BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Macura Vít** Personal ID number: **474413**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Software Engineering and Technology**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Interpretability of Deep Neural Networks for Image Segmentation

Bachelor's thesis title in Czech:

Interpretovatelnost hlubokých neuronových sítí pro segmentaci obrazu

Guidelines:

The aim is to propose and implement Deep Neural Network [1, 2, 3] based solution for the SPACENET dataset [4] and compare it with the state-of-the-art results. Due to complex nature of the SPACENET dataset understanding the process of DNN model training is essential to the model design. Various methods of DNN model interpretability [5, 6] should be explored and evaluated experimentally. Comparison with the results of the original competition is integral part of the work. Recommendation: implementation should be done in Python, using Keras [7] and TensorFlow [8] frameworks. Milestones:
1. Onboarding into the DNN state-of-the-art [1, 2, 3].
2. Exploring and understanding the baseline solutions for SPACENET dataset [4].
3. Understanding and implementing a baseline method for model interpretability [6].
4. Experimental evaluation of selected interpretability methods, e.g. LCA [5], using both simple datasets (MNIST [9], CIFAR [10]) and SPACENET.
5. Documenting the research & development effort.

Bibliography / sources:

[1] He, Kaiming, et al. "Mask R-CNN" arXiv preprint arXiv:1703.06870 (2017).
[2] Szegedy, Christian, et al. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning." AAAI. 2017. APA
[3] Goodfellow, Ian, et al. „Deep Learning", MIT Press, 2016
[4] https://spacenetchallenge.github.io/datasets/datasetHomePage.html
[5] Janice Lan, Rosanne Liu, Hattie Zhou, Jason Yosinski. "LCA: Loss Change Allocation for Neural Network Training." arXiv preprint arXiv: arXiv:1909.01440 (2019).
[6] https://github.com/jphall663/awesome-machine-learning-interpretability
[7] https://keras.io/
[8] Abadi, Martın, et al. "TensorFlow: Large-scale machine learning on heterogeneous systems, 2015." Software available from tensorflow.org.
[9] http://yann.lecun.com/exdb/mnist/
[10] https://www.cs.toronto.edu/~kriz/cifar.html

Name and workplace of bachelor's thesis supervisor:

Ing. Michal Reinštein, Ph.D., Vision for Robotics and Autonomous Systems, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **13.02.2020** Deadline for bachelor thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Michal Reinštein, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Czech Technical University in Prague
Faculty of Electrical Engineering

Interpretability of Deep Neural Networks for Image Segmentation

Bachelor thesis

Vít Macura

Study programme: Software engineering and technologies
Supervisor: Ing. Michal Reinštein, Ph.D.

Prague, May 2020

Declaration

I hereby declare I have written this bachelor thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis.

In Prague, May 2020

.....
Vít Macura

Abstract

Since the widespread of space exploration, especially in the commercial sector, there has been an enormous supply of satellite imagery. The amount of data supplied by various satellites raises demand in human interpretation of given data in order to obtain valuable information. One example of such data is the SpaceNet dataset.

The aim of this work is to design and evaluate a deep neural network as a solution to the SpaceNet Road Network Detection challenge based on state-of-the-art published architectures. Due to the complex nature of the SpaceNet dataset various methods of neural network interpretability are explored and implemented.

Keywords: Convolutional neural networks, SpaceNet, image segmentation, deep learning.

Abstrakt

Od rozšíření prozkoumávání vesmíru, především v komerčním sektoru, dochází k vzinku obrovského množství satelitních snímků. Množství dat dodávaných různými satelity zvyšuje poptávku po interpretaci těchto dat za účelem získání hodnotých informací. Příkladem takových dat je dataset SpaceNet.

Cílem této práce je vytvoření a vyhodnocení hluboké neuronové sítě, jakožto řešení soutěže SpaceNet Road Network Detection challenge. Kvůli komplexitě datasetu SpaceNet jsou prozkoumány a využity různé metody interpretovatelnosti pro neuronové sítě.

Kelíčová slova: Konvoluční neuronové sítě, SpaceNet, segmentace obrazu, hluboké učení.

Acknowledgements

Here I would like to thank those who helped me finish this work. My biggest thanks belongs to Ing. Michal Reinštein, Ph.D., without whose constructive critique and great guidance I could not have finished this thesis. Further, big thanks goes to my family and friends for keeping me in touch with reality while working on this thesis. Lastly, I would like to thank Hicoria Hosting s.r.o. for providing me with a private server without which I would not have finished all the necessary computations.

Contents

Introduction

This thesis focuses on design, implementation and evaluation of a deep convolutional neural network on the SpaceNet roads dataset. This chapter offers a general overview of the problem, Chapter 1 provides an introduction to the field of machine learning and deep learning. Chapter 2 then describes the SpaceNet roads dataset, related competition and its solutions. Chapter 3 further focuses on interpretability of machine learning and provides an overview of published interpretability methods.

Motivation

Visual data are the easiest form of displaying information, be it graphs, tables or images. An example of such data are maps. Most of the places and roads on Earth are already well mapped, but challenges emerge when we take into account the validity of some maps. For example humanitarian organizations may require reliable road network maps of areas struck by a natural disaster. In such case, it is neither safe nor fast to manually attempt correction or validation of available data. Updated and reliable data can be acquired through satellite imaging, though. Recent advancements in technology led to an increase in the amount of satellite data available. This further raised the demand for tools that would simplify or automate processing of available data.

Neural networks have become a widely used tool for such tasks, unfortunately, trained models are often viewed as black-boxes that provide a prediction using some sort of internal magic. The field of interpretability focuses on discovery of methods that would help interpret the inner workings of neural networks. In recent years, this field has seen many new publications. Sadly, the proposed methods often aren't evaluated by anyone else than the publisher or simply do not receive enough attention.

Aims

The aim of this work is to design and implement a deep convolutional neural network based on the already published state-of-the-art solutions for the SpaceNet Road detection challenge and to provide an overview of existing interpretability methods along with experimental evaluation of a selected method on proposed network.

Related work

The Machine Learning (ML) field is a very broad area of interest, therefore not many publications cover it as whole, yet some works try to fit in and introduce all general concepts of ML, for example [21, 48, 9]. Usually, publications tend to focus on a more

specific field of ML instead. Some are discussing classical ML, others [3, 73] focus more on the topic of deep learning which has lately become an extremely active field of research, even more so in relation to object detection [20, 49], image segmentation [39, 7, 14] and computer vision in general [4, 5].

Majority of publications regarding road detection tend to work with imagery obtained on ground level, mostly for use in autonomous navigation systems and similar applications, some examples of such are [36, 71]. Publications approach road detection and segmentation from satellite imagery in many different ways, G. Cheng et al. [16] proposes a multi-step approach first segmenting the road region, thereafter generating a road centerline network and finally using an algorithm for overcoming the ineffectiveness of existing methods in the roads' intersections.

An intricate area of interest for the purpose of this work is ML interpretability. Recent publications present new ways to interpret the learning of deep neural networks (DNNs), such as LCA [33], an overview of existing interpretability methods is covered here and in [2, 22].

Chapter 1

Machine Learning and Neural Networks

Machine Learning (ML) is a field of Artificial Intelligence (AI) , which in general addresses the question *Can machines think?*, asked by A. Turing [70], and attempts to answer in such manner to show that machines can, indeed, think [25]. The aim of ML algorithms is to build a mathematical model based on sample data, usually called *training data* or similarly, in order to make predictions or decisions without being explicitly programmed to do so. This is generally achieved by providing the algorithm with a description of features available in the training data. Features are distinct properties of the data, which one can use for example to decide whether a picture is that of a dog or a cat.

Neural networks (NNs) are algorithms based on the idea of imitating the function of a human brain, thus called neural. They were first described long before the 2000s, but were not widely used until the much later introduction of the backpropagation algorithm, a learning methodology which greatly impacted the performance of NNs. Nowadays, NNs are dominant in many AI fields like computer vision, natural language processing, speech recognition or big data processing.

1.1 Perceptron as the Foundation of Neural Networks

The Perceptron algorithm was first introduced by Frank Rosenblatt in 1957 [43]. It is an algorithm for learning a binary classifier - a function that maps its input x, usually a real-valued feature vector, to an output value $f(x)$, a single binary value. In context with neural networks, perceptron can be perceived as a single-layer neural network.

Given the Perceptron parameters, weights vector $\vec{w} \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$, classification $\hat{y}$ for a vector $\vec{x} \in \mathbb{R}^n$ to two classes $\{-1, 1\}$ is performed as

$$\hat{y} = \text{sign}(\vec{w} \cdot \vec{x} + b) \tag{1.1}$$

which is a linear function $l(x) = \vec{w} \cdot \vec{x} + b$ followed by a non-linear function $\sigma(z) = \text{sign}(z)$, thus $\hat{y} = \sigma(l(x))$. A more complex classifier can be created using more perceptrons with shared input, as well as using their outputs as inputs to others, such structure can be seen in Figure 1.1.

This network is the most basic version of a Neural Network (NN). When building NNs, we stack different types of layers to achieve the best results. One layer is, simply put, a collection of neurons, where a single neuron is implemented using a linear function, such

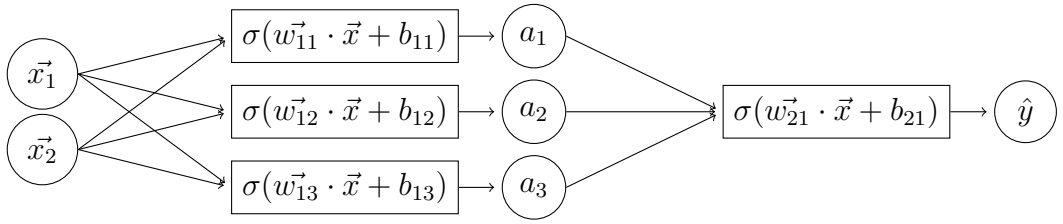

Figure 1.1: Structure of a simple Multilayer Perceptron (MLP). Image created using TikZ TEX package.

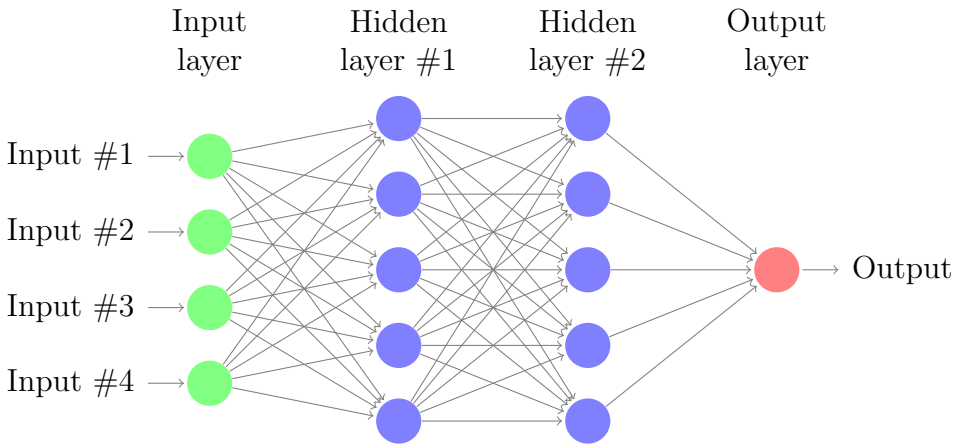

Figure 1.2: Neural Network with 2 hidden layers. Image created using TikZ TEX package.

as previously mentioned $l(x) = \vec{w} \cdot \vec{x} + b$, followed by a nonlinearity like $\sigma(x)$, called an *activation function*.

1.2 Deep Learning

Deep learning is an approach to NNs where more hidden layers are used. Since their widespread, various types of NNs have been introduced, including, but not limited to, Multi-Layer Perceptrons (MLPs), Convolutional NNs (CNNs, ConvNets), Recurrent NNs (RNNs) and Generative Adversarial Networks (GANs).

The most abstract building blocks of NNs are layers. Layer is a collection of neurons, and any regular NN can be decomposed into its layers, which can further be broken down into respective neurons in each layer. These collections of neurons are called layers due to the fact that the output of i-th layer is the input of $(i+1)$-th layer, clearly shown on Figure 1.2. Training data are fed into an input layer, thereafter processed by one or more hidden layers and then finally forwarded to an output layer. If there is more than one hidden layer, the network is referenced to as a deep neural network (DNN).

1.2.1 Activation Function

In ML, activation functions are differentiable nonlinear functions allowing NNs to model complex nonlinear relationships between features. Common activation functions are the *Sigmoid* and the *ReLU function*, shown in Figure 1.2.1. Pros and cons of different activation functions as well as some pitfalls when used along with backpropagation are described in [28].

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad R(x) = max(0, x)$$

Figure 1.3: Common activation functions. *Left:* the sigmoid function. *Right:* the ReLU function.

$$\lambda(x) = C(t - x^2) \qquad L(\hat{y}, y) = I(\hat{y} \neq y)$$

Figure 1.4: Common loss functions. *Left:* the quadratic loss function. *Right:* the 0-1 loss function.

1.2.2 Loss function

Loss function is a function that maps values of one or more variables onto a real number, intuitively representing a kind of cost associated with given values. It is a wrapper around the network's output saying how good or bad did the model score on given sample.

Selecting an appropriate loss function is not an easy task and many recent publications [40, 67, 72, 27] present new types of computing loss. Common loss functions are the *quadratic loss function* and the *0-1 loss function*, seen in Figure 1.2.2.

1.2.3 Backpropagation

In ML, backpropagation is an algorithm used in the process of training of NNs. Backpropagation efficiently computes the gradient of the loss function with respect to the weights of the network for a single training example.

Gradient of a scalar function $f(x_1, x_2, \ldots, x_n)$, notation ∇f, is the direction and rate of the fastest increase. Computationally, it is a vector of partial derivatives of f as its value at index i is the partial derivative of f with respect to x_i.

While the term backpropagation refers only to the algorithm computing the gradient, it is often loosely used to refer to the entire learning algorithm including how the gradient is used. This is often done using gradient descent or its variations like stochastic gradient descent (SGD). Even though combination of backpropagation with the use of SGD is applicable to both regression and classification, even with multiple classes, it does not guarantee the reach of a global minima.

1.3 Convolutional Neural Networks

Contrary to regular networks, most layers in Convolutional Neural Networks (ConvNets, CNNs) are not fully-connected, convolutional layers in CNNs send their outputs only to some neurons in the next layer. They are most commonly applied to tackle problems of computer vision such as image classification and segmentation or medical image analysis.

$$\begin{array}{|c|c|c|c|} \hline 0 & 1 & 3 & 1 \\ \hline 1 & 2 & 2 & 3 \\ \hline 0 & 0 & 2 & 2 \\ \hline 0 & 0 & 0 & 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 3 & 2 & 1 \\ \hline 0 & 2 & 2 \\ \hline 4 & 0 & 1 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 15 & 21 \\ \hline 13 & 22 \\ \hline \end{array}$$

Figure 1.5: Result of the convolution operation applied to an image of size 4x4 with filter size 3x3 and stride 1

1.3.1 Layers in Convolutional Neural Networks

Convolutional Layer

Convolutional layer is based on the operation of convolution in signal processing, it is defined as $(f * g)(n) = \sum_{m=-\inf}^{inf} f(m)g(n-m)$. In deep learning, this operation is performed as element-wise multiplication and addition [6, 47].

In NNs, Convolutional layers are implemented as filters of given size that move across the whole input. At each position of the input a matrix multiplication of overlaid part of the data and the filter is performed. Each filter is described using its size (width) and a stride, which tells how many columns or rows the filter moves each iteration. In Figure 1.5 it is shown that the dimensions of the convolution output are smaller than the input, if the operation should preserve the dimensionality of the data, zero padding is applied to the input, this means that given number of zeroes is appended to the input so that the resulting dimensions are the same.

For 2D grayscale input ($n \times n$ matrix of 0-255 values), the value x_{ij}^l of output on i-th row and j-th column of l-th layer with filter size $k \times k$ is computed as [21] :

$$x_{ij}^l = \sum_{x=0}^{k-1} \sum_{y=0}^{k-1} w_{xy} o_{(i+x)(j+y)}^{l-1} + b$$

Nonlinearities are sometimes considered to be a part of convolutional layers, other times they are explicitly stated as a layer, either as *activation layer* or by the name of used nonlinear function like ReLU or sigmoid.

Pooling Layer

Pooling layers allow downsampling of the input data. Besides the obvious reason being less computational requirements for next layers, this is especially useful for extracting important elements in the data. For example in a high-res image, location of a feature is very precise and adjusting the image would greatly impact the detection of given feature so in order to prevent such impact pooling layer downsamples the image resolution and important features remain dominant. Pooling layers are usually applied after nonlinearities (activations) and require selecting a pooling operation, very much like filter size and stride in convolutional layers. Majority of NNs use pooling with filter size 2x2 pixels applied with a stride of 2, which means that each dimension of pooling layer is halved on the output. Most common pooling operations are max pooling, which finds the maximum value for each section of the input, and average pooling, which computes the average value for each section of the input.

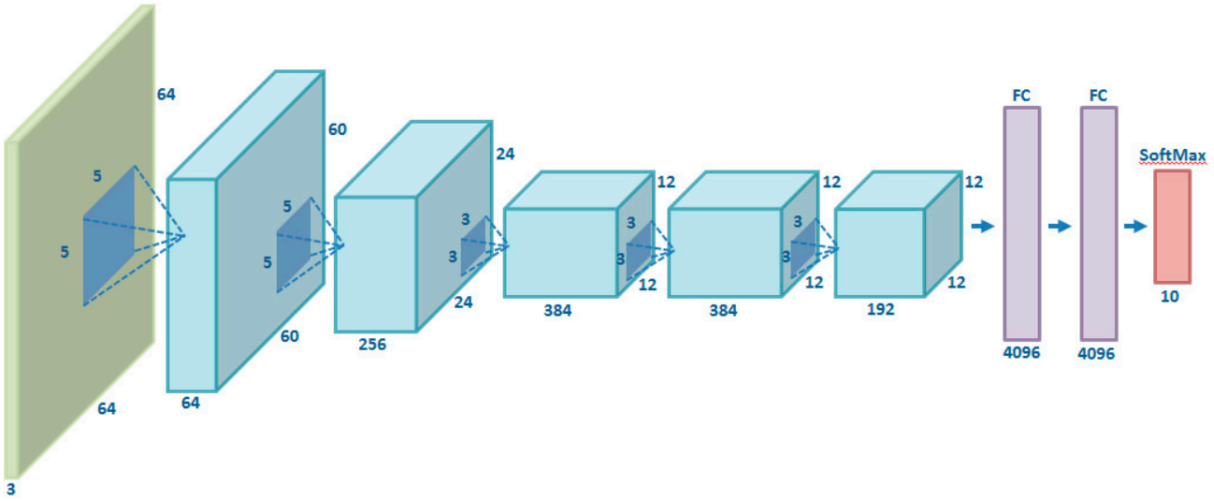

Figure 1.6: Scheme of the AlexNet network. Image from [37], CC-BY-4.0.

Fully Connected Layer

Fully connected (FC) layers are the same as used in regular NNs, not only CNNs. In FC layers, neurons have connections to all activations of the previous layers, ouput is the result of matrix multiplication, same as in Convolutional layers. In comparison to FC layers, a 1x1 Convolutional layer with stride 1 computes a weighted sum of all the input channels with the same weights for each spatial location, while a FC layer stores weights for each spatial location, meaning that a FC layer has significantly more parameters with increasing number of channels.

1.3.2 Common Architectures

While NNs as such are so widely used only recently, some types of architectures have already proven to be more efficient for given tasks than others. This section discusses a selection of important CNN architecture designs and related discoveries.

LeNet

LeNet, introduced by Yann LeCun et al. in 1998 is considered to be the first successful CNN. The network used *tanh* as the activation function and incorporated the usage of average pooling. While this publication is a big breakthrough in the field of ML, CNNs were not widely used until the later success of AlexNet, mainly due to performance restrictions [35].

AlexNet

A groundbreaking paper was published by Alex Krizhevsky et al. in 2012. The network known as AlexNet, shown in Figure 1.6, won the 2012 ImageNet Large Scale Visual Recognition Challenge [45], scoring an error rate more than 10 % lower than any other competitor. This led to an extremely quick adoption of CNNs in ML. The network consists of five convolutional layers, some followed by max pooling layers, all further followed by three fully connected layers. Krizhevsky also puts emphasis on the effect of using *ReLu* as the activation function instead of *tanh* or *sigmoid* [30].

VGG

VGG's[1] architecture is similar to that of AlexNet, only it is deeper. AlexNet consists of 8 layers with parameters, also called weighted layers, in total, while VGG consists of 16 weighted layers. Of these 16 layers 13 are convolutional layers and three are fully connected. The convolutional layers are divided into 5 groups each followed by a max-pooling layer. The deeper convolutional layers in VGG have bigger filter sizes, which means the receptive field is larger hence VGG generally performs better than AlexNet [50].

GoogLeNet/Inception

While VGG achieves great accuracy on the ImageNet dataset, its deployment was a problem due to its high computational requirements. This is what the GoogLeNet attempts to solve by introducing a module called *inception module*, shown in Figure 1.7. This module uses so-called bottleneck layers with 1x1 convolutions for dimensionality reduction which are followed by convolutions of various sizes to capture details at different scales. Furthermore, GoogLeNet replaced previously used fully connected layers at the end with a simple average pooling layer. This leads to better utilization of computational resources inside the network. Overall, these innovations help GoogLeNet achieve over 93% accuracy on the ImageNet dataset while having much lower computational requirements than VGG [65].

ResNet

Previous architectures have shown that with increasing depth of a network the accuracy increases until it gets saturated and then drops rapidly. Residual Neural Networks (ResNets) address this problem by using so-called skip connections or residuals. Skip connections are implemented in the form of two paths in the network, shown in Figure 1.8. One path is as usual, for example three convolutional layers, and the second path is simply one convolutional layer with filter 1x1, these two paths are then merged. The benefit of these connections is retained information which helps fight the problem of vanishing gradient [46]. ResNets introduce so-called conv blocks and identity blocks, where a conv block looks as described here and identity blocks simply omit the 1x1 convolutional layer in second path. ResNets widely popularized skip connections and helped design even deeper CNNs without a compromise to generalization [23].

U-Net

Initially developed for biomedical image segmentation for use where less labeled training data is available, the U-Net architecture yielded yet another insight into what connections and information could be useful for CNNs. The architecture consists of a contracting and an expansive path in the form of so-called encoders and decoders. The contracting path uses encoders to reduce spatial information, thus it decreases dimensions and allows for faster processing of available data, while the expansive path uses decoders to combine learned features with original spatial information using so-called up-convolutions [42].

[1] VGG can refer to multiple architectures described in the same paper, discussed here is VGG-16

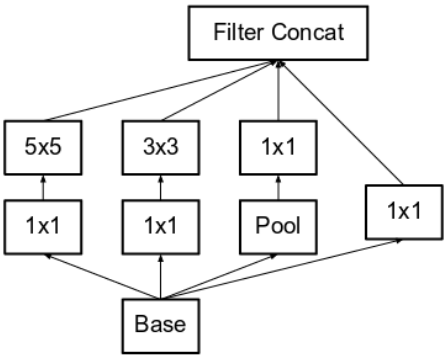

Figure 1.7: Scheme of an inception module from the GoogLeNet architecture. Image from [66].

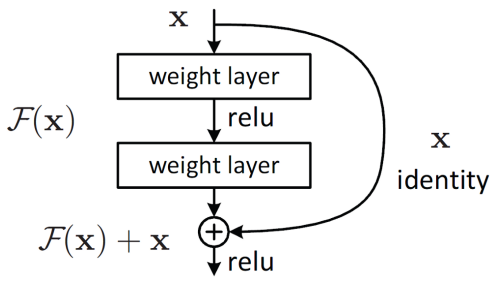

Figure 1.8: Scheme of a ResNet's residual block. Image from [23].

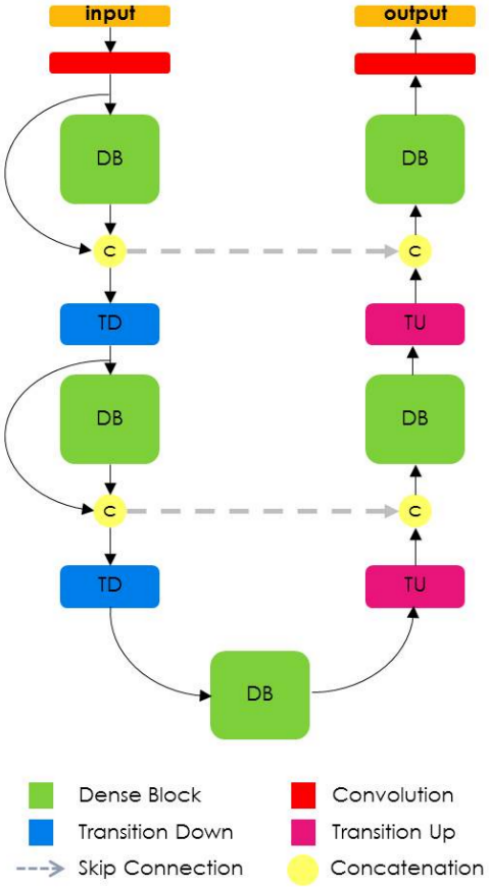

Figure 1.9: Scheme of the Tiramisu network. Image from [26].

DenseNet

ResNets have shown that CNNs can be more accurate and efficient if they have direct connections between layers close to the input and close to the output. Dense Convolutional Networks (DenseNets) take this idea a step further by introducing so-called *dense blocks* in which each layer takes input from all preceding layers directly. In a block with n layers there are $\frac{n(n+1)}{2}$ direct connections in the dense block only compared to n direct connections in a simple feed-forward NN. The increased number of direct connections help reduce the rate of gradient vanishing while reducing the number of parameters. It has also been observed that dense connections reduce over-fitting on smaller training sets [24, 12].

Tiramisu

Fully convolutional DenseNet, also called the One Hundred Layer Tiramisu, is an architecture described by Simon Jégou et al. in [26]. Inspired by the U-Net architecture, this network contains a contracting and an expansive path while incorporating dense blocks used in DenseNets, its scheme can be seen in Figure 1.9. When moving forward through the network, features are learned in given dense blocks and outputs of the dense blocks are then downsampled or upsampled depending on whether they are located on the contracting or expansive path.

LinkNet

LinkNet, based on the findings of previously successful architectures, such as ResNets and U-Nets, attempts to further utilize information learned by the encoder by sharing it efficiently with the decoder after each downsampling block. It was designed for applications in self-driving or augmented reality thus has the possibility of real-time performance on both GPUs and embedded devices. At the time of publication, it can be seen in [13] that the network clearly outperformed current state-of-the-art on both CamVid [10] and Cityscapes [17] datasets.

Chapter 2

SpaceNet Dataset and Challenges

Starting in November 2016 [69], SpaceNet [63] has released five challenges, each along with a related dataset, requiring application of ML methods on difficult mapping challenges. Since the release of the first of these challenges, the topics have been as follows: Building Detection [56, 52, 57, 53], Road Network Detection [58, 55], Off-Nadir Building Detection [59, 54] and Road Extraction and Route Travel Time Estimation [51, 60].

The aim of these challenges is to advance the field of ML as a whole. By providing financial rewards to the best competitors, SpaceNet motivates many new people to compete along. The best performing solutions are open-sourced after publishing results so that anyone can view and discuss introduced methods and architectures [63].

2.1 Dataset Overview

For this work it was decided to work with The SpaceNet Roads dataset (further referred to as *the dataset*), published with the Road Network Detection challenge (further referred to as *the challenge*) [58], because it offers vast amount of data suitable for usage in CNNs and offers a fairly simple evaluation metric - the average path length similarity (APLS) - discussed further in section "Average path length similarity metric". The best solutions of the challenge had APLS score around 0.66 [55].

The dataset contains satellite imagery from the four following areas of interest (AoIs): Las Vegas, Paris, Shanghai and Khartoum. Each one of these cities is represented by images taken in both RGB and multiple spectra, containing overall more than 8670km of labeled road data. The goal of the challenge was to detect Road networks in provided satellite imagery. Summary of the imagery available is shown in Table 2.1. The data are located on an AWS s3 storage with path *s3://spacenet-dataset/spacenet/SN3_roads/train/*.

Data for each AoI are available in four different folders corresponding to four different types of raster data provided by the WorldView-3 satellite [61], the following types of

AoI	Length of labeled roads	Covered area	Number of samples	Size on disk
Las Vegas	3685km	216 km^2	989	38.9 GB
Paris	425km	1030 km^2	310	13 GB
Shanghai	3537km	1000 km^2	1198	48 GB
Khartoum	1030km	765 km^2	283	12 GB

Table 2.1: Summary of available data in the SpaceNet Roads dataset. Data from [62] and custom measurements.

$$APLS = 1 - \frac{1}{N} \sum min(1, \frac{|L(a,b) - L(a',b')|}{L(a,b)})$$

Figure 2.1: The APLS metric. N is the number of unique paths in the graph, $L(a, b)$ is the length of path from point a to point b. [18]

images are available:

- *MS* contains multi-spectral images with 8 channels per image,
- *PAN* contains grayscale (panchromatic) images with 1 channel per image,
- *PS-MS* contains multi-spectral images with 8 channels per image pan-sharpened to resolution of 0.3m,
- *PS-RGB* contains RGB images with 3 channels per image pan-sharpened to resolution of 0.3m.

2.1.1 Average Path Length Similarity Metric

Adam Van Etten, in his article about the challenge, introduces a metric called Average Path Length Similarity (APLS) for similarity matching of ground truth and proposed graphs. Based on the Djikstra's shortest path algorithm, this new metric allows for comparison of both logical and physical topology of the graphs [18]. The article also compares this new metric with other available metrics such as Jaccard index or, more widely used, F1 score, which computes similarity per-pixel, weighting each pixel equally, thus a break in the graph structure is much less penalized than different road widths. Definition of the APLS metric is in Figure 2.1

2.2 Challenge's Published Architectures

This chapter discusses and focuses on the top five solutions of the challenge which had APLS scores of 0.6663, 0.6661, 0.666, 0.6567 and 0.6284 respectively. All of these solutions can be split into two parts, a neural network part and a "image to graph" part building the final graph. These top five solutions are also open-sourced and available for further exploration.

All of the architectures are described by their authors in [64]. The descriptions are written in terms of implementation, used libraries, inspirations and possible improvements and pitfalls.

2.2.1 Albu

The winning submission of the challenge achieved resulting APLS score of 0.6663.

For the neural network part, the solution uses a network based on the U-Net architecture trained on square crops of training images of size 512px with batch size 11 for 30 epochs. The learning rate is dropped a while into training, starting with 10^{-4} for the first 20 epochs, decreasing to 2×10^{-5} for the next five epochs and decreasing once again

to 4×10^{-6} for the last five epochs. The prediction is performed on a full image with resolution 1344px.

For transforming the probability map into graph, a threshold is applied to the probability map to simply differentiate pixels into two classes - *road* and *not road*. The binary image is denoised from small objects and holes and then run through a skeletonization algorithm to thin the road widths. The solution then uses a package called sknw[1] to transform the skeleton into a graph. Further, the author found that skeletonization does not work properly on borders and in order to fix this problem he replicated the border of the binary image and only after that he applied skeletonization.

The author states that the designed architecture might not generalize well for different weather conditions or zoom levels, which is a limitation for all ML solutions. The use of information from adjacent tiles or OSM[2] is a mentioned potential improvement.

Overall, this network won the challenge with APLS score 0.6663. It also scored the highest of all submissions on the images of Khartoum, being the only network that surpassed 0.65 in APLS score in this city.

2.2.2 cannab

The author of the second solution does not go into much detail with the description of his solution, nevertheless some information can be found. It is certain that the solution uses an ensemble of multiple models based on U-Net and LinkNet architectures. To obtain the final graph, it seems that the solution incorporates similar techniques to the first solution. Skeletonization is used to get a graph structure, which is then cleaned off of some noise and obvious mistakes such as small unconnected parts and crosses, finally, it is attempted to fix some broken connections. The author specifies that a separate model for each new city must be trained or at least tuned as he was not able build one model for all the cities.

The resulting network yielded APLS score 0.6661 which was just enough to take the second place in the challenge. Scoring an APLS score of 0.6446 on the Paris images, in this region it beat all subsequent submissions by 0.04 points.

2.2.3 pfr

On the third place followed a submission using an ensemble of nine NNs, all based on an architecture called DPN-92, described in [15]. The output image is cropped by 24px on all sides in order to remove borders which have less spatial context available. The third solution is one of the only two that mention data augmentation prior to training - author describes using squared crops of training images which are then randomly flipped or rotated by multiples of 90 degrees.

Similarly to the previous designs, thresholding is used to obtain a binary image which is further denoised, skeletonized and smoothed using shapely library[3] to produce the final graph structure. Same as previous competitors, short isolated and dead-end segments are removed.

The author states that his network could be possibly improved by using multi-spectral images instead of only RGB or implementing a more sophisticated vectorizer in place of

[1] `https://pypi.org/project/sknw/`

[2] OpenStreetMap - `https://www.openstreetmap.org/`

[3] `https://pypi.org/project/Shapely/`

simple thresholding. It is also necessary to manually provide six calibration coefficients when applying the algorithm to a new area of interest.

The final APLS score of this network is 0.666 which resulted in taking the third place in the challenge. This network had a best score for two separate cities, Las Vegas and Shanghai, having scored 0.8009 and 0.6646 for these cities respectively.

2.2.4 selim_sef

Submission that ended up taking the fourth place in the challenge consists of an ensemble of six models based on U-Net and LinkNet architectures to generate the image mask. The author notes that it is crucial to use both cross-entropy and soft dice in loss when dealing with images.

For transforming the image mask to a graph the author used the sknw library. This solution is the second one mentioning data augmentation prior to training - the author used normalization of the contrast of the images prior to learning and states that this might have had a notable impact onto performance.

Using multi-spectral images, adding post processing and incorporating a loss term that punishes topology violation are some potential improvements mentioned by the author. He also says that the current approach does not handle bridges and multilevel intersections very well.

2.2.5 fbastani

The solution on the fifth place uses square image crops of size 256px for training and targets output image masks with resolution of 128px. The network uses U-Net like architecture, similarly to all the previous solutions, and also starts with learning rate 10^{-3}, decreasing it during training. The author describes using an ensemble of four trained model per city and averages the segmentation output to obtain a single image mask.

To transform the image mask into a graph, thresholding is used to obtain a binary mask, which is then thinned. Further, borders are copied on all sides as a form of padding optimization in attempt to connect roads more precisely. Finally, small components and dead-ends are removed and roads ending close to each other are connected.

The author discusses that since road matching threshold in APLS is very strict it could be helpful to align vertices along the road in the satellite imagery as a form of post-processing. He also says that, similarly to the fourth submission, overpasses and underpasses are not handled well.

2.2.6 Summary

All architectures can be divided into two parts: a neural network part and a part transforming probability map into graph structure using thresholding and skeletonization. Aside from the obvious difference being in chosen network structure, the solutions differ in the size of input, data augmentation, and details of how the final graph is created. Some authors mention interesting potential improvements to their algorithms, some of which could also be used also other networks.

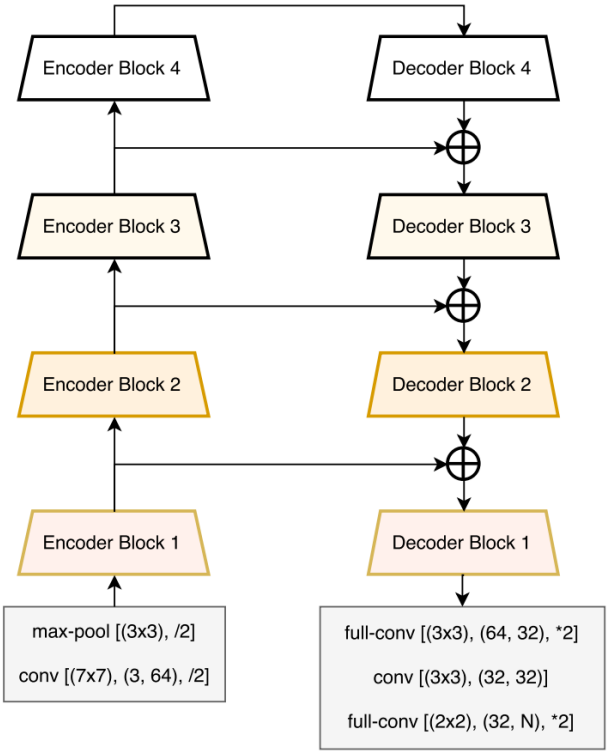

Figure 2.2: The original LinkNet architecture. Image from [13].

2.3 Designed Network

The architecture for experimental evaluation is heavily inspired by a baseline solution of the SpaceNet Road Network Detection challenge. It is based on the LinkNet architecture, first described in [13].

The SpaceNet data have to be manually downloaded from the Amazon Web Services (AWS) s3 location, described in a bit more detail in [62], note that the current AWS s3 structure is different from the one being described on the website.

A data generator is used to feed data into the network. In each batch, at first, image masks, used as the ground truth, are generated from the original SpaceNet dataset and then fed together with a crop of the input image into the network.

The final layer of the network returns an image with values in between 0 and 1, which, when multiplied by 255, yield the prediction mask.

The original LinkNet architecture is shown in Figure 2.2 for reference. The input to the designed network is first fed through two sets of blocks with a convolutional (Conv) layer, followed by a batch normalization layer (Bn), followed by a ReLu activation layer (Relu), together abbreviated as "ConvBnRelu", and then passed further to five encoder-decoder pairs. In this section of the network, there is a set of encoders, each downsampling the image to a smaller spatial dimension, this resembles the contracting path of a U-Net-like network as mentioned in Section 1.3.2. The output of the last encoder is fed to the first decoder which upsamples the image and then further combines with output of the encoder of corresponding depth, this resembles the expansive path.

Each encoder consists of a downsampling ConvBn layer, followed by one or multiple residual blocks. The decoders are constructed as one ConvBnRelu layer, one DeconvBnRelu layer and another ConvBnRelu. In the Deconv layer, transposed 2D convolution is used instead of convolution in order to upsample the image.

After the encoder-decoder section, the values are passed through a single convolutional layer with a single filter of size 1x1 to reduce the number of channels to one, and finally a sigmoid activation follows which yields per pixel values between 0 and 1, which, when multiplied by 255, result in the predicted image mask.

Chapter 3

Machine Learning Interpretability

Interpretability of ML is a field that attempts to provide better insight into the process of training. The goal is to understand the process more in order to make better adjustments to networks' architectures and hyperparameters, as one would call it in regular programming, debug the code.

3.1 State-of-the-art Methods

Even though the field as such exists as long as ML itself, it has become an active area of research quite recently [19, 38, 22], only after the spread of NNs people really realized that it is great when programs achieve something without being explicitly told how to do so, but it is even better when we can understand why the results were such as they were and what would make the program act differently.

3.1.1 Model Agnostic Methods

Model agnostic methods attempt to separate explanation from a specific ML model. This means that any ML model could be used and the interpretability methods could still be applied regardless of the algorithms and methods used to construct given model.

Partial Dependence Plot

Partial dependence plot (PDP) simply shows the effect a feature has on the predicted result of a ML model [8, 38]. PDP provides an intuitive insight and can also compute the correlation of two features, e.g. age and weight when predicting the probability of a person being obese. Some disadvantages are the ability to compute PDP only for 2 features due to restrictions of human imagination (for n features it would be a n-dimensional graph) and more importantly so-called *assumption of independence* - it is assumed that the features for which we compute the partial dependence are not correlated with other features, which is obviously false. Unfortunately, it is fairly inconvenient to apply in this work since NNs learn features in their hidden layers and these features would have to be uncovered.

3.1.2 Example Based Explanations

Example based explanations are methods that attempt to explain the behavior of ML models in relation to training data. Example based explanations do not create summaries

of features, instead they help understand the model by selecting instances from the training dataset. This works well with images and similar structured data, as they can be easily represented in a meaningful way for humans [11, 38].

Adversarial Examples

Adversarial examples are data instances with intentional changes that are designed specifically to make the model yield a false prediction. While this is not directly a method of interpretability for the training process, adversarial examples provide an insight into what could be a security issue with neural networks [68]. Consider a face identification software for example - one could possibly wear glasses of a very specific shape or color and as a result this person could be either simply misinterpreted as another random person, or even worse, might be able to imitate another person's look in the "eye" of the network [38].

3.1.3 Neural Network Interpretability

Neural networks are becoming more popular and widespread every day, this raises demand for methods of interpretation of NNs' behavior. Since NNs can have even millions of parameters, depending on the architecture, it is humanly impossible to understand and follow each and every step of the training process or prediction. This demand has led to introduction of varied interpretability methods using either learned features or gradients of the NN.

Learned Features

When working with many kinds of ML algorithms, for example SVMs[1], different features based on available data have to be manually created. Neural networks learn these features by themselves in hidden layers. One branch of interpretability attempts to uncover these features and visualize them for human understanding. An example of such visualization is activation maximization. The idea behind activation maximization is fairly simple - the goal is to generate an input image that maximizes the output activations of given filter. This helps understand which patterns or textures help activate a particular filter [29].

Lucid

Lucid[2] is an open-source collection of tools for research of feature visualization using the TensorFlow [1] framework. It implements visualization out of the box on many different models such as AlexNet, VGG, ResNet or Inception, all available in the Modelzoo library provided as a part of Lucid. Sample code is available in the form of Google Colaboratory notebooks which can be run in the cloud with no initial setup.

The visualizations generated by Lucid tell us that the network first looks for different shapes and textures, seen in Figure 3.1. Later, deeper, it seems that the network is looking for more profound patterns in the data, seen in Figure 3.2. Sample images were generated using one of Google Colaboratory notebooks[3] mentioned earlier.

[1] Support Vector Machines

[2] `https://github.com/tensorflow/lucid`

[3] https://colab.research.google.com/github/tensorflow/lucid/blob/master/notebooks/modelzoo.ipynb

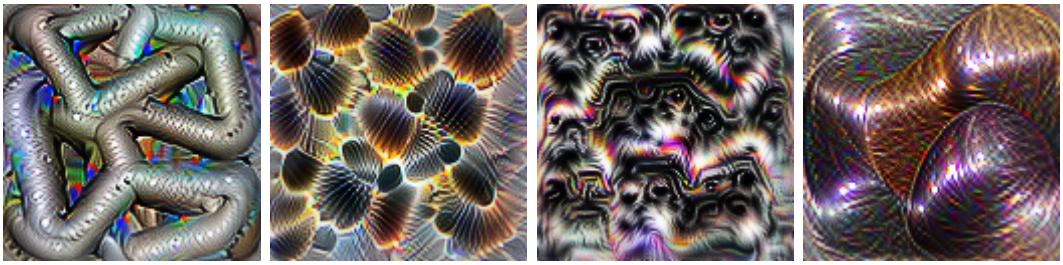

Figure 3.1: Examples of Lucid visualizations of the InceptionV4 architecture trained on the ImageNet dataset. *Left to right:* channels 0 to 3 of the InceptionV4/InceptionV4/Mixed_6b/concat layer.

Figure 3.2: Examples of Lucid visualizations of the InceptionV4 architecture trained on the ImageNet dataset. *Left to right:* channels 0 to 3 of the InceptionV4/InceptionV4/Mixed_7d/concat layer, the last layer before prediction.

Lucid also provides tools called advanced objectives and transformations. Using transformations, the visualization can be made more robust and overcome different effects of batch normalization or similar layers to display more understandable output. Objectives are used to look for different kinds of results, such as negative activations, which symbolize what the neuron is not looking for, or activation grids, showing how the network perceives spatial locations in the image.

Many more visualizations generated by the Lucid library can be seen in the OpenAI Microscope[4] collection.

3.2 Loss Change Allocation

Loss Change Allocation (LCA) is an interpretability method introduced by Janice Lan, Rosanne Liu, Hattie Zhou and Jason Yosinski in [33]. The proposed method provides a rich window into the neural network training process by measuring how much each neuron "learns" at any iteration [32]. Each neuron has assigned a score describing how much it helped or hurt (decreased or increased the loss, respectively) the network in given iteration, the method can be summed across multiple neurons or layers, allowing convenient visualization and fine-grained insight of acquired data.

The publication [33] demonstrates the LCA method on some common datasets and DNN architectures - FC and LeNet on MNIST[5], and AllCNN and ResNet on CIFAR-10[6], for description of architectures see Section 1.3.2.

[4] `https://microscope.openai.com/models`

[5] Dataset of handwritten digits

[6] Dataset containing images of 10 different classes

3.2.1 Definition of the Method

The introduced method defines the Loss Change Allocation approach by deriving from a general formula for calculating a path integral along the loss landscape. Consider starting parameter value θ_0, ending parameter value θ_T, the loss landscape $L(\theta)$ and any path P from θ_0 to θ_T, then the change in loss from θ_0 to θ_T can be calculated as:

$$L(\theta_T) - L(\theta_0) = \int_P \langle \nabla_\theta L(\theta), d\theta \rangle$$

where $\langle \cdot, \cdot \rangle$ denotes a dot product. The publication further discusses the approximation of this path integral using a series of first order Taylor approximations and improving the accuracy of this approximation using the fourth-order Runge-Kutta method (RK4) [44, 31] or Simpson's rule which yields the following:

$$L(\theta_T) - L(\theta_0) = \sum_{i=0}^{K-1} (\nabla_\theta L(\theta_t))^{(i)} (\theta_{t+1}^{(i)} - \theta_t^{(i)}) := \sum_{i=0}^{K-1} A_{t,i}$$

where the parameter θ contains K elements, $v^{(i)}$ denotes the i-th element of a vector v and $A_{t,i}$ an individual *Loss Change Allocation* component. These K components represent LCA for a single iteration of the training process [33].

3.2.2 Revealed Insights

The LCA paper mentions three new resourceful insights into the training process of neural networks - barely over half of parameters help in any given iteration, some entire layers overall move against the training gradient and lastly, iterations of peak learning appear to be synchronized. Generally, LCA offers a great metric to measure the effectiveness of model training that could further yield even more insight into the training process of (D)NNs as more people encounter this metric and attempt to uncover some unique insight.

Learning Is Noisy

Noisy learning means that there is high variability in helping parameters and overall just slightly over half of parameters are helping the training. Figure 3.3, showing LCA for ResNet, demonstrates this result in detail. We can see that the distribution of LCA is not narrow around the mean, but rather wide to both sides. This implies that the helping and hurting parameters compete during the training process rather than move all in a single direction [33].

Some Layers Hurt Overall

The LCA score can be summed over multiple neurons, layers or even multiple layers which is useful for a more macroscopic view of the network. This made possible visualization of a pattern where some layers appear to be hurting the overall loss. More specifically, the ResNet network trained on CIFAR dataset has consistently hurting the first and the last layer. Janice Lan et al. [33] tried freezing the first layer to see if it would prevent the layer from hurting the loss. They learned that even though the first layer hurts less, other layers aren't able to help as much as without the first layer frozen. Surprisingly,

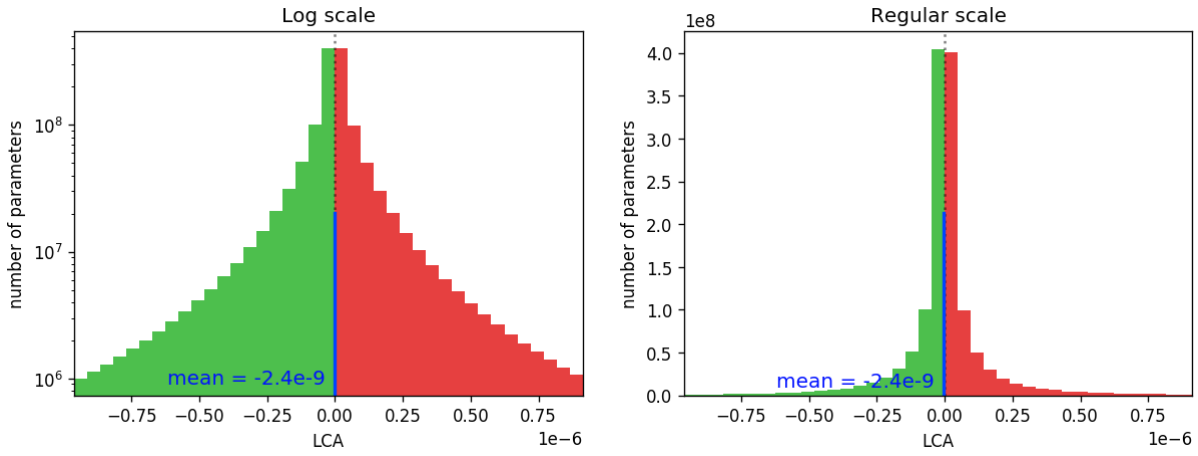

Figure 3.3: A histogram of all LCA values (all parameters at all iterations). *Left:* logarithmic scale. *Right:* regular scale. Image from [32].

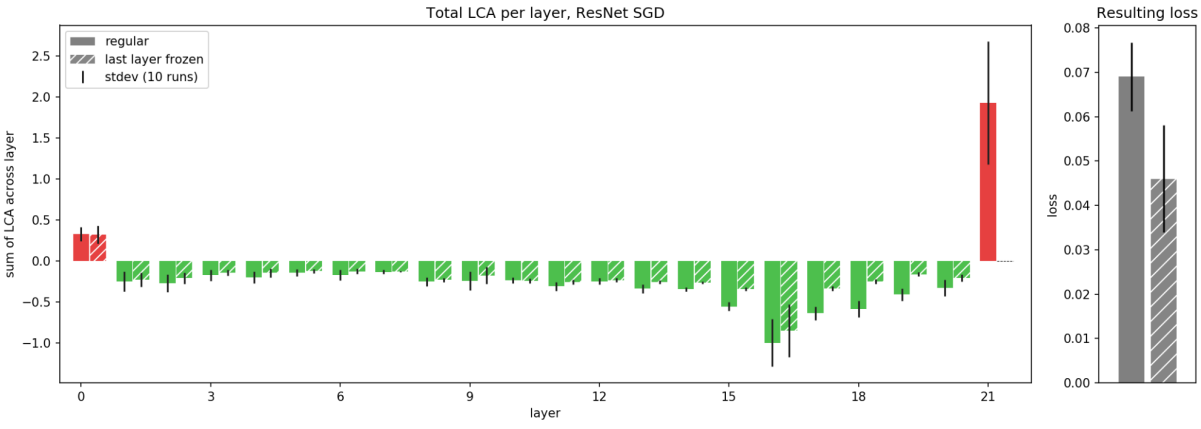

Figure 3.4: *Left:* LCA for ResNet, aggregated per layer. *Right:* overall network loss. Solid bars represent regular training, hatched bars represent training with last layer frozen at initialization. Image from [32].

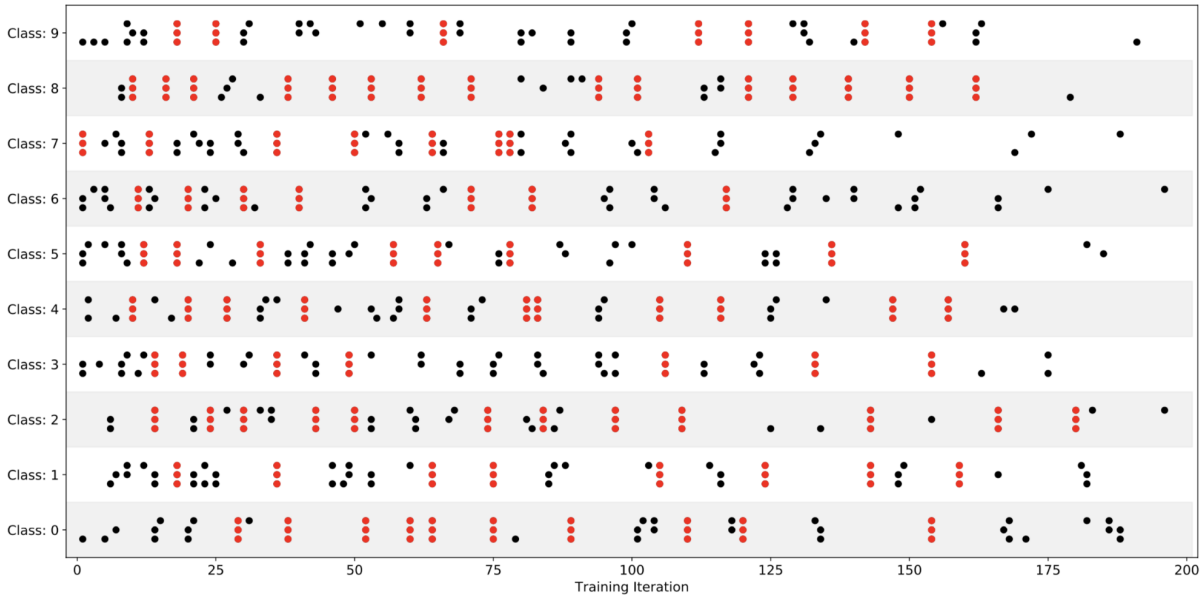

Figure 3.5: Peak moments of learning by layer and class for MNIST dataset and a three-layer FC network. Each dot represents a peak in LCA for given layer and class. Dots highlighted in red symbolize synchronized peaks. Image from [32].

freezing only the last layer results in such improvement in performance of the last layer that it easily compensates for the change in other layers, which aren't helping as much. This trend can be seen in Figure 3.4.

Learning Is Synchronized Across Layers

Janice Lan et al. [33] further define a *peak moment of learning*: "we define a *peak moment of learning*" for a layer and a class as a local minimum of LCA for that layer and class. In other words, the loss for that class decreased due to the motion of that layer on iteration t more than it decreased due to the motion on iteration t+1 or t-1". This is visualized in Figure 3.5 for easier interpretation and clearly shows that learning in many layers appears to be synchronized.

Chapter 4

Results

This work focused mainly on a recently published machine learning interpretability method, called the *Loss Change Allocation* (LCA) method. Firstly, it was experimentally evaluated on the MNIST and CIFAR10 datasets and smaller neural network architectures. Then it was attempted to evaluate the LCA method on a deeper architecture and larger dataset. Alongside these experiments, another machine learning interpretability method called activation maximization was evaluated using *Lucid*, an open-source visualization libary.

4.1 LCA on Simpler Architectures

The first goal of this work was achieving the same results as in the LCA publication [33] and summarizing the underlying concepts of machine learning.

The same dataset-architecture combinations as specified in the LCA paper were trained and evaluated. Due to time and performance constraints, each was run only once and only with SGD as optimizer. These combinations are MNIST-FC, MNIST-LeNet, CIFAR10-AllCNN, CIFAR10-ResNet, some of them are described in more detail in Section 1.3.2.

Computed results correspond to those in [33], stating that only slightly over half of parameters help reduce the overall loss in any given iteration, this can be seen in Figure 4.1 which says that out of 3600 iterations[1] a parameter has helped approximately in 1819 iterations and out of 273066 trainable parameters in the ResNet network on average 137971 helped in a given iteration. Similar results can be seen in Figures 4.2, 4.3 and 4.4.

In Figures 4.3 and 4.4 a notably large amount of parameters that neither help nor hurt in the FC and LeNet architectures can also be seen. This phenomenon is, too, commented on in the LCA paper and occurs mostly due to the many dead pixels in the MNIST dataset. [33] *Dead pixels* is a reference to those pixels that have the same value in every training image. To understand this trend further, various hyperparameter settings on the FC architecture were evaluated. Even though LeNet shows higher extremes in "not-helping" and "not-hurting" parameters, the FC architecture was chosen because of significantly lower computational requirements when compared to LeNet. Various sizes of learning rates and batch sizes were tried, since these parameters are the easiest to tweak and by their nature should have relatively high impact on the training process. Lower learning rate, same as bigger batch size, should lead to more precise and more constant

[1] used CIFAR-10 training set contains 45000 images, trained with batch size 250 that means 180 iterations per epoch are needed and the network was trained for 20 epochs

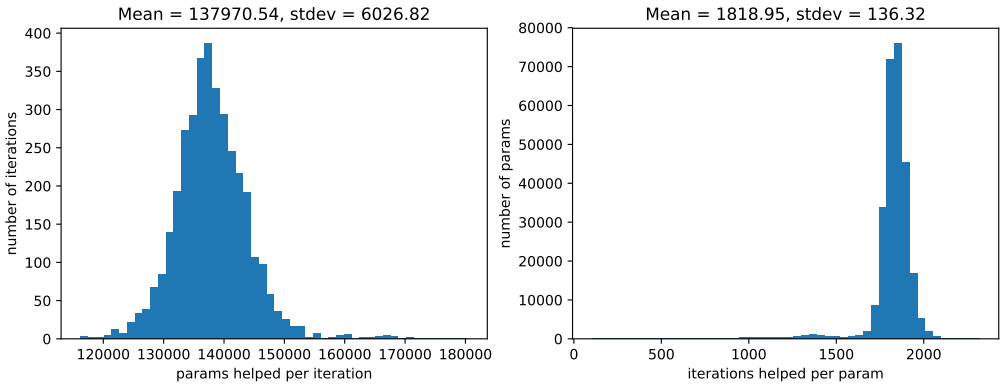

Figure 4.1: LCA results for the ResNet architecture on CIFAR10 dataset. *Left:* Count of parameters that helped each iteration (params per iteration). *Right:* Count of iterations each parameter helped (iterations per param).

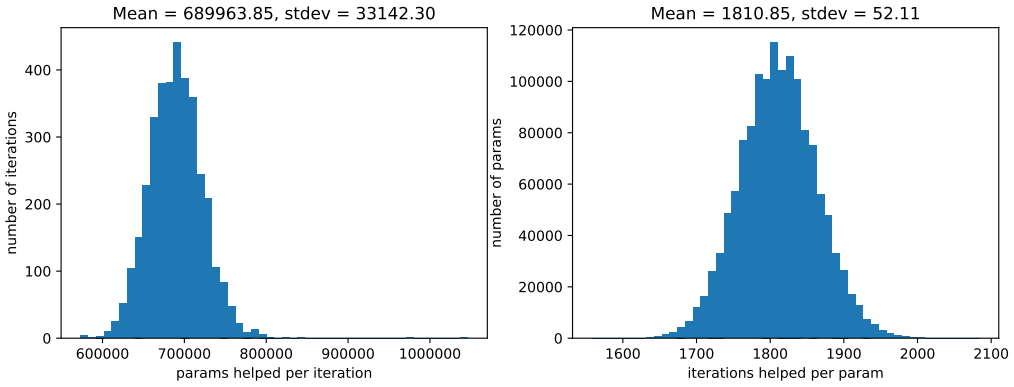

Figure 4.2: LCA results for the AllCNN architecture on CIFAR10 dataset. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

training steps, meaning more parameters should be helping, or in other words, the learning should be less noisy and overall parameters should be helping in more iterations.

The FC architecture was evaluated with learning rate values of 0.1, 0.01 and 0.001 and batch sizes of 125, 250 and 500, all trained in 15 epochs. Having three values of learning rate and three values of batch size led to nine combinations of parameters corresponding to nine experimental runs. These experiments generally confirm earlier hypothesis, that lower learning rate and bigger batch size both contribute to less noisy training. See Figures 4.5, 4.6 and 4.7 that show how the loss curve is less noisy with increasing batch size and decreasing learning rate, note the different scale on the y axis on the right showing the size of change per iteration. Comparing results of models with learning rate 0.1 and batch size 125, seen in Figure A.1, with learning rate 0.001 and batch size 250, seen in Figure A.15, clearly shows the higher amount of helping parameters for lower learning rate and bigger batch size. Visualizations of all the evaluated parameters and loss curves on the FC architecture are in Appendix A.

The downsize of learning with such parameters should be noted. Lower learning rate generally means that the training could get stuck in a local minimum, whereas having bigger batch size is prone to overfitting and models trained with large batch size tend to not generalize well. It is thus necessary to choose these hyperparameters with caution

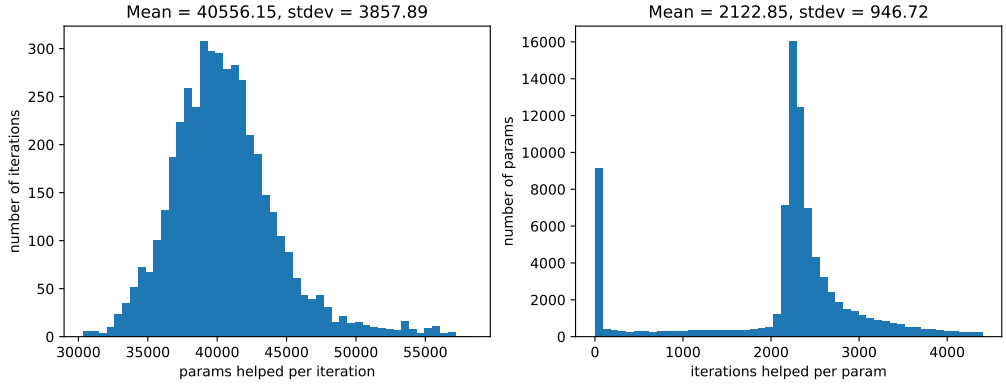

Figure 4.3: LCA results for the FC architecture on MNIST dataset. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped. The FC network was trained with 4400 iterations, the mean is skewed towards zero by the number of not helping neither hurting parameters.

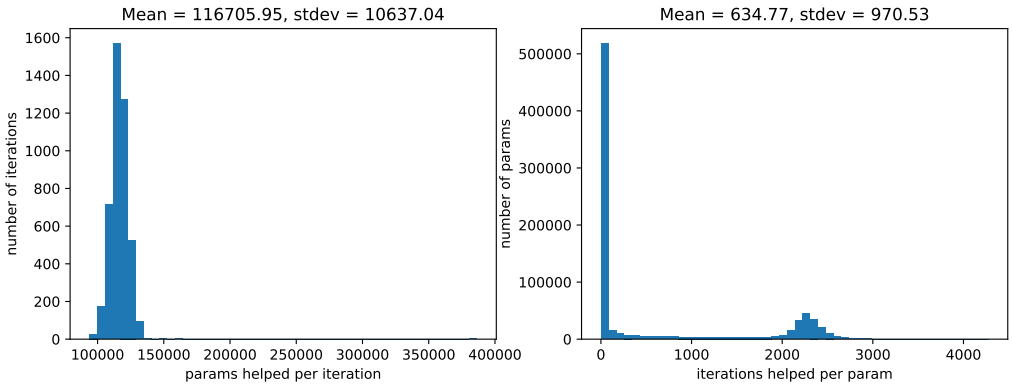

Figure 4.4: LCA results for the LeNet architecture on MNIST dataset. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped. The mean is highly skewed towards zero as in 4.3.

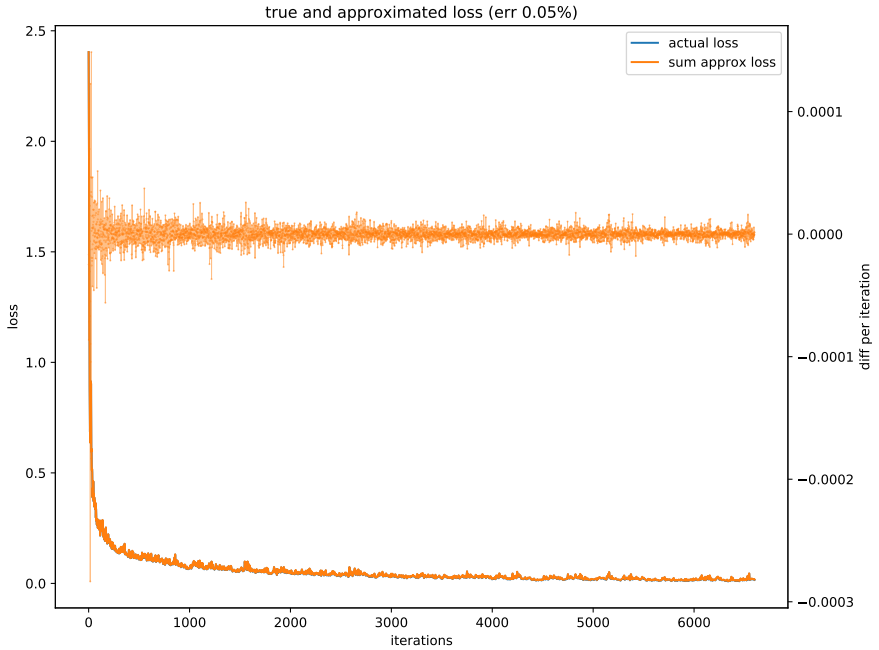

Figure 4.5: Loss of the FC architecture on MNIST dataset with learning rate 0.1 and batch size 125. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

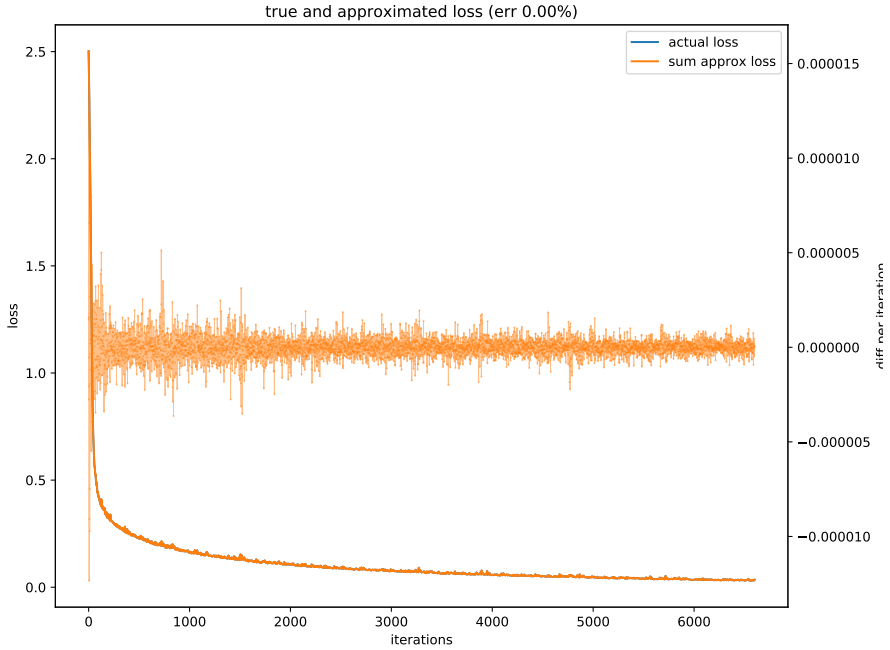

Figure 4.6: Loss of the FC architecture on MNIST dataset with learning rate 0.01 and batch size 125. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

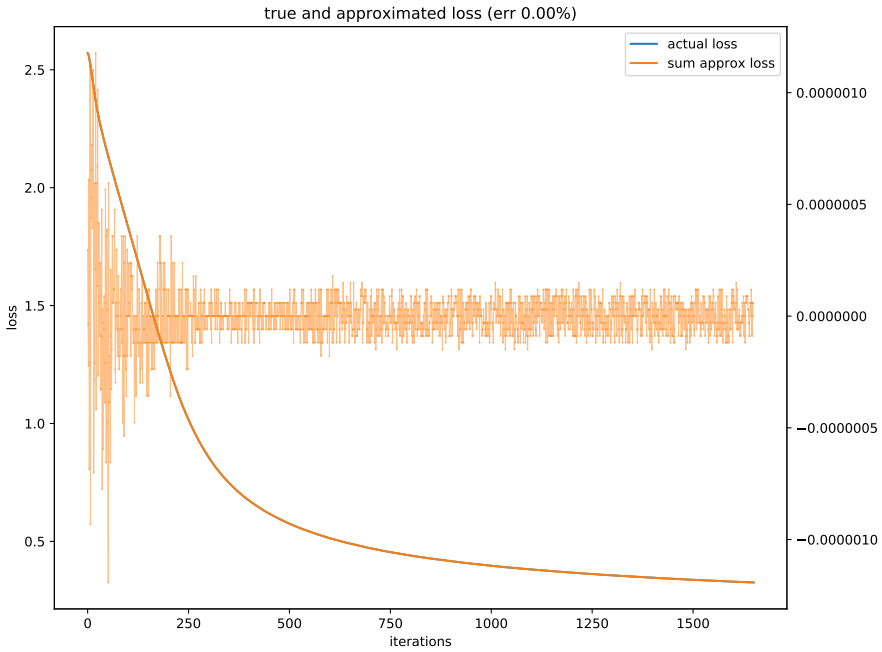

Figure 4.7: Loss of the FC architecture on MNIST dataset with learning rate 0.001 and batch size 500. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

and perform multiple experimental runs with different settings.

4.2 LCA on the SpaceNet dataset

After evaluating and discussing the results of LCA [33, 34], the same was attempted with a baseline solution of the SpaceNet Roads Network Detection challenge.

In order to obtain at least some results, a model trained for only 300 iterations overall was evaluated, the issues behind this are discussed in Sections 4.4 and 4.5. Results of the LCA method on this model can be seen in Figure 4.8. It should be noted that the number of trainable parameters in this model overall is 21 485 089, meaning that less than half of parameters are helping at a given iteration on average. From the graphs we can see that the training is very noisy, with the lowest number of parameters helping in an iteration being under 5 million and the highest being over 12 million. This could be expected as a small batch size was used, confirming earlier findings of experiments on smaller datasets. The trend of noisy learning can further be seen in Figure 4.9, which shows the loss curve during the course of training. Many parameters did not help at all or only sporadically, this is probably caused by the complex nature of the dataset and might improve if the network was trained for longer. Unfortunately, application of the LCA method on a more precisely trained model was unsuccessful due to problems with available storage space and memory.

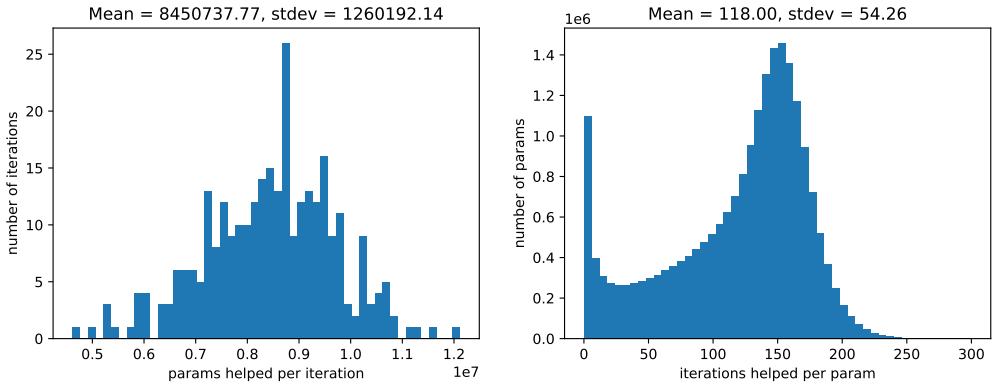

Figure 4.8: LCA results for the LinkNet architecture on SpaceNet Paris dataset. *Left:* Count of parameters that helped each iteration (params per iteration). *Right:* Count of iterations each parameter helped (iterations per param).

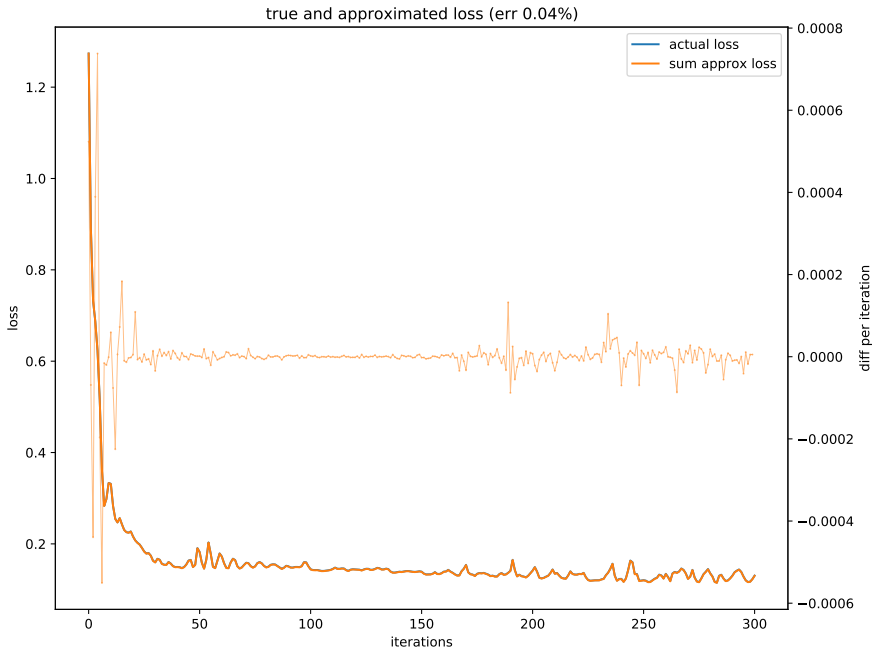

Figure 4.9: Loss of the LinkNet architecutre on SpaceNet Paris dataset. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

4.3 Lucid

Aside from the LCA method, the Lucid library was used to generate activation maps of the baseline solution. At first a version of Lucid available from the pip[2] tool was used. The API seemed very complicated as the TensorFlow graph has to be manually saved, frozen and imported, which resulted in many errors about incompatible shapes or non-existent layers in the graph. After a few attempts, cloned version of Lucid directly from GitHub was tried, which turned out to be easier to use as there was a newer version available which had implemented a simple Model.save() call with a few parameters defining the model. This saved model file is then loaded with Lucid without any need to construct the network graph manually.

After saving the model definition for Lucid, at first, errors regarding incompatible layer shapes were still encountered. These errors were related to the number of input channels. There were either 8 or 12 channels in the input depending on the run configuration when training the model. The errors did not occur when using simpler RGB images with three channels as input, thus, the RGB images were further for this experiment.

The resulting visualizations definitely do not show as much detail, nor look as appealing, as the sample images from ImageNet and CIFAR10 datasets. This is can be caused by the following reasons:

- the network could not be trained for long enough to learn precise patterns,
- in some locations, the roads are obscured by trees or other objects and it is not possible to clearly see them,
- roads themselves are not that much distinctive to their surroundings, therefore there would not be much to be seen in the visualizations, possibly only some oriented edges in places where certain directions of roads are more common than others,

examples of such images with their corresponding masks are shown in Figure 4.10 Taking these points into consideration, some indications of roads or even shapes of blocks of buildings in the later layers of the network can still be seen. Examples of the generated visualizations can be seen in Figure 4.11. Moreover, the results from evaluation of this method support the claim that neural network first learn simpler textures and shapes and later on more difficult patterns, this is shown in Figure 4.12. While some of the visualizations definitely provide an insight into the network, many of them also look just like almost random noise, like the ones shown in Figure 4.13.

4.4 Experiments with training

For evaluating the LCA on the SpaceNet dataset, a solution on the 4th place by author whose nickname is selim_sef was chosen as a baseline solution. It is one of the only two solutions using TensorFlow and Keras libraries, which makes it significantly more similar to LCA's code than the other solutions which use PyTorch. The codebase of the 4th solution also seems to be more understandable and readable when compared to the second solution using TensorFlow and Keras. Firstly, the implemented LinkNet-Inception architecture was used, later, a simpler LinkNet architecture with ResNet encoder due to fewer parameters in the network was chosen.

[2] `https://pypi.org/project/pip/`

Figure 4.10: Examples of roads difficult to detect. *Left:* image from Las Vegas showing road heavily obscured by trees. *Middle:* image from Khartoum showing various types of roads and their contrast to surroundings. *Right:* image from Khartoum showing road network among buildings covered by sand or dust that is extremely difficult to detect even by human eye.

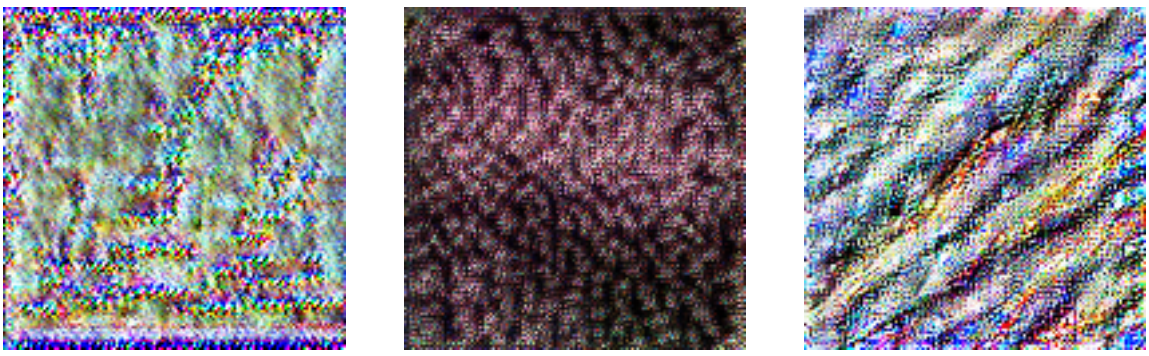

Figure 4.11: Sample visualizations obtained using the Lucid library. *Left:* activation for layer "decoder1/c2_relu/Relu:0" of the LinkNet model trained on SpaceNet images from Khartoum. *Middle:* activation for layer "fc_2_Relu/Relu:0", which is the last layer, of the LinkNet-Inception model trained on images from Las Vegas. *Right:* activation for layer "activation_32/Relu:0" of the LinkNet-Inception model trained on images from Las Vegas

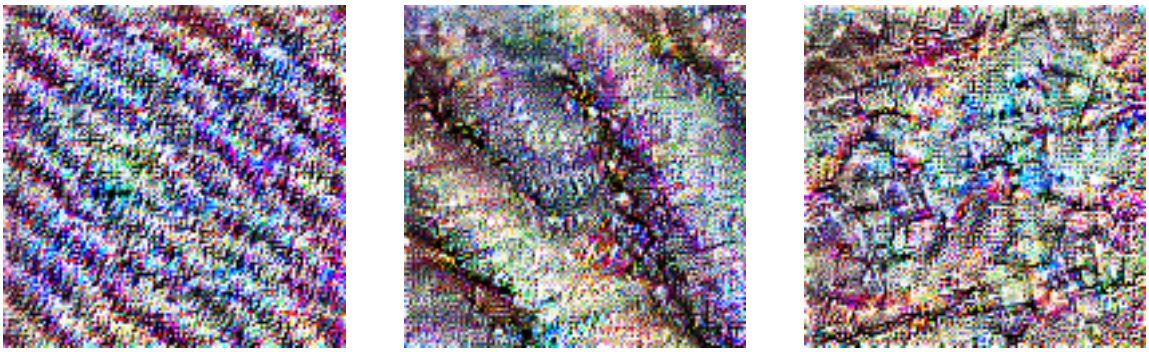

Figure 4.12: Increasing complexity of uncovered features in the Linknet-Inception network trained on images from Las Vegas. *Left:* layer "activation_9/Relu:0". *Middle:* layer "activation_29/Relu:0". *Right:* layer "activation_50/Relu:0".

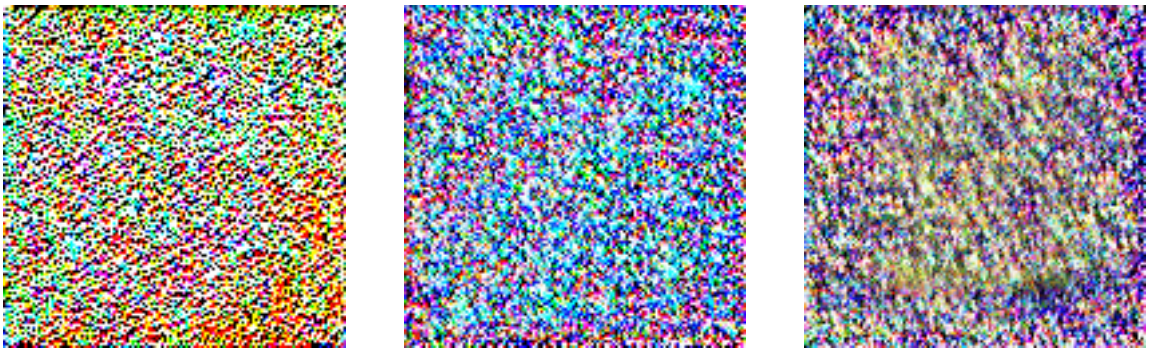

Figure 4.13: Examples of visualizations that look like noise generated for the LinkNet architecture trained on images from Khartoum. *Left:* layer "encoder1/residualBlock0_conv2/Conv2D:0". *Middle:* layer "encoder2/residualBlock1/cvbnrelu_relu/Relu:0". *Right:* layer "decoder2/c1_relu/Relu:0".

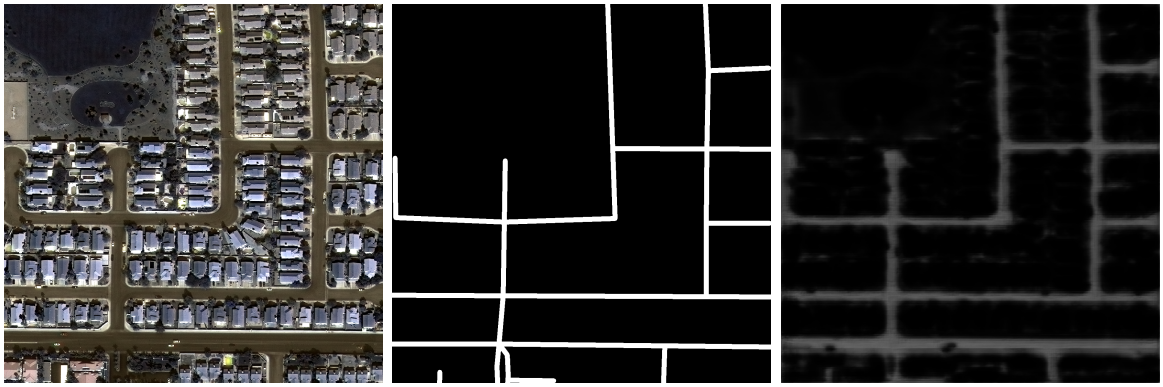

Figure 4.14: Sample prediction on img490 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask.

The training of this architecture was fairly quick and it was implemented using Python's argparse which simplified the usage of different possible parameters for the training script. Many different parameters can be tweaked, from learning rate, optimizer, number of epochs, batch size or loss function, through the size of the image crop on which the network works or number of iterations per epoch, to setting a decaying learning rate or using different preprocessing for the input images.

The codebase was well written and understandable for simple usage, it was possible to edit the data generator to read images of only one location without any major setbacks. This had to be done to reduce the performance bottleneck and to prevent lack of available storage space since images from the Las Vegas location only take up 40 GB of space.

Before attempting to apply LCA directly to LinkNet on SpaceNet data, the LinkNet model was trained on the MNIST and CIFAR datasets, at first with fewer layers for faster and simpler testing, then the whole network. On both of these datasets the network overfit very quickly, as expected.

Next goal was to apply the LCA method to the chosen baseline solution. This was the step that required the most work with source code files, as it was necessary to implement the LinkNet architecture atop of the LCA codebase. At first, usage of the training generator from the baseline solution was implemented atop of the LCA code. This resulted in as successful run of the training using SpaceNet data on a simple ResNet model for validation of the data generator.

Then, small incremental changes were made to an existing ResNet architecture while verifying proper functionality of the source code and only then a custom LinkNet model class was created. The LCA codebase did not provide any type of prediction whatsoever so a script for predicting on the trained model had to be implemented. This script was verified on the earlier mentioned MNIST and CIFAR10 datasets and simpler architectures. After solving various issues mentioned in Section 4.5, it was finally managed to train a model which yielded first meaningful predictions. While some predictions looked reasonably well, example shown in Figure 4.14, other seemed to struggle with segmentation of parking lots, identifying rooftops or similar objects as roads, example of such prediction is in Figure 4.15. More examples of predictions are in Appendix B.

Since the APLS metric was evaluated on the side of SpaceNet during the challenge, it is not implemented in the baseline solution. To get a sense of how well or badly the network performs, the Intersection over Union (IoU) metric, also known as the Jaccard index, was evaluated. IoU measures the area of overlap of the ground truth and the

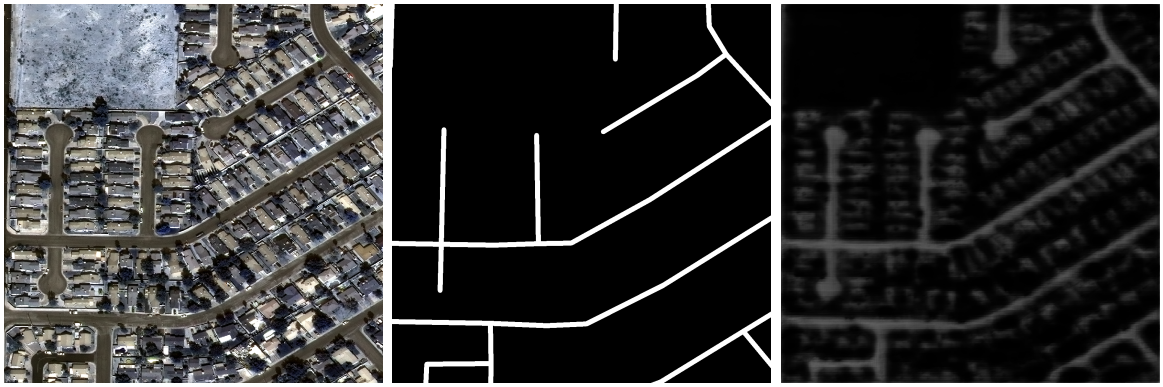

Figure 4.15: Sample prediction on img1036 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask showing incorrect classification of the buildings' rooftops.

predicted segmentation divided by the area of union of the ground truth and the predicted segmentation. The metric ranges from 0, signifying no overlap, to 1, signifying perfect segmentation. For two sets, A and B, the IoU is computed as [41]:

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

The mentioned LinkNet model resulted in IoU score of 0.333 when using classification threshold 0.2.

4.5 Implementation Issues

While trying to reach the same results as the LCA publication [33], various errors regarding installation of the tensorflow-gpu[3] python package had to be solved. These errors were related to installation of correct CUDA[4] toolkit, CuDNN[5] library, and making sure python executable was able to locate necessary files.

While attempting to evaluate the LCA method on a baseline solution of the SpaceNet challenge, various issues were encountered. The baseline solution uses Keras API which works mainly with the keras.layers.Model class and its interface, while the LCA method is implemented using an older TensorFlow version without Keras so it works with TensorFlow's graph definition and session. These different libraries and versions posed many challenges regarding implementation details and compatibility.

After implementing the LinkNet model atop of the LCA codebase, it was not possible to verify the model since the prediction always resulted in the whole image being labeled as "not road" except for a few pixels. The model was verified on the CIFAR10 and MNIST datasets, which worked well. Here the longest setback was encountered, not knowing what caused the prediction to be so off. Finally, different normalization intervals of the input images were noticed. The images from CIFAR10 and MNIST datasets were normalized to values between 0 and 1 whereas the SpaceNet generator returns images in range from -1 to 1. After editing the generator to return values between 0 and 1, the model resulted in first meaningful predictions.

[3] `https://www.tensorflow.org/guide/gpu`
[4] `https://developer.nvidia.com/cuda-zone`
[5] `https://developer.nvidia.com/cudnn`

This model was trained without batch normalization layers for faster training and easier evaluation, unfortunately, after using the batch normalization layers again, the trained model predicted almost the whole to be road. Different values of learning rate combined with different batch sizes were tried in an attempt to see what would have effect on the result. Changing values of said hyperparameters didn't affect the prediction much, only when notably fewer batch normalization layers were used, the prediction was still a bit blurred but resulted in a meaningful output. The reason behind this issue was not found.

While trying different values of hyperparameters to find the reason behind mentioned issue with batch normalization, the LCA method was applied on a model without batch normalization layers. The computations necessary for the final LCA results can be split into three steps:

1. Train the model,
2. Compute the gradients of the weights in respect to the loss,
3. Compute LCA

All of these computations demand lot of performance and getting results for only one model took over six days. Problems with insufficient RAM which couldn't fit the resulting values into memory were also encountered. The files containing resulting values together take up over 400 GB of space and even though not all of them are necessary at once, the required RAM was still in order of tens of gigabytes.

As all the computations in deep learning are computationally exhaustive, the available performance for this work was almost always a limitation. From the evaluation of LCA on the simple dataset, through training of the baseline solution to generating visualizations using the Lucid library, various errors regarding insufficient memory and storage space were encountered and overall the computations required for the purpose of this work ran only very slowly for long periods of time.

4.6 Discussion

While working on this thesis, the author was presented by large amounts of new information, most notably the basics behind machine learning and related fields. For the first time the author also worked with Google Cloud, Google Colaboratory and Amazon Web Services, even if only with smart parts of these products, it is definitely valuable experience. Managing multiple platforms at once was, too, an integral part of the work, as it was necessary to use a PC, Google Colab and a private server all at once in order to produce required results in sufficient time.

This work shows that the LCA metric provides good insight into the training process of neural network. Using this interpretability method, it is possible to evaluate multiple values of hyperparameters and compare various pros and cons of learning with different values. The downsize of this interpretability method is that it requires a lot of computation and thus either needs a lot of time or performance to reach some results. With the implementation provided alongside the LCA paper it is also necessary to execute three different scripts in order to have the final data from which it is possible to generate various graphs and visualizations.

4.7 Future Work

It could be attempted to try more training runs with different hyperparameter values of the LinkNet model and evaluate these models using the LCA method to obtain more insights into the training of this architecture. Moreover, the LCA method could be applied to different models from the baseline solution or even different solutions of the SpaceNet Road Network Detection challenge. This would result in an overview of efficiency of submitted solutions.

A new network architecture could be proposed to solve the challenge, taking into consideration the results of the challenge and the interpretability methods discussed in this work, a better performing architecture might be invented.

Moreover, the Lucid library could be used including its more advanced functions such as different objectives to take a look deeper into the model. Lucid's transformations could be implemented to make the resulting visualizations less prone to noise from different effects of batch normalization layers.

Chapter 5

Conclusion

The aim of this work was to evaluate various interpretability methods on a deep neural network. In order to accomplish this, numerous state-of-the-art publications and research papers were studied and summarized. Based on the acquired knowledge, two selected interpretability methods, the Loss Change Allocation (LCA) and activation maximization, were evaluated. The evaluation was performed on both simple datasets and the SpaceNet dataset, as intended. The LCA method was thoroughly evaluated on various smaller networks, most notably a network consisting of three fully-connected layers. The initial findings were confirmed in an experimental evaluation on a deep network on the SpaceNet dataset. Activation maximization was evaluated using an open-source library in order to present more visually pleasing results and in an attempt to uncover hidden features of the network.

The initial task of designing and evaluating deep neural network as a solution to the SpaceNet Road Network Detection challenge was fulfilled only partially since a baseline solution, only with tweaked parameters, was used instead of a custom architecture in order to focus on the evaluation of selected interpretability methods. Due to the lack of available resources the training of the network was run for only a short time. Since the training was not run for many epochs, a lower performance than that of presented solutions of the SpaceNet challenge is expected and thus the results of the network were not compared to those of the original competition.

This work has demonstrated the viability of a new interpretability method for neural networks, the Loss Change Allocation (LCA) method, and shown that it can provide valuable insights into the training process of both shallow and deep neural networks. Different impacts of hyperparameters, especially learning rate and batch size, on the training process were discussed and interpreted using outputs of the aforementioned LCA method.

The functionality of a baseline solution of the SpaceNet Road Network Detection Challenge has been verified prior to applying the LCA method to this model. Furthermore, the Lucid library was used in an attempt to interpret said model and uncover its learned features using interpretability method called activation maximization.

Lastly, this work contains introductory chapters discussing the basics of neural networks and the field of interpretability which should be comprehensible even to those with no prior knowledge in these exact fields.

References

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

[2] Maximilian Alber. *A toolbox to iNNvestigate neural networks' predictions!* https://github.com/albermax/innvestigate. [Online; accessed 12-December-2019]. 2019.

[3] Andrea Apicella, Francesco Isgrò, and Roberto Prevete. "A simple and efficient architecture for trainable activation functions". In: *Neurocomputing* 370 (2019), pp. 1 –15. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.08.065. URL: http://www.sciencedirect.com/science/article/pii/S0925231219312032.

[4] Maryam Babaee, Zimu Li, and Gerhard Rigoll. "A dual CNN–RNN for multiple people tracking". In: *Neurocomputing* 368 (2019), pp. 69 –83. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.08.008. URL: http://www.sciencedirect.com/science/article/pii/S0925231219311270.

[5] Furui Bai et al. "Densely convolutional attention network for image super-resolution". In: *Neurocomputing* 368 (2019), pp. 25 –33. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.08.070. URL: http://www.sciencedirect.com/science/article/pii/S0925231219312081.

[6] Kunlun Bai. "A Comprehensive Introduction to Different Types of Convolutions in Deep Learning". In: *Towards Data Science* (Feb. 11, 2019). [Online; accessed 17-December-2019]. URL: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215.

[7] Maria Baldeon-Calisto and Susana K. Lai-Yuen. "AdaResU-Net: Multiobjective adaptive convolutional neural network for medical image segmentation". In: *Neurocomputing* (2019). ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.01.110. URL: http://www.sciencedirect.com/science/article/pii/S0925231219304679.

[8] Dan Becker. *Partial Plots*. https://www.kaggle.com/dansbecker/partial-plots. [Online; accessed 19-May-2020].

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[10] Gabriel J. Brostow et al. "Segmentation and Recognition Using Structure from Motion Point Clouds". In: *Computer Vision – ECCV 2008* (2008). Ed. by David Forsyth, Philip Torr, and Andrew Zisserman, pp. 44–57.

[11] Carrie Cai, Jonas Jongejan, and Jess Holbrook. "The effects of example-based explanations in a machine learning interface". In: (Mar. 2019), pp. 258–262. DOI: 10.1145/3301275.3302289.

[12] Manish Chablani. "DenseNet". In: *Towards Data Science* (Aug. 25, 2017). [Online; accessed 19-March-2020]. URL: https://towardsdatascience.com/densenet-2810936aeebb.

[13] Abhishek Chaurasia and Eugenio Culurciello. "LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation". In: *CoRR* abs/1707.03718 (2017). URL: http://arxiv.org/abs/1707.03718.

[14] Hao Chen et al. "Brain tumor segmentation with deep convolutional symmetric neural network". In: *Neurocomputing* (2019). ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.01.111. URL: http://www.sciencedirect.com/science/article/pii/S0925231219304692.

[15] Yunpeng Chen et al. "Dual Path Networks". In: *CoRR* abs/1707.01629 (2017). URL: http://arxiv.org/abs/1707.01629.

[16] Guangliang Cheng et al. "Accurate urban road centerline extraction from VHR imagery via multiscale segmentation and tensor voting". In: *Neurocomputing* 205 (2016), pp. 407 –420. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.04.026. URL: http://www.sciencedirect.com/science/article/pii/S0925231216302855.

[17] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *CoRR* abs/1604.01685 (2016). URL: http://arxiv.org/abs/1604.01685.

[18] Adam Van Etten. "SpaceNet Road Detection and Routing Challenge - Part I". In: *The DownLinQ* (Oct. 3, 2017). https://medium.com/the-downlinq/spacenet-road-detection-and-routing-challenge-part-i-d4f59d55bfce.

[19] P.H.N. Gill. *Introduction to Machine Learning Interpretability*. O'Reilly Media, Incorporated, 2018. ISBN: 9781492033158. URL: https://books.google.cz/books?id=4CyVtgEACAAJ.

[20] Tao Gong et al. "Using multi-label classification to improve object detection". In: *Neurocomputing* 370 (2019), pp. 174 –185. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.08.089. URL: http://www.sciencedirect.com/science/article/pii/S0925231219312585.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262337373. URL: https://books.google.cz/books?id=omivDQAAQBAJ.

[22] Patrick Hall. *A curated list of awesome machine learning interpretability resources*. https://github.com/jphall663/awesome-machine-learning-interpretability. [Online; accessed 10-January-2020]. 2020.

[23] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385.

[24] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993 (2016). URL: http://arxiv.org/abs/1608.06993.

[25] Built In. *What is Artificial Intelligence? How Does AI Work?* [Online; accessed 10-January-2020]. URL: https://builtin.com/artificial-intelligence.

[26] Simon Jégou et al. "The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation". In: *CoRR* abs/1611.09326 (2016). URL: http://arxiv.org/abs/1611.09326.

[27] Ming Jiang et al. "Weighted triple-sequence loss for video-based person re-identification". In: *Neurocomputing* (2019). ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.11.088. URL: http://www.sciencedirect.com/science/article/pii/S0925231219316923.

[28] Andrej Karpathy. "Yes you should understand backprop". In: *Medium* (Dec. 19, 2016). [Online; accessed 17-December-2019]. URL: https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b.

[29] Raghavendra Kotikalapudi and contributors. *keras-vis*. https://github.com/raghakot/keras-vis. [Online; accessed 05-May-2020]. 2017.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: (2012). URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[31] W. Kutta. *Beitrag zur näherungsweisen Integration totaler Differentialgleichungen*. Teubner, 1901. URL: https://books.google.cz/books?id=K5e6kQEACAAJ.

[32] Janice Lan et al. "Introducing LCA: Loss Change Allocation for Neural Network Training". In: *Uber Engineering* (Sept. 10, 2019). [Online; accessed 12-December-2019]. URL: https://eng.uber.com/loss-change-allocation/.

[33] Janice Lan et al. "LCA: Loss Change Allocation for Neural Network Training". In: *Proceedings of the NeurIPS 2019 conference* (Sept. 3, 2019). https://arxiv.org/pdf/1909.01440.pdf.

[34] Janice Lan et al. *Loss Change Allocation*. [Online; accessed 11-January-2020]. URL: https://github.com/uber-research/loss-change-allocation.

[35] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* (1998). URL: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

[36] Yisha Liu et al. "Autonomous road detection and modeling for UGVs using vision-laser data fusion". In: *Neurocomputing* 275 (2018), pp. 2752 –2761. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.11.042. URL: http://www.sciencedirect.com/science/article/pii/S0925231217318015.

[37] Jose Llamas et al. "Classification of Architectural Heritage Images Using Deep Learning Techniques". In: *Applied Sciences* 7 (Sept. 2017), p. 992.

[38] C. Molnar. *Interpretable Machine Learning*. Lulu.com, 2020. ISBN: 9780244768522. URL: https://books.google.cz/books?id=RHjTxgEACAAJ.

[39] Xuejing Niu et al. "Effective image restoration for semantic segmentation". In: *Neurocomputing* 374 (2020), pp. 100 –108. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.09.063. URL: http://www.sciencedirect.com/science/article/pii/S0925231219313311.

[40] Elaheh Rashedi et al. ""Stream loss": ConvNet learning for face verification using unlabeled videos in the wild". In: *Neurocomputing* 329 (2019), pp. 311 –319. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.10.041. URL: http://www.sciencedirect.com/science/article/pii/S092523121831230X.

[41] Seyed Hamid Rezatofighi et al. "Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression". In: *CoRR* abs/1902.09630 (2019). URL: http://arxiv.org/abs/1902.09630.

[42] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[43] Frank Rosenblatt. "The Perceptron - a perceiving and recognizing automaton". In: *Cornell Aeronautical Laboratory* (Jan. 1957). https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf.

[44] C. Runge. "Ueber die numerische Auflösung von Differentialgleichungen." In: *Mathematische Annalen* 46 (1895), pp. 167–178. URL: http://eudml.org/doc/157756.

[45] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[46] Sumit Sarin. "VGGNet vs ResNet (The Vanishing Gradient Problem)". In: *Towards Data Science* (Dec. 29, 2019). [Online; accessed 19-May-2020]. URL: https://towardsdatascience.com/vggnet-vs-resnet-924e9573ca5c.

[47] Irhum Shafkat. "Intuitively Understanding Convolutions for Deep Learning". In: *Towards Data Science* (June 1, 2018). [Online; accessed 17-December-2019]. URL: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1.

[48] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[49] Yuhu Shan, Wen Feng Lu, and Chee Meng Chew. "Pixel and feature level based domain adaptation for object detection in autonomous driving". In: *Neurocomputing* 367 (2019), pp. 31 –38. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.08.022. URL: http://www.sciencedirect.com/science/article/pii/S092523121931149X.

[50] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556.

[51] SpaceNet. *Automated Road Network Extraction and Route Travel Time Estimation from Satellite Imagery*. [Online; accessed 13-May-2020]. URL: https://spacenet.ai/sn5-challenge/.

[52] SpaceNet. *Building Extraction Round 1*. [Online; accessed 13-May-2020]. URL: https://spacenetchallenge.github.io/Challenges/Challenge-1.html.

[53] SpaceNet. *Building Foot Print Extraction: Round 2*. [Online; accessed 13-May-2020]. URL: https://spacenetchallenge.github.io/Challenges/Challenge-2.html.

[54] SpaceNet. *Off-Nadir Building Footprint Extraction*. [Online; accessed 13-May-2020]. URL: https://spacenetchallenge.github.io/Challenges/Challenge-4.html.

[55] SpaceNet. *Road Extraction and Routing*. [Online; accessed 13-May-2020]. URL: https://spacenetchallenge.github.io/Challenges/Challenge-3.html.

[56] SpaceNet. *SpaceNet 1: Building Detection v1*. [Online; accessed 13-May-2020]. URL: https://spacenet.ai/spacenet-buildings-dataset-v1/.

[57] SpaceNet. *SpaceNet 2: Building Detection v2*. [Online; accessed 13-May-2020]. URL: https://spacenet.ai/spacenet-buildings-dataset-v2/.

[58] SpaceNet. *SpaceNet 3: Road Network Detection*. [Online; accessed 13-May-2020]. URL: https://spacenet.ai/spacenet-roads-dataset/.

[59] SpaceNet. *SpaceNet 4: Off-Nadir Buildings*. [Online; accessed 13-May-2020]. URL: https://spacenet.ai/off-nadir-building-detection/.

[60] SpaceNet. *SpaceNet 5 Challenge*. [Online; accessed 13-May-2020]. URL: https://www.topcoder.com/challenges/30099956.

[61] SpaceNet. *SpaceNet Imagery*. https://spacenetchallenge.github.io/datasets/spacenetImagery-summary.html. [Online; accessed 19-May-2020].

[62] SpaceNet. *SpaceNet Roads Dataset*. https://spacenetchallenge.github.io/datasets/spacenetRoads-summary.html. [Online; accessed 10-May-2020].

[63] SpaceNet. *spacenet.ai - Accelerating Geospatial Machine Learning*. [Online; accessed 13-May-2020]. URL: https://spacenet.ai.

[64] SpaceNet. *Winning Solutions from SpaceNet Road Detection and Routing Challenge*. https://github.com/SpaceNetChallenge/RoadDetector. [Online; accessed 29-April-2020]. 2018.

[65] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842.

[66] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[67] Hui Tian et al. "Person re-identification via adaptive verification loss". In: *Neurocomputing* 359 (2019), pp. 93 –101. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.05.037. URL: http://www.sciencedirect.com/science/article/pii/S0925231219307209.

[68] R. Tomsett et al. "Why the Failure? How Adversarial Examples Can Provide Insights for Interpretable Machine Learning". In: *2018 21st International Conference on Information Fusion (FUSION)*. 2018, pp. 838–845.

[69] TopCoder. *TopCoder, SpaceNet Challenge*. [Online; accessed 13-May-2020]. URL: https://community.topcoder.com/tc?module=MatchDetails&rd=16835.

[70] A. M. Turing. "Computing Machinery and Intelligence". In: *Mind* 59.236 (1950), pp. 433–460. ISSN: 00264423, 14602113. URL: http://www.jstor.org/stable/2251299.

[71] Qi Wang, Jianwu Fang, and Yuan Yuan. "Adaptive road detection via context-aware label transfer". In: *Neurocomputing* 158 (2015), pp. 174 –183. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.01.054. URL: http://www.sciencedirect.com/science/article/pii/S0925231215001010.

[72] Yidan Wang and Liming Yang. "A robust loss function for classification with imbalanced datasets". In: *Neurocomputing* 331 (2019), pp. 40 –49. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.11.024. URL: http://www.sciencedirect.com/science/article/pii/S0925231218313523.

[73] Jiagang Zhu et al. "Convolutional relation network for skeleton-based action recognition". In: *Neurocomputing* 370 (2019), pp. 109 –117. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.08.043. URL: http://www.sciencedirect.com/science/article/pii/S0925231219311816.

List of Appendices

Appendix A

LCA results for different hyperparameters

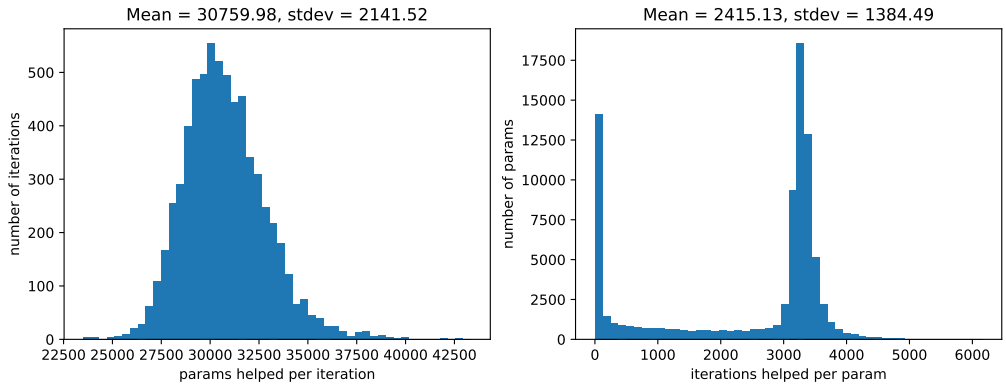

Figure A.1: LCA of MNIST-FC, learning rate = 0.1, batch size = 125, 6600 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

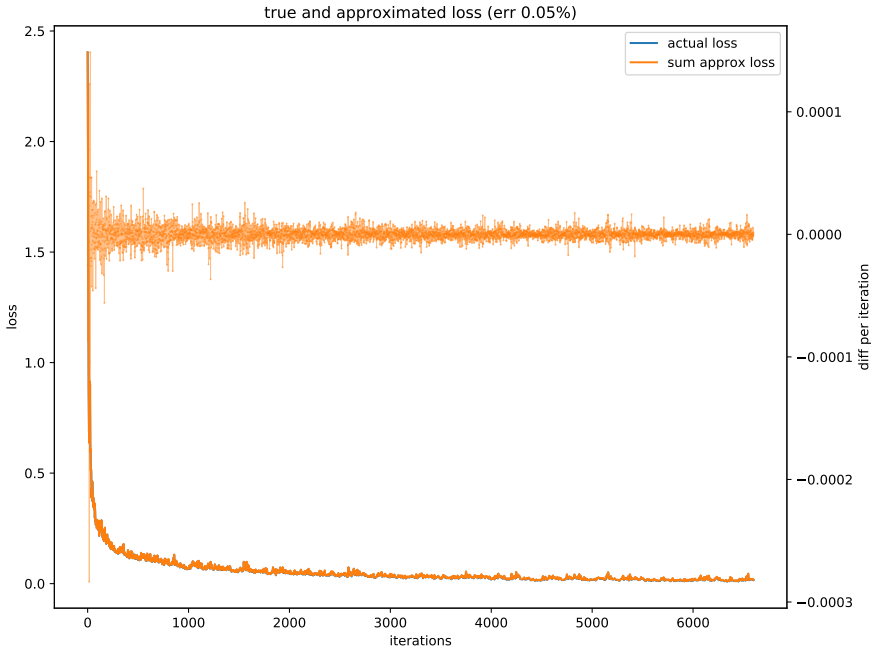

Figure A.2: Loss of MNIST-FC, learning rate = 0.1, batch size = 125, 6600 iterations. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

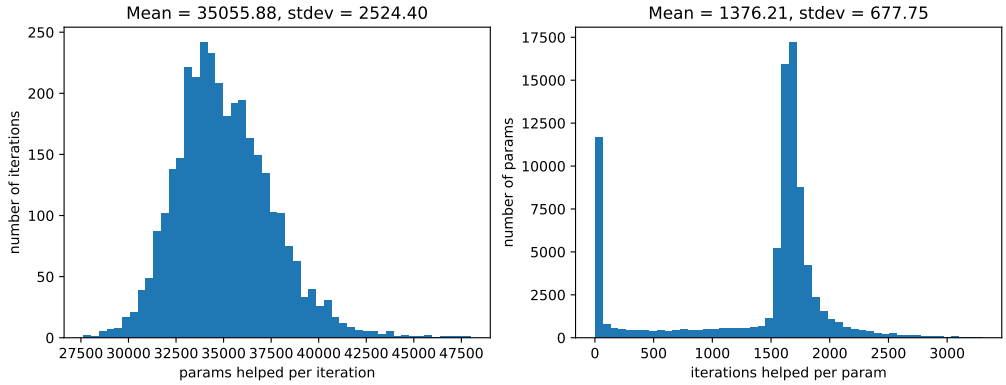

Figure A.3: LCA of MNIST-FC, learning rate = 0.1, batch size = 250, 3300 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

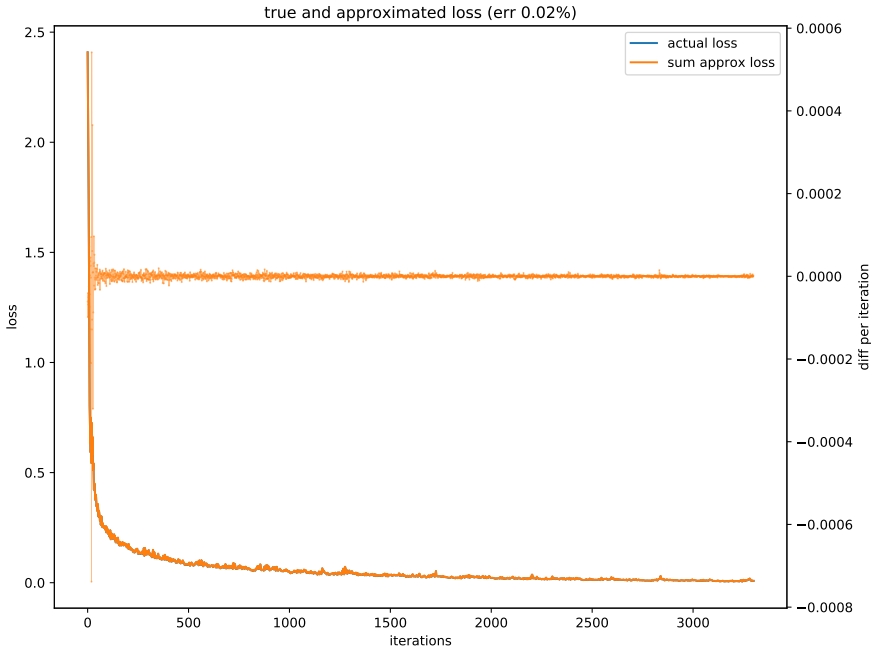

Figure A.4: Loss of MNIST-FC, learning rate = 0.1, batch size = 250, 3300 iterations. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

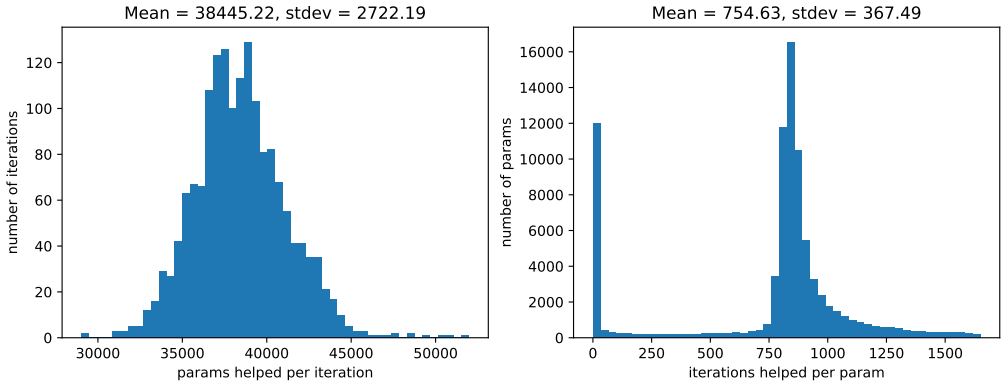

Figure A.5: LCA of MNIST-FC, learning rate = 0.1, batch size = 500, 1650 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

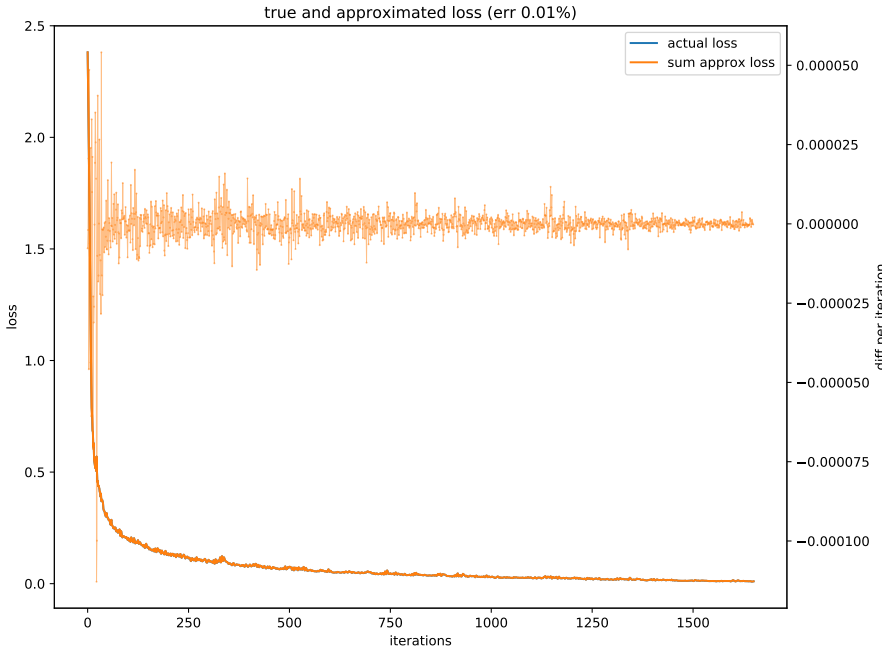

Figure A.6: Loss of MNIST-FC, learning rate = 0.1, batch size = 500, 1650 iterations. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

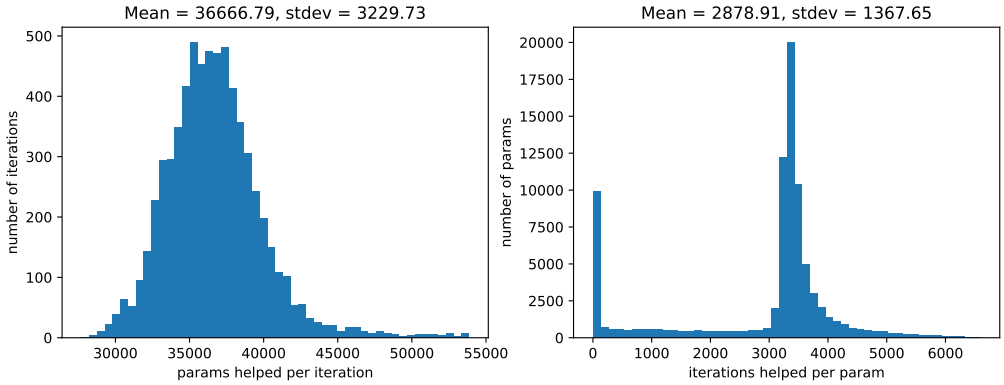

Figure A.7: LCA of MNIST-FC, learning rate = 0.01, batch size = 125, 6600 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

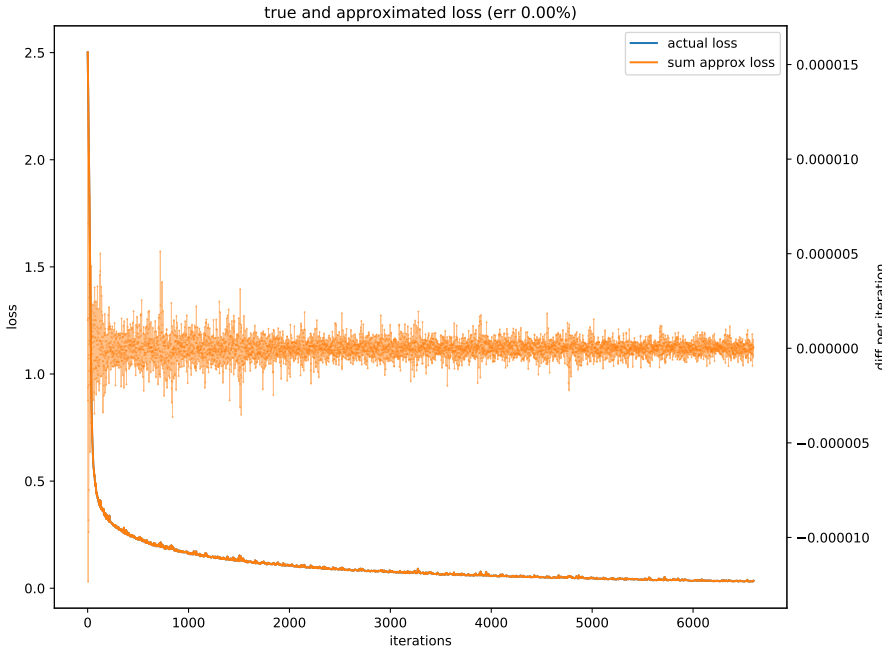

Figure A.8: Loss of MNIST-FC, learning rate = 0.01, batch size = 125, 6600 iterations Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum..

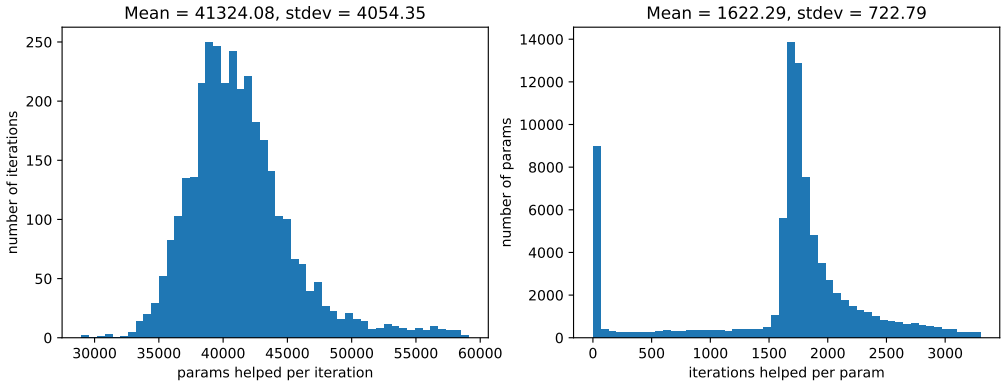

Figure A.9: LCA of MNIST-FC, learning rate = 0.01, batch size = 250, 3300 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

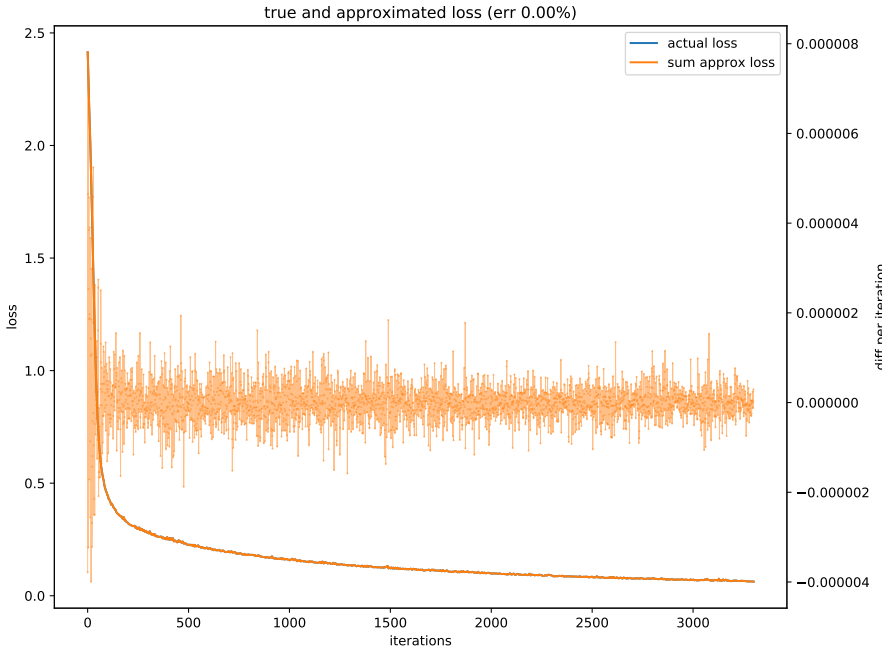

Figure A.10: Loss of MNIST-FC, learning rate = 0.01, batch size = 250, 3300 iterations. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

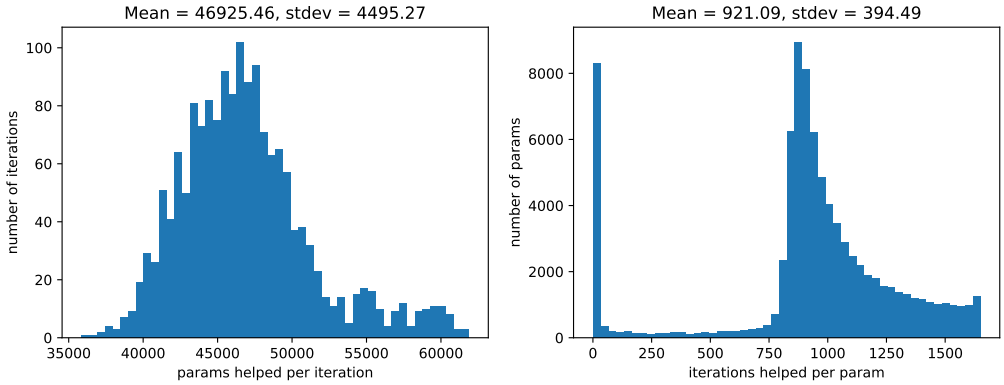

Figure A.11: LCA of MNIST-FC, learning rate = 0.01, batch size = 500, 1650 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

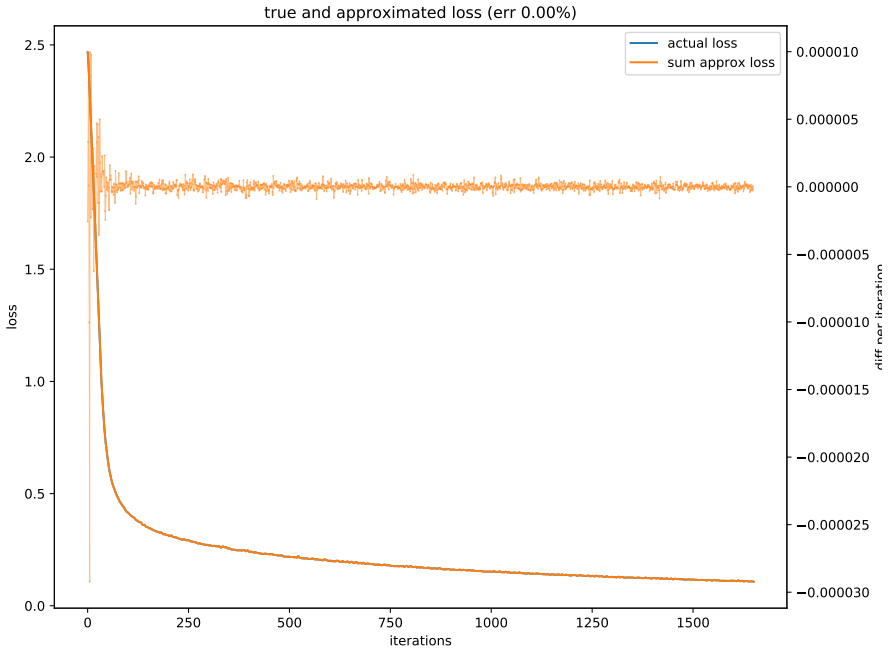

Figure A.12: Loss of MNIST-FC, learning rate = 0.01, batch size = 500, 1650 iterations. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

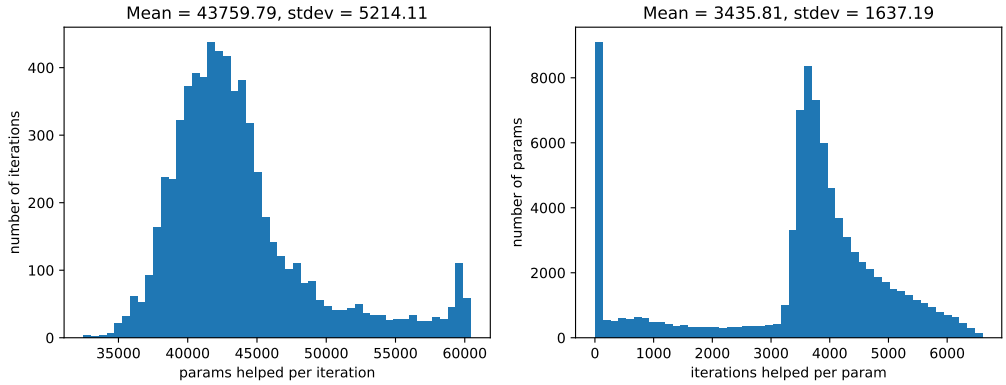

Figure A.13: LCA of MNIST-FC, learning rate = 0.001, batch size = 125, 6600 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

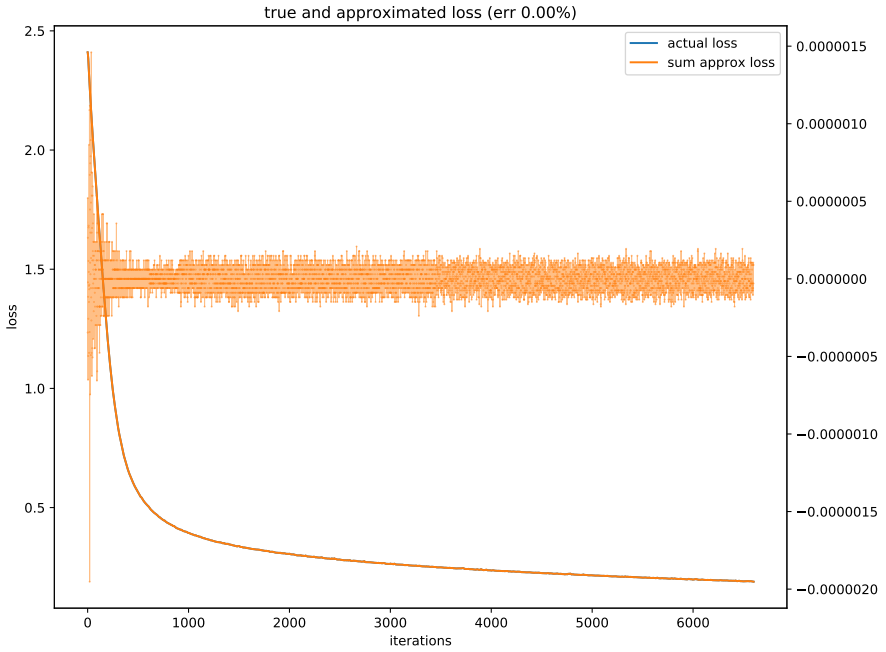

Figure A.14: Loss of MNIST-FC, learning rate = 0.001, batch size = 125, 6600 iterations. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

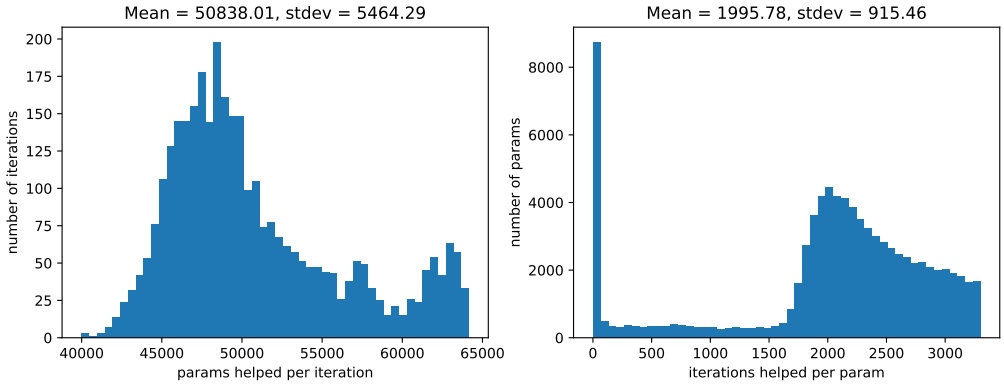

Figure A.15: LCA of MNIST-FC, learning rate = 0.001, batch size = 250, 3300 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

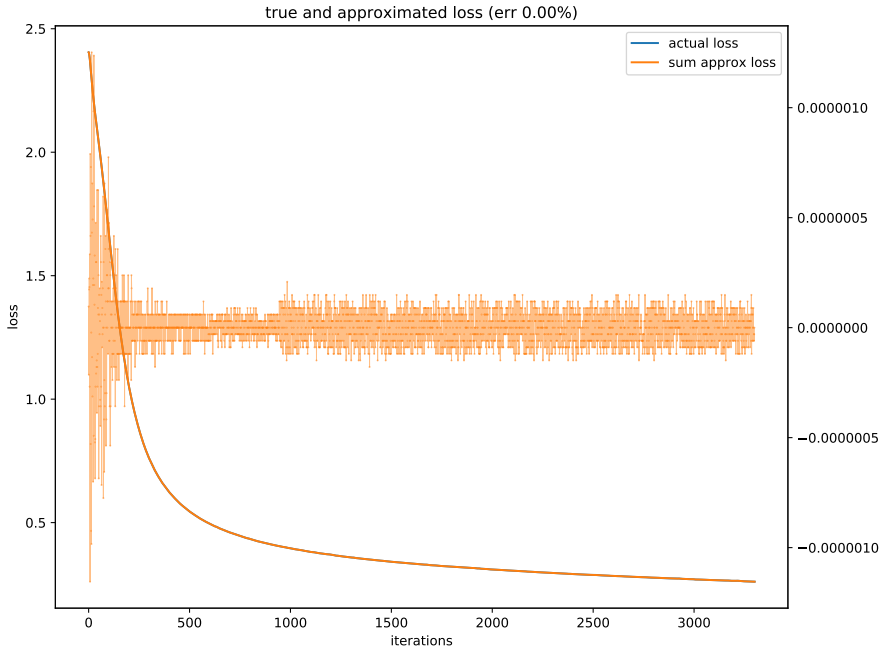

Figure A.16: Loss of MNIST-FC, learning rate = 0.001, batch size = 250, 3300 iterations. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

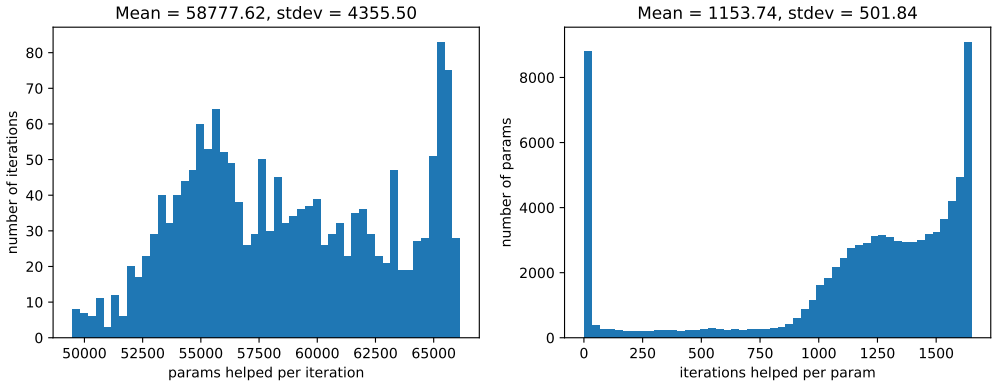

Figure A.17: LCA of MNIST-FC, learning rate = 0.001, batch size = 500, 1650 iterations. *Left:* Count of parameters that helped each iteration. *Right:* Count of iterations each parameter helped.

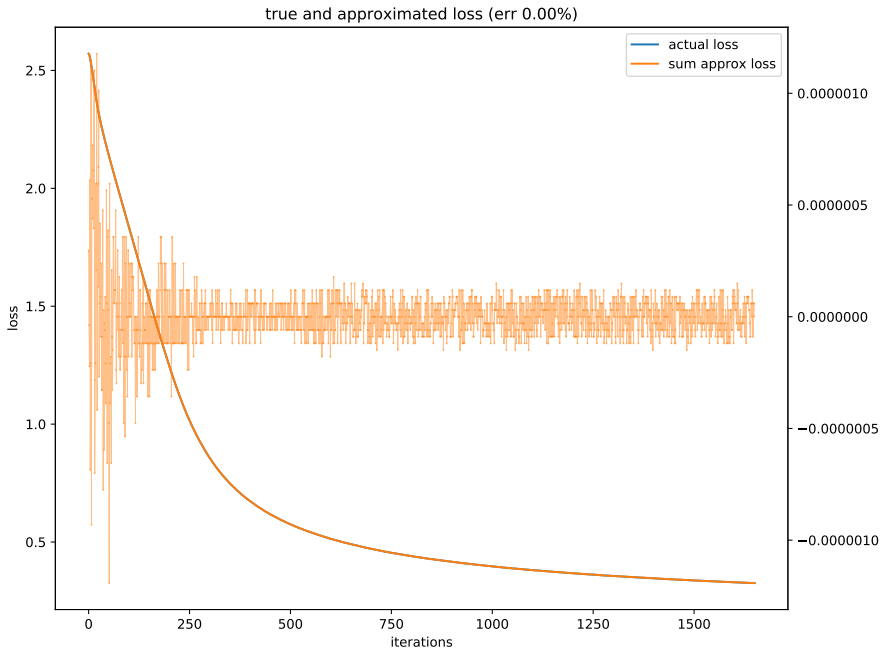

Figure A.18: Loss of MNIST-FC, learning rate = 0.001, batch size = 500, 1650 iterations. Differences in loss per iteration and sum of the differences are shown in orange. Actual loss in given iteration is in blue, but it is covered by the orange sum.

Appendix B

Examples of predictions on SpaceNet dataset

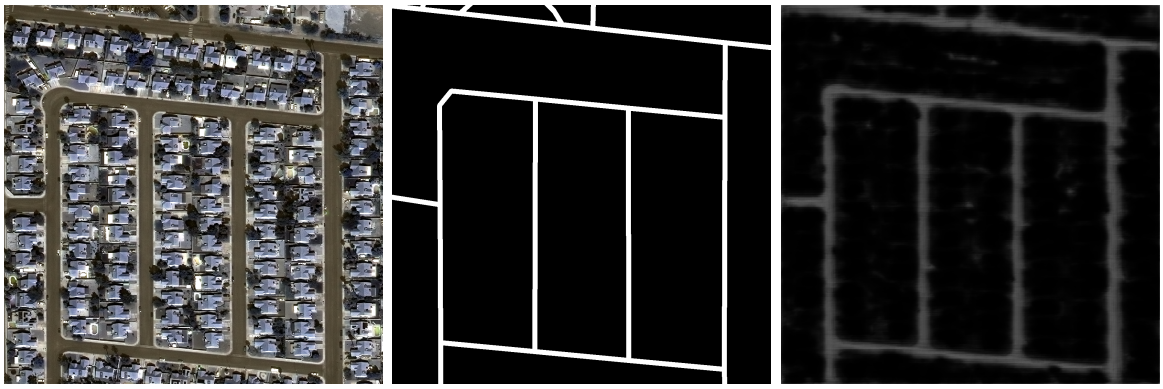

Figure B.1: Sample prediction on img731 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask.

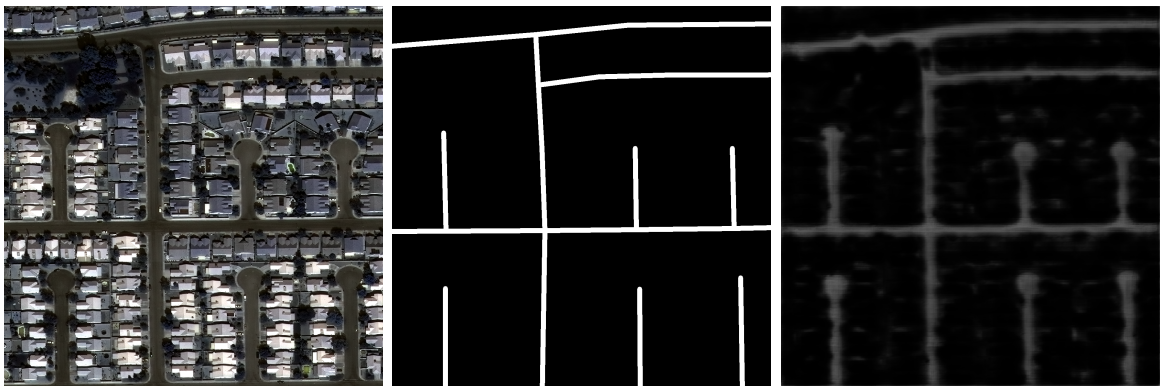

Figure B.2: Sample prediction on img1082 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask.

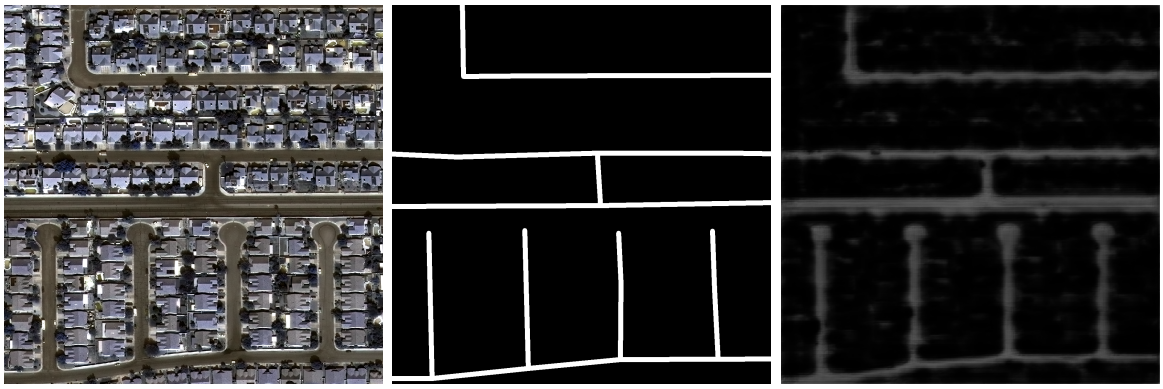

Figure B.3: Sample prediction on img1083 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask.

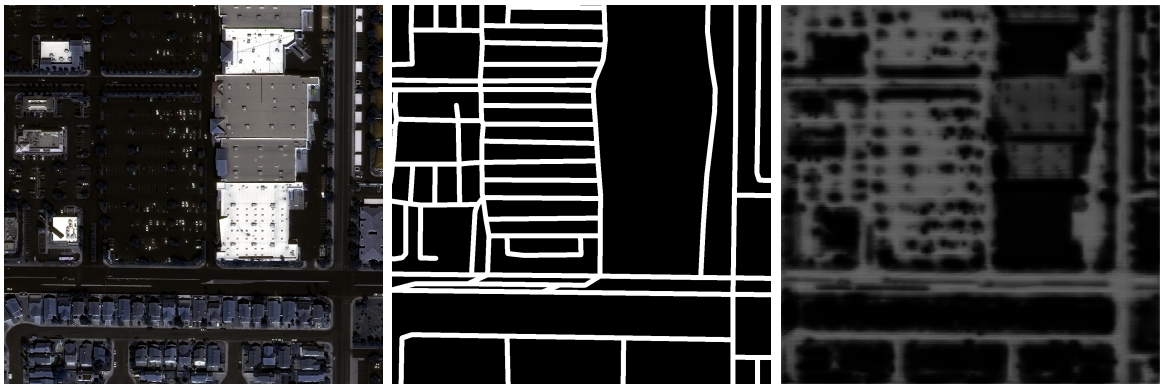

Figure B.4: Sample prediction on img635 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask showing incorrect segmentation for parking lots and a shopping mall roof.

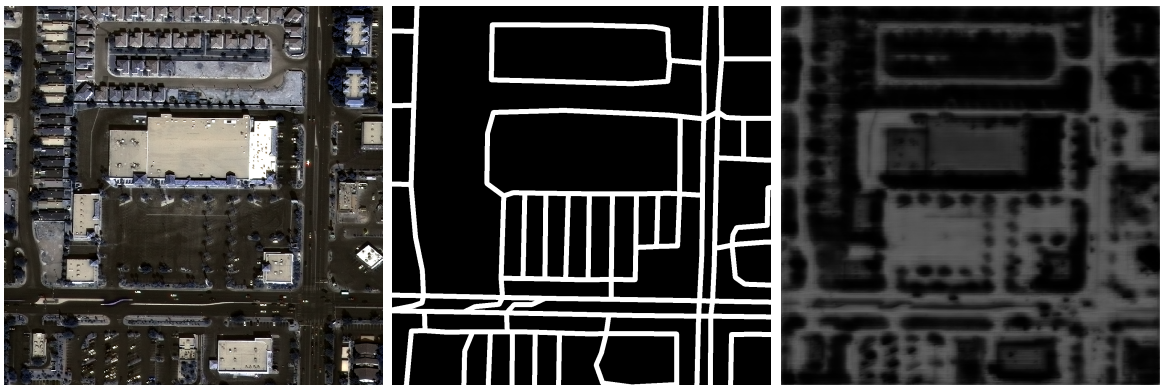

Figure B.5: Sample prediction on img795 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask showing incorrect segmentation for parking lots.

Figure B.6: Sample prediction on img601 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask showing incorrect segmentation for empty parking lots, note that emptier parking lots generally lead to worse results.

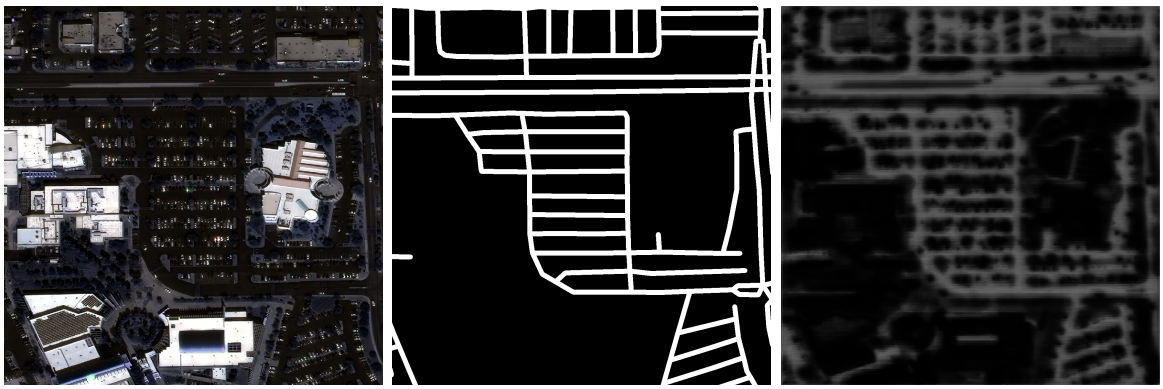

Figure B.7: Sample prediction on img704 from the Las Vegas location. *Left:* the original multispectral image. *Middle:* ground truth mask. *Right:* predicted mask showing an above average result for segmentation of parking lots, probably due to a lot of parked cars.

Figure B.8: Sample prediction on img629 from the Las Vegas location. *Left:* the original multispectral image containing roads heavily covered by trees. *Middle:* ground truth mask. *Right:* predicted mask.