCN-16s which included skip connections from *Pool*4 alone, and FCN-32s, without the use of any skip connections. These are shown at a high level overview, in Figure 5.4.

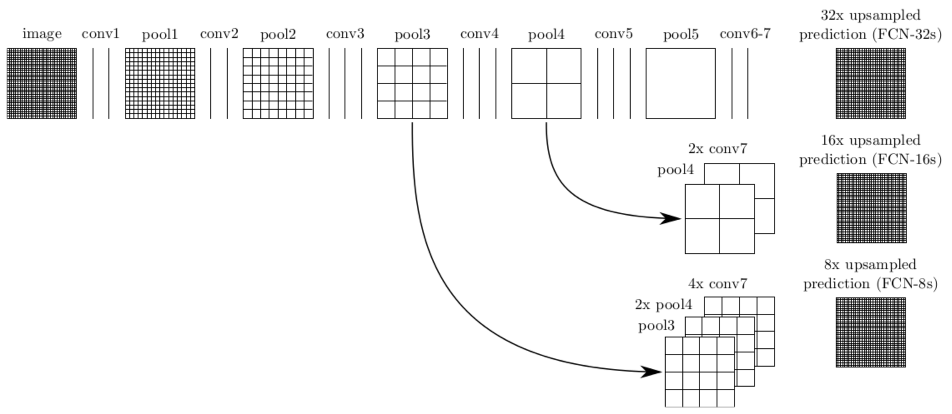


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

FIGURE 5.4 – A visualization of the VGG-FCN architectures. Showcased here is the skip connection architecture introduced in [25]. The image depicts 3 models: the FCN-32s (without skip connections) on top, FCN-16s and FCN-8s variants in the middle and bottom. Convolution layers are indicated by straight lines between the pooling layers, which show spatial density. Adapted from [25].

The advantage with this approach, of discarding the fully connected final layers is that the spatial information - crucial in our context, is retained. In fully connected layers, each neuron is pairwise mapped to every single neuron of its preceding layer, regardless of spatial position. In stark contrast, a convolutional layer connects only to the neurons in its effective receptive field in a deep network.

Most of the parameters of the estimated 140 million for the VGG16 are found in the final fully connected layers. For the purpose of this work, these final layers will be discarded, by simply tapping into the features at the higher intermediate pooling layers. Additional convolutions are applied for each

Layer	Layer output grid (feature maps)
7	125,125(256)
10	63,63(512)
13	32,32(512)

TABLE 5.1 – Layer output shapes at the features extracted and used for the skip connections in FCN-8s.

of the Pool5, Pool4 and Pool3 features, before feeding them into the final classifier. This results in a variant of the FCN-8s model, shown in Figure 5.5.

Also to note here is the use of deconvolution layers [1], which resizes the final scores predicted in a small feature space back to the input image spatial size. This step allows the FCN architecture to take in images of any input dimensions.

Feature maps at three stages from the VGG network are used for the FCN-8s model. For our purposes, we focus on extracting at the pooling layer of the 3rd, 4th and 5th convolution stage (Layer 7, 10 and 13 respectively), with the outputs shown in Table 5.1.

5.4 EVALUATION

Given any input, a binary classifier predicts either of two outcomes : positive, or negative. For the pixel classification problem here, road pixels are considered *positive*, and background pixels, *negative*. The classifier output can be listed as:

- True Positives (TP) : Road pixels are classified correctly.
- True Negatives (TN) : Background pixels are classified correctly.
- False Positives (FP) : Background pixels are mistakenly classified as roads.
- False Negatives (FN) : Road pixels are mistakenly classified as the background.

These numbers are generally represented in a confusion matrix, as shown below, and the standard metrics used to evaluate a classifier are derived as different ratios from it. The models are then evaluated with these metrics on the initially isolated test data. This gives us a proxy measure of how well the model is able to generalize.

[1] Sometimes better known as a convolution transpose, which describes the operation that takes place here

		prediction outcome		
		+	-	**total**
actual value	+	True Positive	False Negative	P′
	-	False Positive	True Negative	N′
total		P	N	

Two metrics relevant here are Precision and Recall, which are defined as:

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

These were chosen as the primary metrics due to the high class imbalance present in the data: roads are generally sparse compared to the background, which covers a far greater spatial area per image. Metrics such as the True Positive Rate (TPR) and False Positive Rate (FPR) tend to be misleading in such cases, but are provided along with the results in Chapter 7 for a comparison anyway. Their definitions are as below:

$$FPR = \frac{FP}{FP + TN} \tag{5.3}$$

$$TPR = \frac{TP}{TP + FN} \tag{5.4}$$

In the case of roads segmentation, precision (correctness) can be understood as the fraction of predicted road pixels that are true roads, while recall (sensitivity) is the fraction of true road pixel predicted.[28] In [28], [35] which deal with road segmentation, the evaluation of these metrics were relaxed to a certain degree. The model predictions were given a leeway of 3 pixels, wherein pixels within that range of a correctly classified road were also considered to be true positives.

The performance of a classifier that produces decision values (i.e., probabilities of a pixel value belonging to either class) can be interpreted more intuitively with the Precision-Recall (Pre-Rec) and Receiver Operating Characteristic (ROC) curves.

Depending on the use case of the problem, one specifies a threshold value between 0 and 1, and this quantifies how cautious or liberal the classifier is in predicting the labels. Too high or low a threshold value, and we increase our risk of mis-classification, resulting in a high number of false positives and/or false negatives in the predictions. A Pre-Rec curve is then a plot of precision vs. recall of the model at a range of such thresholds.

An indication of which threshold value is the best for our model is given by the Break-Even Point (BEP), defined as that threshold value where the precision equals recall. In the current problem, this would be the threshold value to be chosen for the final classifier deployed, unless the problem at hand demands otherwise.

An ROC curve on the other hand, plots the TPR with respect to the FPR. The TPR is the fraction of actual road pixels that are correctly predicted as roads, while the FPR is the fraction of actual background pixels incorrectly predicted as roads.

The area under an ROC curve provides a means of measuring the classifier's ability to discriminate between classes in the dataset. By this definition, maximizing the area (ROC-AUC) leads to a better classification accuracy. This metric is provided for comparison, and care must be taken to note that the datasets we deal with are highly imbalanced.

Finally, the best performing models on each dataset are assessed with the F1-score and Intersection Over Union (IOU), that consolidate the above results into fewer metrics. These are defined as follows:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5.5}$$

$$IOU = \frac{TP}{TP + FP + FN} \tag{5.6}$$

The F1 score is the harmonic mean of Precision and Recall, while IOU is interpreted as the name suggests: the intersection of the actual and predicted values, divided by the union of this set for a specific class.

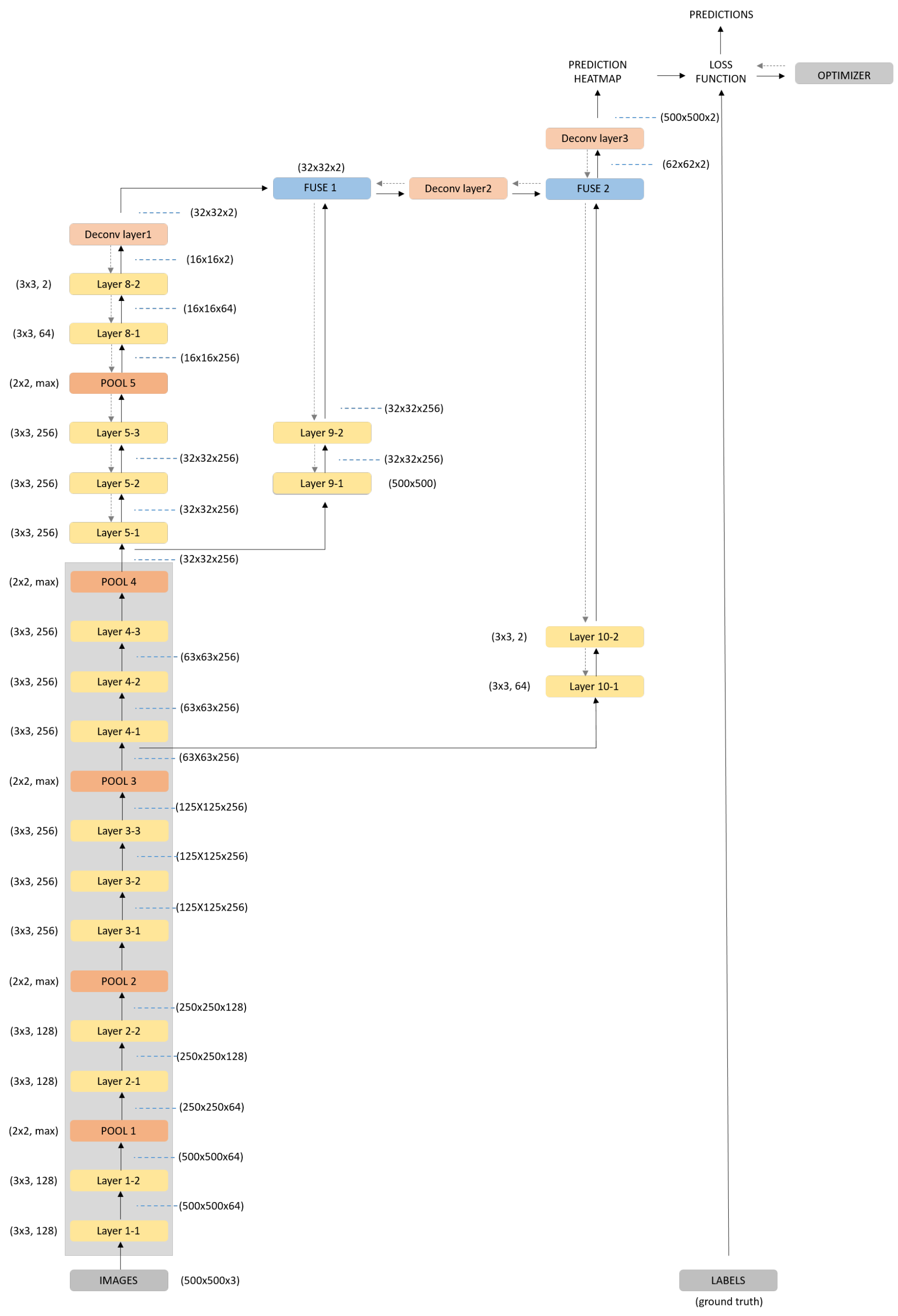


FIGURE 5.5 – Fully convolutional network architecture used presently. Grayed out layers indicate frozen layers (lower layers of the VGG16), where parameters are not learned.

CHAPTER 6

Implementation

This chapter presents details on the segmentation pipeline developed as introduced in Chapter 5. The first section describes the idea of computational graphs, and the second delves into TensorFlow as a deep learning framework.

6.1 computational graph

One way of visualizing mathematical equations is to represent them as graph networks, with each computation forming a node in the graph. An example of this was introduced along with backpropagation in Chapter 3.2.3.

A core concept in functional programming, the backpropagation algorithm can be intuitively understood with this representation. For a deep neural network, computational graphs can get notoriously complex, with a few million parameters atleast. Building a NN from scratch is a daunting task, with many mathematical and implementation specific subtleties to be addressed. For this reason, we generally make use of openly available frameworks that abstract most of the computations away from the user.

6.2 tensorflow

The experiments carried out in this work are for the most part built on top of TensorFlow [1], in Python. TensorFlow[1] is a recently open sourced deep

[1] A major portion of the work in this thesis was implemented using TensorFlow version 0.8. `https://www.tensorflow.org`

Framework	*Caffe*	*Torch*	*Theano*	*TensorFlow*
Language	C++, Python	Lua	Python	Python
Pre-trained	Yes	Yes	Yes(Lasagne)	Inception
Multi-GPU: (Data)	Yes	Yes	Yes	Yes
Multi-GPU: (Model)	No	Yes	Experimental	Yes (best)
Readable source code	Yes(C++)	Yes(Lua)	No	No

TABLE 6.1 – A comparison of currently popular deep learning frameworks. It compares the four major deep learning frameworks on availabity of pre-trained models and possibility of extending the network across a GPU cluster. Reproduced from [19].

learning framework developed at Google, allowing a user to quickly and efficiently implement various algorithms fundamental to neural networks. Given the wide range of functions already made available, as well as the community support, TensorFlow was chosen over other well known frameworks at this time, for reasons made clear in Table 6.1.

TensorFlow is developed for efficient parallelized computations on multiple devices. This makes the framework useful for further investigation even after the thesis, where different architectures can be implemented with very little change across multiple devices. This makes it extremely useful in multi-GPU training, where different devices can be used to store variables for hyperparameter optimization schemes.

Tensors here are defined as multi-linear maps from vector spaces to real numbers, thereby making all scalars, vectors and matrices different instantiations of a tensor. In this sense, they can be thought of as multi-dimensional arrays.

What makes TensorFlow, and most other deep learning frameworks different is that an *"operation"* only defines a node representing the particular operation in a graph structure, and does not execute sequentially. Work-flow is hence divided into two separate phases: the graph construction and an explicit execution phase.

1. **Construction** : Here, one declares symbolic operations that represent equations and functions of the chosen architecture. This includes convolutions, loss and cross entropy calculation, dropout probabilities, pooling, their constituent operations and more.

2. **Execution** : Data is fed into the graph, and the above defined model is run in an executable environment, referred to as sessions.

TensorFlow ships with a convenient web-based visualization tool, TensorBoard. Hyperparameter statistics and parameter distributions were visualized throughout the learning process, whenever necessary. This allowed for a much richer understanding of how the network responded to subtle changes, following which the architecture was reviewed iteratively until acceptable. The data flow graph for each model used was also be visualized, as in Figure 6.1.

6.3 NETWORK ARCHITECTURE

TensorBoard provided easy visualizations of the graphs developed in this work, with each implementation run. This allowed for rapid iteration over different model architectures, and in getting to grips with understanding the computational data flow in a neural network.

6.3.1 *Basic architecture*

The TensorBoard visualization for the model described in Chapter 5.2 is provided in 6.1. The using of customized name scopes in TensorFlow also allows one to efficiently track the many components in a neural network. This becomes important when dealing with larger and larger models.

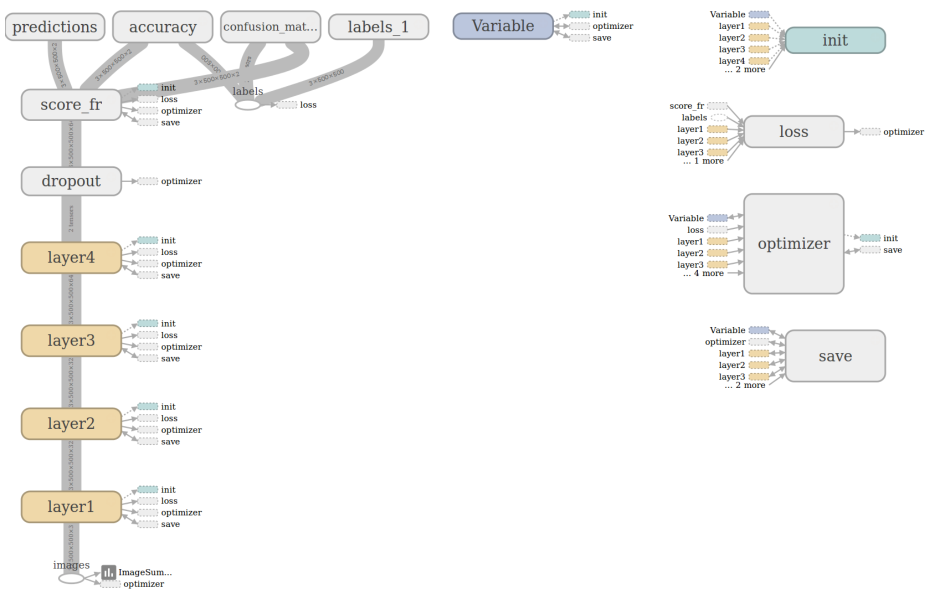


FIGURE 6.1 – Computational graph for the 4 layer neural network. Note how TensorBoard visualizes layer output shapes at different levels, displayed in the greyed connections.

6.3.2 *FCN architecture*

A portion of the TensorBoard visualization for one of the FCN model is provided here in Figure 6.2 The different colors indicate the type of layer, with each of them connected to input and output nodes on the side.

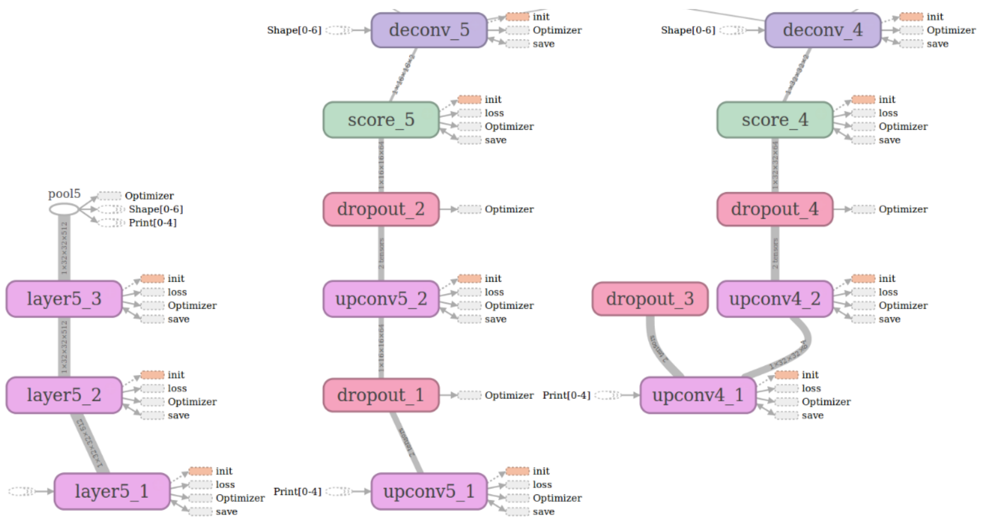


FIGURE 6.2 – Computational graph for a few of the higher layers in the FCN architecture with skip connection.

6.4 HYPERPARAMETERS AND TUNING

With millions of parameters at any given time, neural networks are notorious for causing poor convergence, and hyperparameters need to be tuned methodically. The experiments conducted were always tested by scaling down the dataset to a simpler use case, and the networks were built to overfit on the same. The hyperparameters discussed in the following sections were first introduced in Chapter 3.3, and the procedures laid out there was adhered to as much as possible.

6.4.1 *Learning Rate*

When the network has too high of a learning rate, it is expected that the network blows up. With a high learning rate, exponentials in our softmax function used for the loss calculation caused the network to diverge completely. A general rule of thumb: the lower the learning rate is, better is the performance, with the trade off of a longer training time.[5]

To counter this trade off, the learning rate was decayed by scaling it down periodically. This is to ensure that the optimizer does not diverge

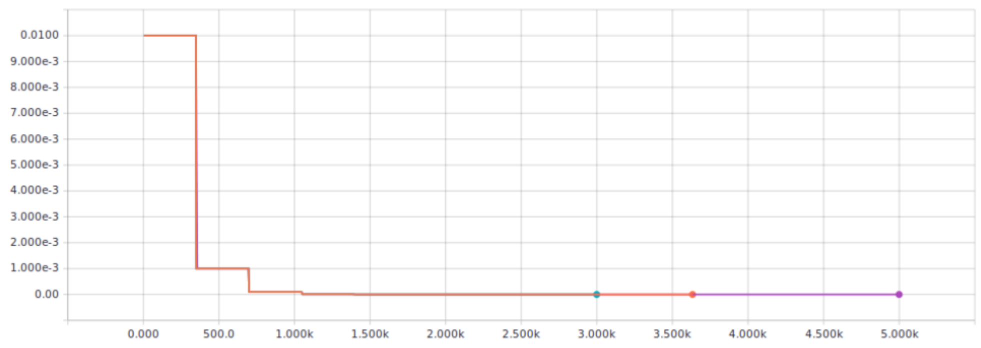


FIGURE 6.3 – Exponentially decaying learning rate over multiple runs of the 4 layer network on a subset of training data. For the FCN architecture, the decay was found to be optimal when activated at around 0.8 epochs.

by fluctuating around the minimum. An exponential decay rate scheduling was chosen here, shown in Figure 6.3. While grid search or random search algorithms exist that allow one to choose a finely tuned initial learning rate with a decay, the learning rate in this work was iterated upon until best performances in terms of loss convergence at a rate of a few hundred iterations was obtained.

6.4.2 *Initialization*

The recommended procedure for parameter initialization is to extract them from a normal distribution of a standard deviation of $\sqrt{2/n}$, [5] with n being the number of inputs to the unit. [2]

Network parameter distributions at different layers can also be visualized for a better understanding of how the network behaves. In tensorboard, these visualizations are called histograms. [3]

The histogram contour plot shows 7 lines, representing the mean and the first three standard deviations away from it. The area between the darkest contours closest to the mean, as in Figure 6.4 represent the fraction of weights in that particular matrix that are within one standard deviation from the mean, in other words, the 68th percentile. The next lines show the 95th per-

[2] A quick sanity check here is to see that upon initialization, loss (without any regularization) should be $-log(1/k)$ where k is the number of classes. This, in our case turned out to be 0.6947

[3] Histograms here are a misnomer - what the graph represents is a distribution of values in the weight matrices or bias values over time, shown as contour plots for different standard deviations from the mean.

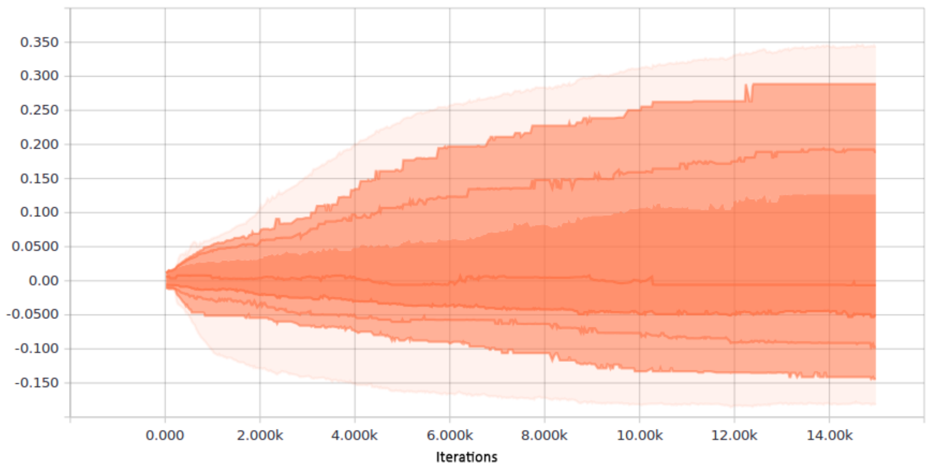


FIGURE 6.4 – Sample of parameter distribution over time for a higher layer in the FCN architecture.

centile. The palest regions extend beyond four standard deviations show the maxima and minima.

Very peaky histograms can be interpreted as meaning that at the particular train step, data fed into the network could have been of a higher variance as compared to other time stamps.

The central line follows fluctuations in the mean. Lower line indicates how the least values in the distribution have changed, and the highest contour indicates the same for the maximum values. This gives us an interpretable visualization for understanding how the network responds online to different data batches. Figure 6.4 depicts the growth in the bias values of one of the final layer convolutions in the FCN-8s model.

These plot regions grow and shrink in vertical width as the variation of the monitored values increases or decreases. The plots may also shift up or down as the mean of the monitored values increases or decreases. For instance, in Figure 6.5, we notice that the weights have stopped growing and plateaued over hundreds of iterations, indicating saturation, wherein overfitting might have occurred either because too many images fed in batches during that period were of similar quality, or the network is simply not learning anymore, due to a low rate of change in the loss function. This was indeed found to be the case, and a decay in the learning rate introduced in this region rectified the same.

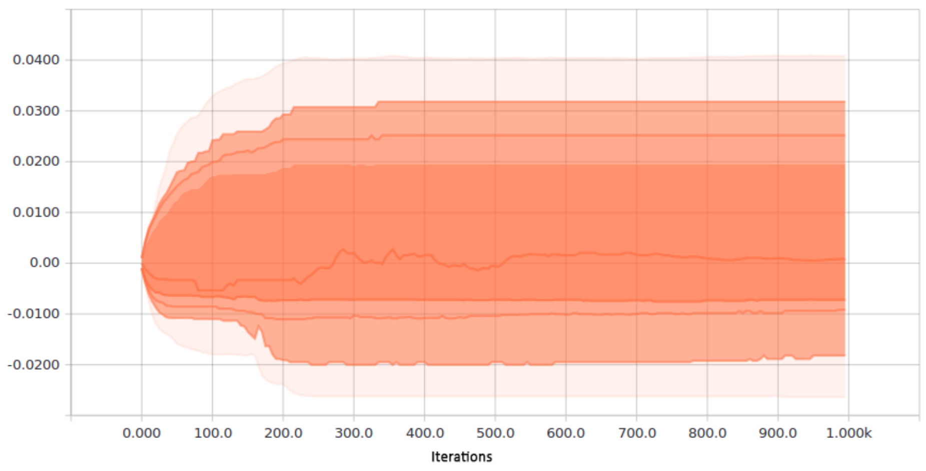


FIGURE 6.5 – Sample of parameter distribution over time for the final layer in our 4-layer network.

The rate of growth of the network over time, visible qualitatively as the vertical width of the contours represents the time taken for weights in different layers to grow. This becomes important especially in the higher layers, where this can be attributed to a saturation of the logistic function. [13]

Another way to track growth of the network is to count the fraction of non-zero elements in the non-linearity activations of each layer, referred to as ReLU sparsity here.

6.4.3 *Data feed*

A minibatch of 3 images was chosen for training, based on computational constraints. The order of these training examples was shuffled after each epoch, to speed up convergence.

6.4.4 *Training and Classifier*

Multiple runs were performed for each model, since hyperparameters were tuned by a manual search in most cases. Figure 6.6 shows the loss function variations for one such training run on the FCN-8s model. Training times varied for each model, especially during the initial hyperparameter exploration phase, when the models were trained on a subset (1500 instances, chosen as an arbitrary fraction of the training set size) of the training set. The classifier chosen was the softmax classifier for all models, introduced in Chapter 3.1.5. Initial experiments also included a variation by using the SVM classifier with a hinge loss, but the softmax was preferred for ease of interpretation.

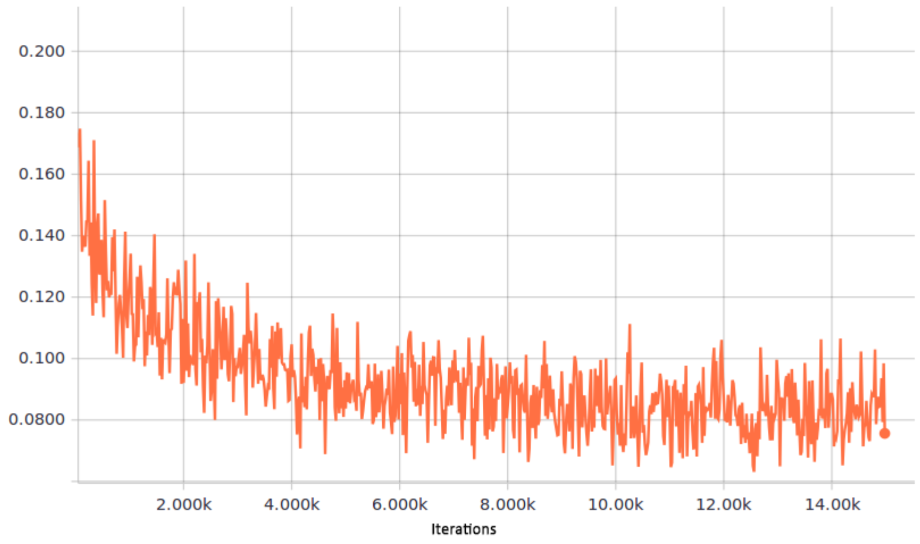


Figure 6.6 – Portion of the cross entropy loss function based on the softmax classifier for the FCN-8s architecture, over approximately 1.5 epochs.

6.4.5 *Adaptive Moment Estimation*

The Adaptive Moment Estimator (Adam) described in Chapter 3.2.3 was chosen for the training process. Similar to most other optimizers, it calculates an adaptive learning rate for all parameters in the network. By storing an exponentially decaying average of past gradients, it has been shown that Adam performs better than other algorithms such as RMSProp. [20]

6.4.6 *Regularization*

A dropout of keep probability 0.5 was used in all models, set as a default value at the higher layers of the FCN models, and for the final layer in the basic 4 layer architecture. An $L2$ regularization term, also described in Chapter 3.1.6, was added to the cross-entropy loss function.

6.5 DEVICE

For the initial experiments, a low-tier CPU with 8GB memory was used, resulting in impractical timelines for the training process. Eventually, a single NVIDIA GeForce GTX 660 became the machine of choice, resulting in a 10x speed increase. With massive parallelization in GPUs being key, using TensorFlow on a cluster can speed up the learning process manifold. Final training time for the FCN32s (no skip connections) and FCN-8s architecture were approximately 26 hours.

CHAPTER 7

Results and Evaluation

This chapter presents the final results on the models developed over the course of this work. The quantitative metrics introduced in Section 5.4 are given, followed by a discussion. In addition, samples of the predicted images are shown with a qualitative discussion.

7.1 metrics

To get a sense of scale for the numbers we are dealing with, a confusion matrix both normalized and in pixel space, is provided in Table 7.1. The normalized matrix is obtained by normalizing with the actual positive and negative values (roads and background respectively). An example of these matrices is shown in Table 7.1 for the 4-layer basic model evaluated on the Prague dataset.

We can see from these matrices the high class imbalance in the chosen dataset. Out of 6250000 pixels, about 97.79% are representative of background, and only about 2.21% represent actual roads.

The Pre-Rec curves and ROC curves for the three models are shown in Figure 7.1 and Figure 7.2 as evaluated on the Massachusetts test set.

The high Pre-Rec values in Figure 7.1 indicate that the Massachusetts test set is too close to the training set, which also necessitates the need for evaluation on a completely different dataset.

		prediction outcome		**prediction outcome**	
		+	-	+	-
actual value	+	TP 32263	FN 105731	TP 0.234	FN 0.766
	-	FP 105773	TN 6006233	FP 0.017	TN 0.983

TABLE 7.1 – Confusion matrix for the Prague dataset when evaluated on the 4 layer basic-model. On the left is the matrix shown in pixel space, with number of pixels, and on the right is the matrix, normalized by actual positive and negative values.

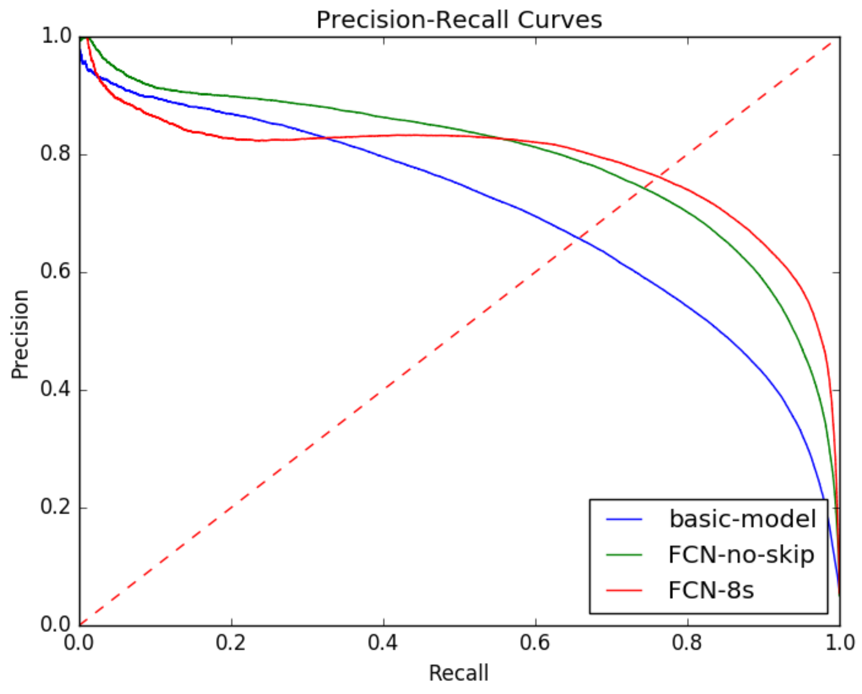


FIGURE 7.1 – Pre-Rec curves for the Massachusetts test set.

This is done by evaluating on the Prague dataset, and the respectives curves are shown in Figure 7.3 and Figure 7.4. Here, we see a more realistic

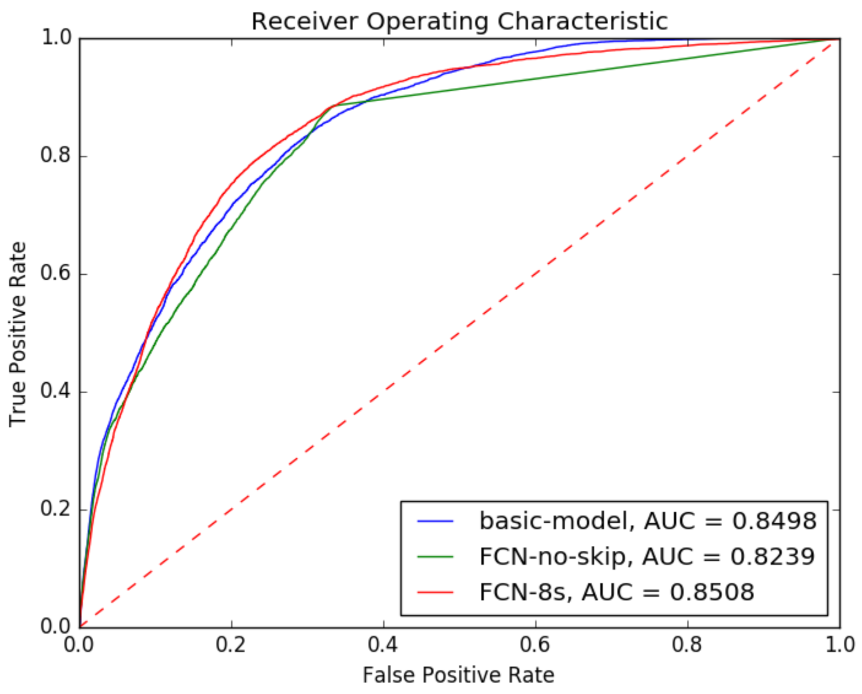


Figure 7.2 – ROC curves for the Massachusetts test set.

representation of the models' generalizability on unseen datasets.

Finally, Table 7.2 and Table 7.3 present a summary of the metrics calculated on the three models for the two test datasets.

Model	*ROC-AUC*	*BEP Threshold*	*Precision*	*Recall*	*F1-Score*
basic-model	0.849	0.274	0.657	0.657	0.657
FCN-no-skip	0.82	0.415	0.742	0.742	0.742
FCN-8s	0.85	0.523	0.762	0.762	0.762

Table 7.2 – Evaluation metrics calculated on the Massachusetts test set. IOU on the FCN-8s model on this dataset was found to be 61%.

For the Massachusetts test set, the FCN-8s model was seen to have the best performance, dominating in both the Pre-Rec and ROC curves. With an F1-score of 0.76, this model yielded an IoU of 61%.

Surprisingly, the FCN-no-skip model, which was similar to the FCN-8s model from this work except for the skip connections inspired by [25], performed better than the FCN-8s on the Prague dataset, with an F1-score of

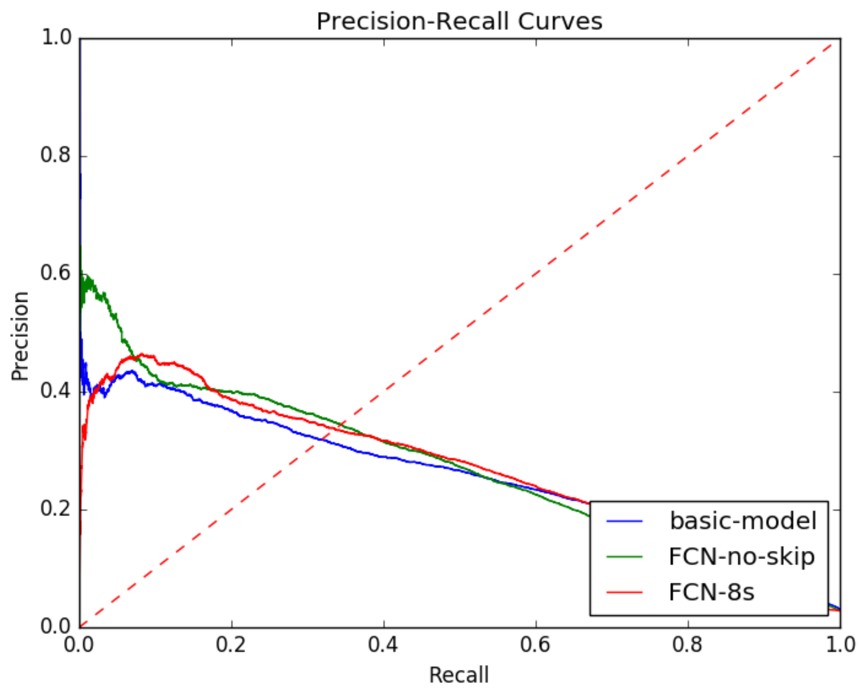


FIGURE 7.3 – Pre-Rec curves for the Prague dataset.

Model	*ROC-AUC*	*BEP Threshold*	*Precision*	*Recall*	*F1-Score*
basic-model	0.7456	0.340	0.233	0.233	0.23
FCN-no-skip	0.7486	0.396	0.319	0.325	0.322
FCN-8s	0.7683	0.487	0.319	0.283	0.30

TABLE 7.3 – Evaluation metrics calculated on the Prague test set.

0.322. The IoU in this case was found to be 20%.

These low values suggest the need for further improvements to be made in the training pipeline, and the same are discussed in Chapter 8.

7.2 QUALITATIVE DISCUSSION

One way to intuitively understand what a neural network learns is to visualize the first layer filters. In Figure 7.5, we see the weights from the 4 layer architecture to consist of mostly straight line descriptors of crosses and edges, and a few central blobs. The visualization here is interpolated for better visual understanding.

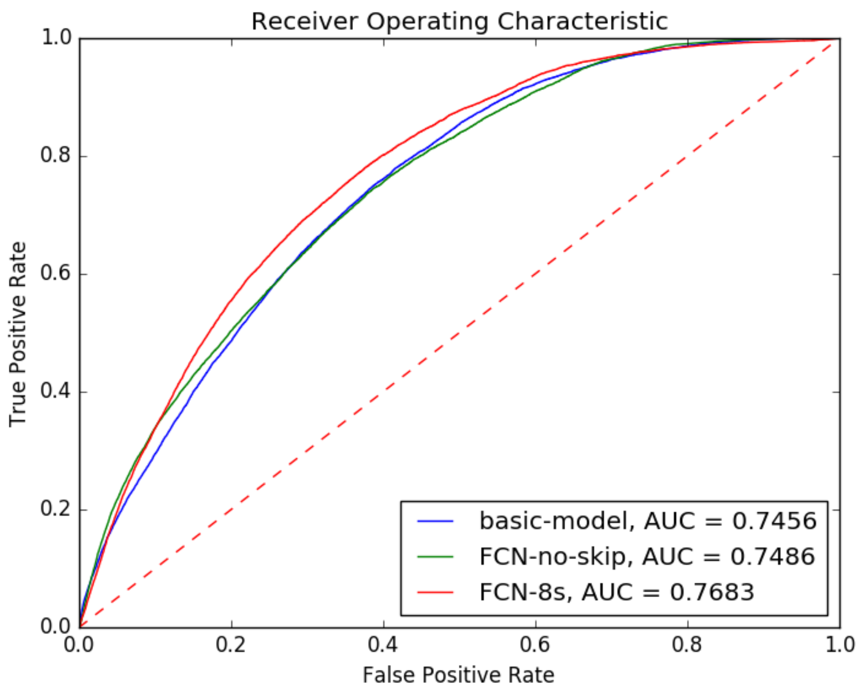


FIGURE 7.4 – ROC curves for the Prague dataset.

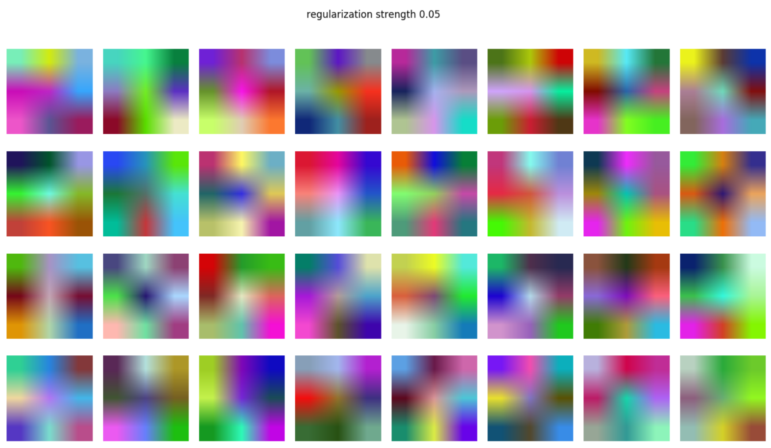


FIGURE 7.5 – First layer weights for the 4 layer network.

Figure 7.6 provides a sample output from the first experiments with the 4 layer architecture. It is easy to see that the network fails to distinguish between roads and buildings with roofs of a similar pixel structure. Notable here is the fact that despite ground truth labels not providing complete information such as the existence of a parking lot, we see the network has already learnt to classify tar as roads, regardless of shape. This already gives us a hint of the advantage of using a machine learned model: especially in cases where manually annotated labels are of extremely poor quality, or error-prone.

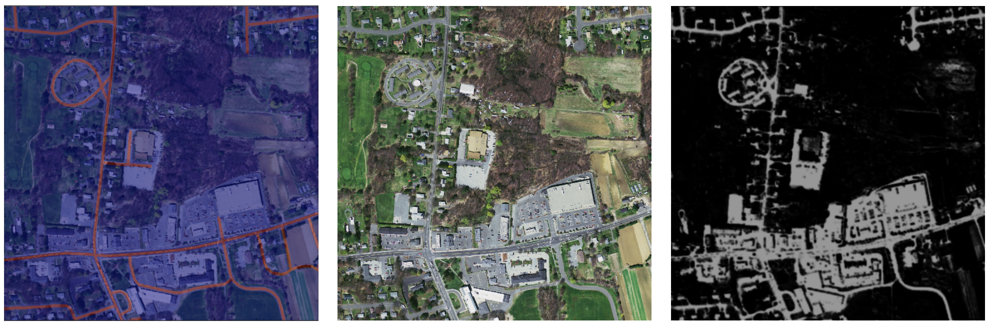


FIGURE 7.6 – Initial results from the 4 layer architecture, indicating poor discrimination between roads and buildings.

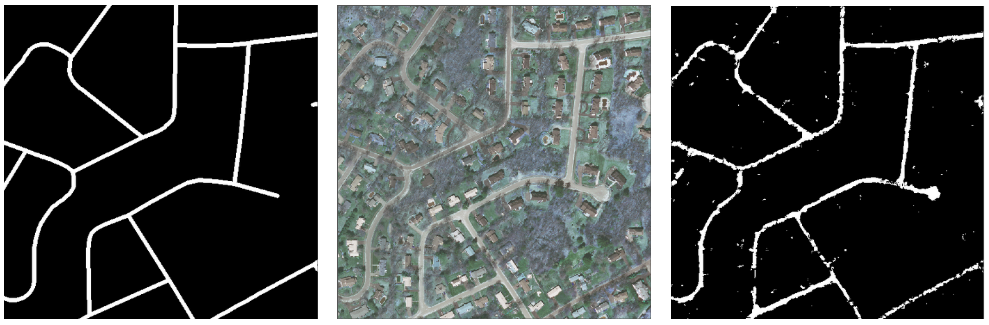


FIGURE 7.7 – Learning the road segmentation on the Massachusetts validation set.

The noticeably good prediction on the Massachusetts validation set in Figure 7.7 indicates one of two possibilities: the validation set examples are very close to the training set, or the model has learnt to predict urban roads extremely well.

We see in Figure 7.8 that the model is unable to handle road regions covered by trees, resulting in discontinuous predictions. The FCN-8s model reached a surprisingly good test accuracy of 93%. A matter of concern here might be that the images from the Massachusetts test and training set are similar to each other in pixel space, thereby making the test set a particularly

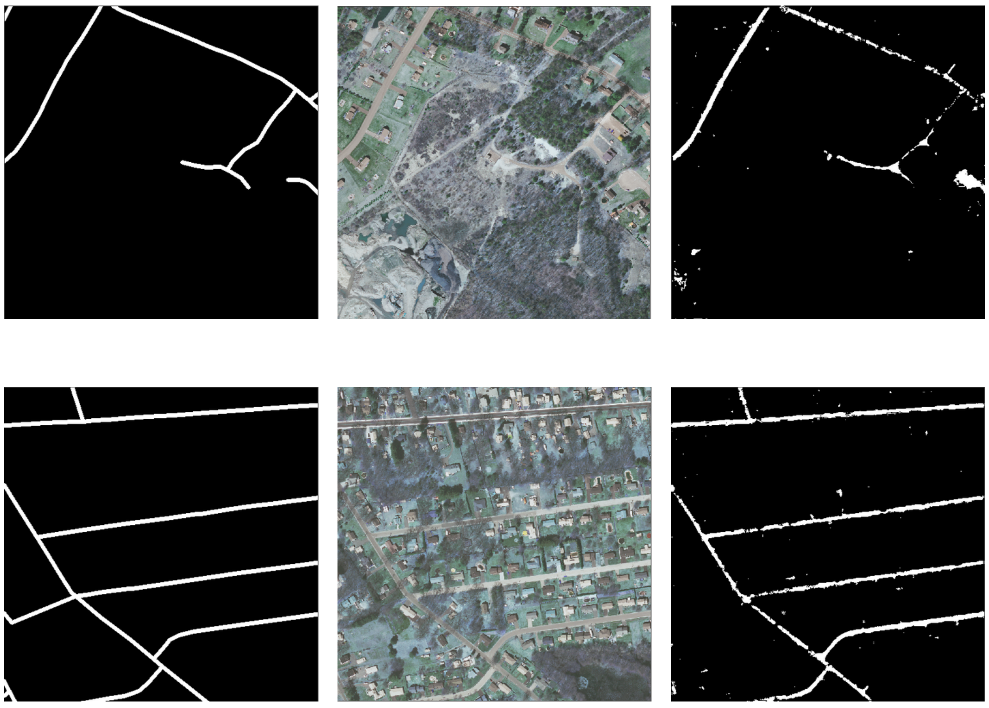


Figure 7.8 – Predictions on a Massachusetts test set example.

easy one. These are shown in Figure 7.9.

The performance in Figure 7.10 is the only one of 25 images that is as expected in the use of skip connections, agreeing with the results of [25]. In Figure 7.11, we see that the FCN-8s model surpasses predictions (visually) of the basic-model on the bottom right, only to our disadvantage. Dark grey-scale roofs are understandably activated as roads. On the other hand, actual roads are more or less well defined in the predictions, with a TPR of 79% in this case.

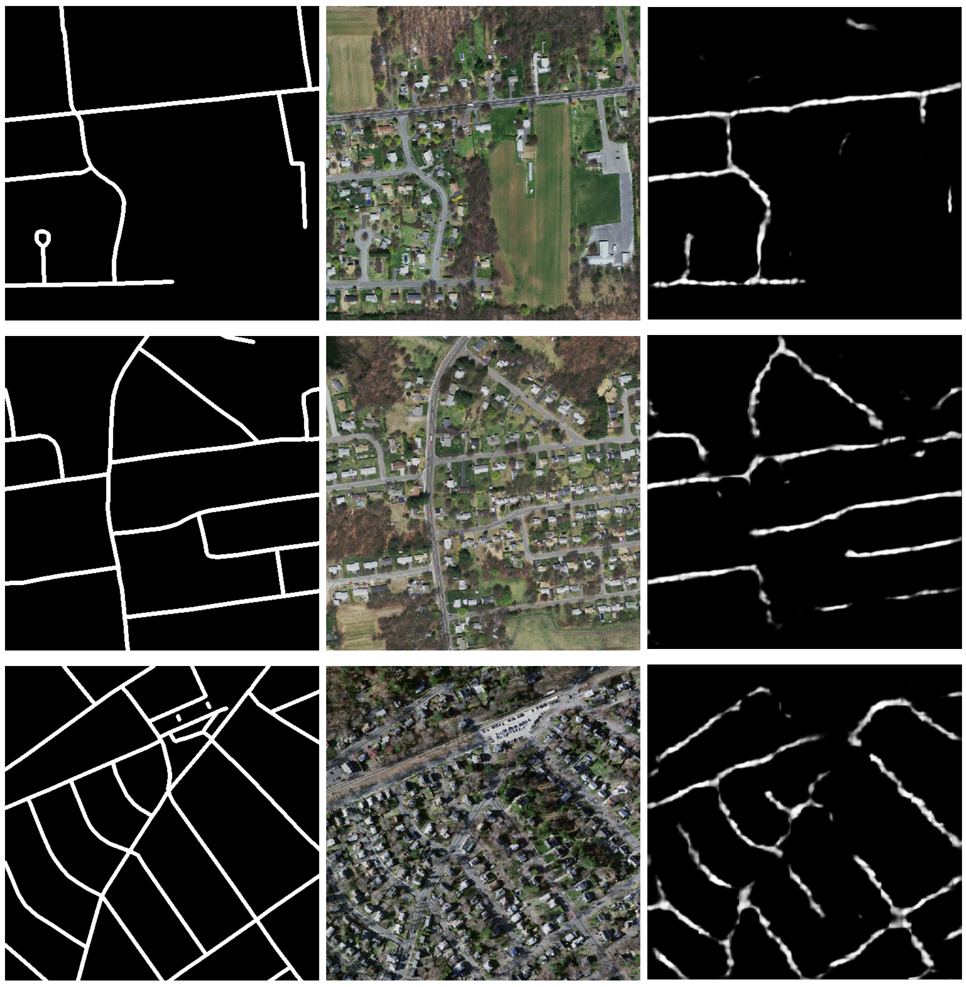


FIGURE 7.9 – Predicted labels for Massachusetts test data from the 4 layer architecture, which are not much worse off.

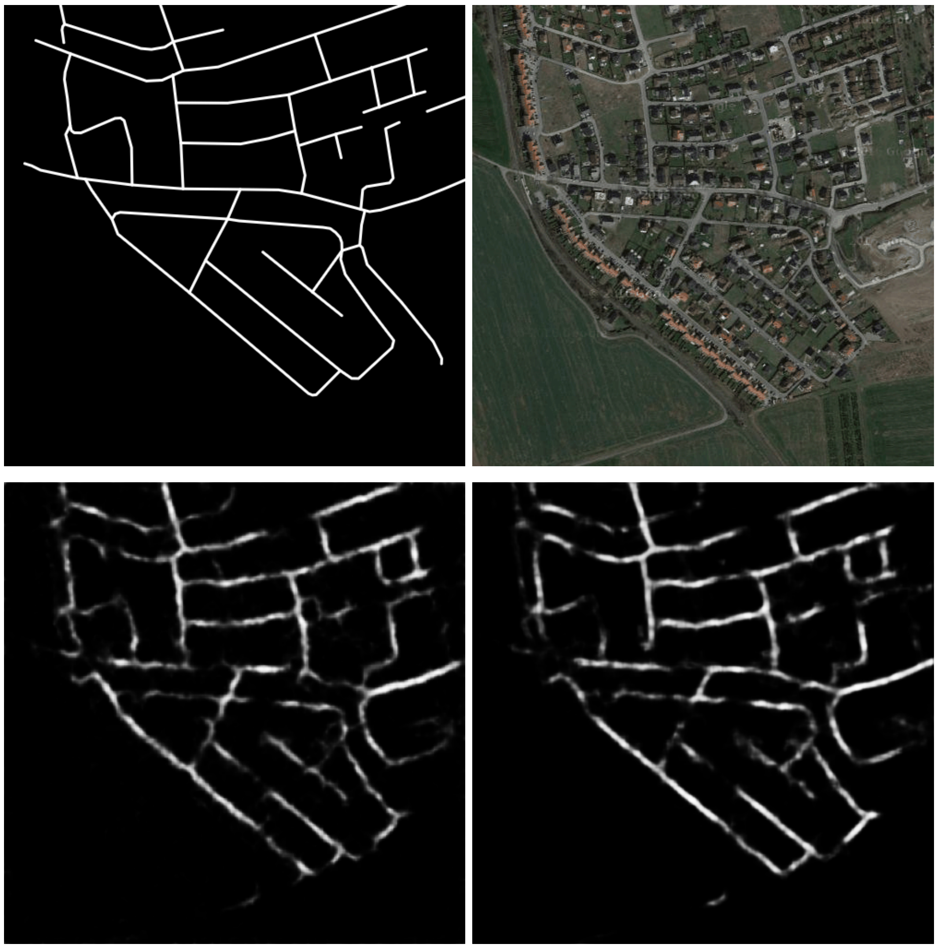


Figure 7.10 – Predicted labels for sample from the Prague dataset. Bottom left shows predictions map by the FCN-no-skip model, while bottom right shows predictions from the FCN-8s model, with a marginal improvement.

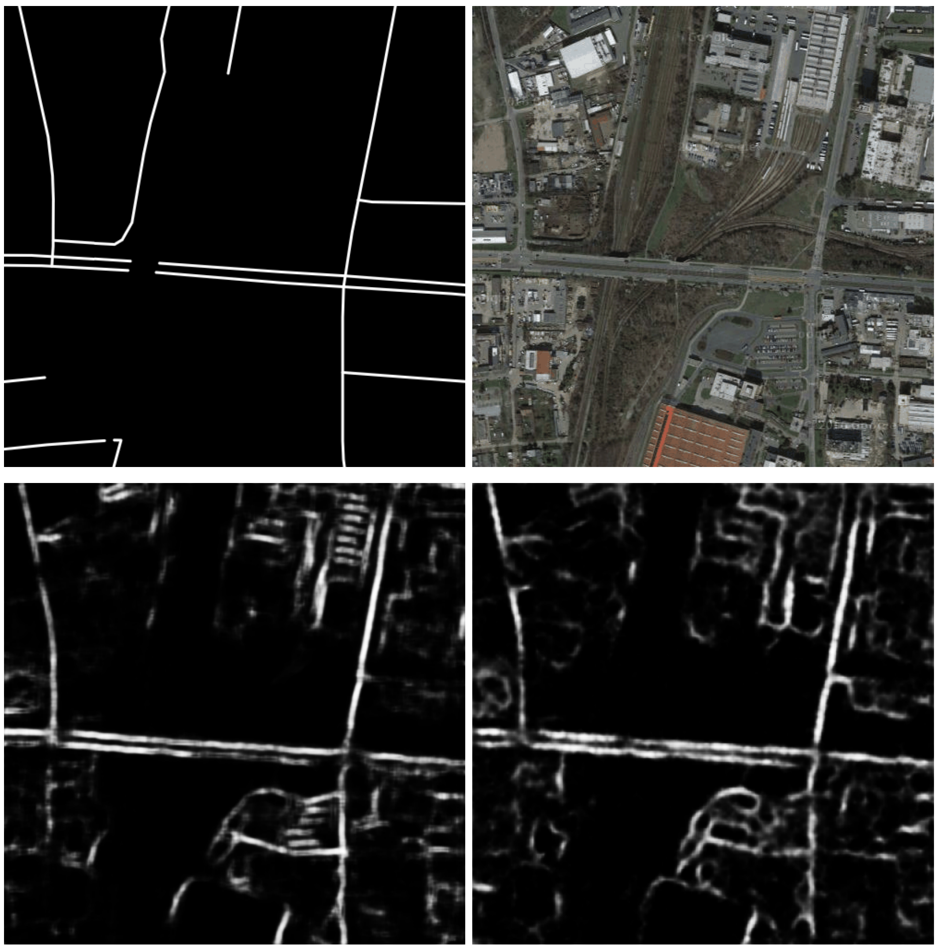


FIGURE 7.11 – Predicted labels for sample from the Prague dataset. Bottom left shows predictions map by the FCN-8s model, while bottom right shows predictions from the basic-model.

III

Concluding Remarks

CHAPTER 8

Future Work

Suggestions for future work to expand upon the work done in this thesis so far are presented here.

The results presented in Chapter 7 have much scope for improvement, with more possible scenarios to be explored. Foremost is the fact that the dataset used in training is fairly small, and better and more varied augmentation schemes can be used. Publicly available resources as described in Section 4.2 can also be leveraged at a larger scale, to obtain more satellite imagery to train the networks on. Care must be taken to ensure appropriate corrections for the Mercator projection, and scaling of images.

Moreover, context in an image is paramount: labels of adjacent pixels are known to be highly dependent deriving from the spatial structure of the images, and this needs to be incorporated into the classifier.

It was observed that discriminating between areas of similar composition required more than just pixel level information - tapping into features describing shapes (polygons for buildings, for instance) so as to clearly identify roads alone. A common workaround for this is be to extract predictions for a smaller region from a given input size, as presented in [28], and [2]. These methods have the advantage of incorporating the context around a pixel into the predictions it makes.

A manual approach was used for hyperparameter fine-tuning in most cases. An algorithmic approach to exploring the hyperparameter space can ensure that the best hyperparameters are selected.

Tying in with the above observation, the problem can also be formulated as a multiclass segmentation problem, dealing with the detection of buildings and roads. Buildings are another important indicator of urban growth and change, and accurately annotated labels for most regions are already publicly available, in the resources listed in Chapter 4.

Varying channels is another important aspect: the images obtained for the Prague dataset included resolutions approximately calculated to be similar to that in the Massachusetts dataset, and were all RGB (3) channeled images. Future research will also have to include more channels, or bands, as discussed in Chapter 2.

CHAPTER 9

Conclusions

The thesis is summarized in this chapter, by briefly reviewing the results and states what has been achieved with regard to the primary objectives specified in Chapter 1.

A major motivation for undertaking this work was to gain an understanding for designing, implementing and evaluating a deep learning pipeline with the focus on satellite image data. This was defined as a semantic segmentation problem, to identify roads in urban areas, in the objectives in Chapter 1. The same are reiterated here:

1. Undertake a brief study on neural network techniques for computer vision.
   Chapters 2 and 3 presented the outcome of this study, with an understanding of how remote sensing techniques are used in earth observation. We also went over state-of-the-art implementations in deep learning techniques along with an introduction to developing a deep learning pipeline for satellite imagery.

2. Build a working deep network pipeline that takes in data to produce semantically segmented maps on the images.
   Chapters 4 and 5 presented in sequence the preprocessing steps taken to create a usable dataset and reasoned a methodology for building the deep learning models in this work. Chapters 6 and 7 showed the implementation details on how the models were built using existing deep learning frameworks, and how segmented maps could be obtained.

3. Compare different neural network structures specified in existing literature. Build on existing models and fine tune them to the present problem (Transfer learning)
   In Chapters 5, 6, a neural network model was built by using the VGGNet [37] model, and extended to the FCN-8s model proposed in [25]. A pre-trained VGGNet model was used to transfer learning outcomes to the satellite imagery problem, on this model.

4. Evaluate the trained network on a different dataset to understand how well it generalizes.
   The models developed were evaluated using standard classifier evaluation techniques, on the unseen Prague dataset, as well as test samples from the Massachusetts dataset. These were compared to provide final assessments on the models.

In addition to the above, taking inspiration from [17], ground truth labels for the Prague dataset were prepared using publicly available resources and frameworks, shown in Chapter 4.

IV

Appendices

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[2] Teymur Azayev. *Object detection in high resolution satellite images*.

[3] Alan S. Belward and Jon O. SkÃÿien. "Who launched what, when and why; trends in global land-cover observation capacity from civilian earth observation satellites". In: *{ISPRS} Journal of Photogrammetry and Remote Sensing* 103 (2015). Global Land Cover Mapping and Monitoring, pp. 115 –128. ISSN: 0924-2716. DOI: http://dx.doi.org/10.1016/j.isprsjprs.2014.03.009. URL: http://www.sciencedirect.com/science/article/pii/S0924271614000720.

[4] Ian Goodfellow Yoshua Bengio and Aaron Courville. "Deep Learning". Book in preparation for MIT Press. 2016. URL: http://www.deeplearningbook.org.

[5] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *CoRR* abs/1206.5533 (2012). URL: http://arxiv.org/abs/1206.5533.

[6] James Bergstra and Yoshua Bengio. "Random Search for Hyper-parameter Optimization". In: *J. Mach. Learn. Res.* 13 (Feb. 2012), pp. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2188385.2188395.

[7] Maureen Caudill. "Neural Networks Primer, Part I". In: *AI Expert* 2.12 (Dec. 1987), pp. 46–52. ISSN: 0888-3785. URL: http://dl.acm.org/citation.cfm?id=38292.38295.

[8] Liang-Chieh Chen et al. "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs". In: *CoRR* abs/1412.7062 (2014). URL: http://arxiv.org/abs/1412.7062.

[9] R.N. Colwell. "Manual of Photographic Interpretation". In: (1997).

[10] *Deep CNN and Weak Supervision Learning for visual recognition*. https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/. Accessed: 2016-07-25.

[11] J. Ronald Eastman. *Guide to GIS and Image Processing Volume 1*. Clark Labs, 2001.

[12] M. Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136.

[13] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATSâĂŹ10). Society for Artificial Intelligence and Statistics*. 2010.

[14] *Google Static Maps API*. https://developers.google.com/maps/documentation/static-maps/intro. Accessed: 2016-07-25.

[15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). URL: http://arxiv.org/abs/1512.03385.

[16] *IRS (Indian Remote Sensing Satellites) - Overview and early LEO Program of ISRO*. https://directory.eoportal.org/web/eoportal/satellite-missions/i/irs. Accessed: 2016-07-25.

[17] Pascal Kaiser. *Learning City Structures from Online Maps*.

[18] Pratistha Kansakar and Faisal Hossain. "A review of applications of satellite earth observation data for global societal benefit and stewardship of planet earth". In: *Space Policy* 36 (2016), pp. 46 –54. ISSN: 0265-9646. DOI: http://dx.doi.org/10.1016/j.spacepol.2016.05.005. URL: http://www.sciencedirect.com/science/article/pii/S0265964616300133.

[19] Andrej Karpathy. *Convolutional Neural Networks for Visual Recognition (Course Notes)*. Accessed: 2016-06-25. URL: http://cs231n.github.io/.

[20] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: http://arxiv.org/abs/1412.6980.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[22] *Landsat Science*. http://landsat.gsfc.nasa.gov/. Accessed: 2016-07-25.

[23] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). URL: http://arxiv.org/abs/1405.0312.

[24] Tsung-Yu Lin and Subhransu Maji. “Visualizing and Understanding Deep Texture Representations”. In: *CoRR* abs/1511.05197 (2015). URL: http://arxiv.org/abs/1511.05197.

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *CoRR* abs/1411.4038 (2014). URL: http://arxiv.org/abs/1411.4038.

[26] *MapBox Studio*. https://www.mapbox.com/data-platform/. Accessed: 2016-07-25.

[27] Dmytro Mishkin and Jiri Matas. “All you need is a good init”. In: *CoRR* abs/1511.06422 (2015). URL: http://arxiv.org/abs/1511.06422.

[28] Volodymyr Mnih. “Machine Learning for Aerial Image Labeling”. PhD thesis. University of Toronto, 2013.

[29] Volodymyr Mnih and Geoffrey Hinton. “Learning to Label Aerial Images from Noisy Data”. In: *Proceedings of the 29th Annual International Conference on Machine Learning (ICML 2012)*. Edinburgh, Scotland, 2012.

[30] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning Deconvolution Network for Semantic Segmentation”. In: *arXiv preprint arXiv:1505.04366* (2015).

[31] *OpenStreetMaps Foundation*. https://wiki.osmfoundation.org/wiki/Main_Page. Accessed: 2016-07-25.

[32] *RADARSAT-2 : High Resolution, Operationally-focused SAR Data for near-real Time Applications*. http://mdacorporation.com/geospatial/international/satellites/RADARSAT-2. Accessed: 2016-08-02.

[33] Raúl Rojas. *Neural Networks: A Systematic Introduction*. New York, NY, USA: Springer-Verlag New York, Inc., 1996. ISBN: 3-540-60505-3.

[34] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *CoRR* abs/1409.0575 (2014). URL: http://arxiv.org/abs/1409.0575.

[35] Shunta Saito, Takayoshi Yamashita, and Yoshimitsu Aoki. “Multiple Object Extraction from Aerial Imagery with Convolutional Neural Networks”. In: *Journal of Imaging Science and Technology* 60.1 (2016, abstract =).

[36] A. L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM J. Res. Dev.* 3.3 (July 1959), pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. URL: http://dx.doi.org/10.1147/rd.33.0210.

[37] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[38] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems* 25. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[39] John. P. Snyder. *Map Projections: A Working Manual.* US Geological Survey, 1987.

[40] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[41] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). URL: http://arxiv.org/abs/1409.4842.

[42] P. M. Teillet. "Towards Integrated Earth Sensing: The Role of In Situ Sensing". In: *Canadian Journal of Remote Sensing* (2002).

[43] *UCS Satellite Database*. http://www.ucsusa.org/nuclear-weapons/space-weapons/satellite-database. Accessed: 2016-08-03.

[44] Jonas Uhrig et al. "Pixel-level Encoding and Depth Layering for Instance-level Semantic Labeling". In: *CoRR* abs/1604.05096 (2016). URL: http://arxiv.org/abs/1604.05096.

[45] Francesco Visin Vincent Dumoulin. *A guide to convolution arithmetic for deep learning.*

[46] Michele Volpi and Vittorio Ferrari. "Semantic Segmentation of Urban Scenes by Learning Local Class Interactions". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2015.

[47] *What is Remote Sensing? A Guide to Earth Observation.* http://gisgeography.com/remote-sensing-earth-observation-guide/. Accessed: 2016-07-25.

[48] Ambro Gieske Ben Gorte Karl Grabmaier Christ Hecker John Horn Gerrit Huurneman Lucas Janssen Norman Kerle Freek van der Meer Gabriel Parodi Christine Pohl Colin Reeves Frank van Ruitenbeek Ernst Schetselaar Klaus Tempfli Wim H. Bakker Wim Feringa. *Principles of Remote Sensing: An Introductory Textbook.* Enschede, The Netherlands: The International Institute for Geo-Information Science and Earth Observation, 2009. ISBN: 978-90-6164-270-1.

[49] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *CoRR* abs/1411.1792 (2014). URL: http://arxiv.org/abs/1411.1792.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Shivaprakash Muruganandham**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Diploma Thesis: **Semantic Segmentation of Satellite Images using Deep Learning**

Guidelines:

The aim of the thesis is to design, implement, and experimentally evaluate a deep neural network for semantic segmentation of urban growth (man-made structures) in satellite images. Beside the deep network, resulting pipeline shall include image pre-processing algorithms to cope with input images of varying quality, resolution, and number of channels. Environmental effects are one of the challenges to be addressed as well (e.g. cloud segmentation). Recommended framework for implementation is TensorFlow for Python.

1. Study the state-of-the-art literature relevant to the thesis [1-4].
2. Explore the TensorFlow framework [4] and use it with Python to design, implement and evaluate a deep neural network.
3. For experimental evaluation use publicly available datasets as in [1] and compare the final performance to [1].
4. Suggested approach: reimplement the VGG [2] into TensorFlow and its fractionally-strided convolution version[3].

Bibliography/Sources:

[1] Mnih, Volodymyr, and Geoffrey E. Hinton. "Learning to label aerial images from noisy data." Proceedings of the 29thfollow International Conference on Machine Learning (ICML-12). 2012.
[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
[3] Noh, Hyeonwoo, Seunghoon Hong, and Bohyung Han. "Learning deconvolution network for semantic segmentation." Proceedings of the IEEE International Conference on Computer Vision. 2015.
[4] Abadi, Martin, et al. "TensorFlow: Large-scale machine learning on heterogeneous systems, 2015." Software available from tensorflow. org.

Diploma Thesis Supervisor: Ing. Michal Reinštein, Ph.D.

Valid until the summer semester 2016/2017

[illegible]

L.S.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická

v.z. [illegible]

prof. Ing. Michael Šebek, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 30, 2016