Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

DIPLOMA THESIS AGREEMENT

Student: Kunc Vladimír

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: Deep neural network for satellite image classification using Open Street Maps

Guidelines:

The task is to design, implement and experimentally evaluate a deep neural network for learning classification of satellite images using labels from Open Street Maps. The resulting pipeline should include image preprocessing, creation of training, validation, and testing datasets with annotations, and thorough experimental evaluation of the proposed architecture. Determining the list of target classes will be based on suitability of available data. The implementation should be in Python; use of the followin g python packages is recommended: tensorflow, tflearn / keras, gdal. For training and evaluation publicly available datasets of high resolution satellite imagery should be used, e.g.Spacenet byDigitalGlobe. Integral part of the work is the analysis of related work. As part of the evaluation, performance comparison with respect to related work has to be included.

Bibliography/Sources:

[1] Mnih, Volodymyr, and Geoffrey E. Hinton. "Learning to label aerial images from noisy data." Proceedings of the 29thfollow International Conference on Machine Learning (ICML-12). 2012.

[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[3] Noh, Hyeonwoo, Seunghoon Hong, and Bohyung Han. "Learning deconvolution network for semantic segmentation." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[4] Abadi, Mart?n, et al. "TensorFlow: Large-scale machine learning on heterogeneous systems, 2015." Software available from tensorflow. org.

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

Valid until the end of the summer semester of academic year 2017/2018

prof. Dr. Michal Pěchouček, MSc.	prof. Ing. Pavel Ripka,CSc.
Head of Department	Dean

Prague, February 20, 2017

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Diploma Thesis

Deep neural network for satellite image classification using OpenStreetMap

Vladimír Kunc

Supervisor: Ing. Michal Reinštein, Ph.D.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

May 25, 2017

Aknowledgements

I would like to express my sincere gratitude to my thesis supervisor Ing. Michal Reinštein, Ph.D. for his support and guidance. He has provided me with the necessary feedback and resources I needed to carry out the work.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24. května 2017 ....

Podpis autora práce

Abstract

There is rising supply of remote sensing data and also rising demand for its automatic analysis, one of the branches of analysis is classification. However, despite the rising demand, there is no suitable dataset for learning neural networks. The aim of this work is to design, implement and experimentally evaluate a deep neural network for learning classification of remote sensing images using labels from *OpenStreetMap*. Thus this work introduces a novel aerial image dataset that was annotated using the *OpenStreetMap*. Using the novel dataset, the suitability of several state-of-the-art architectures for transfer learning from *ImageNet* is assessed and compared to performance of a deep network learned using the novel dataset only. Two augmentation techniques are evaluated for their appropriateness on this dataset. It was found that the ResNet50 from the tested architectures is the most accurate on the dataset and that transfer learning is less suitable than learning from scratch on this dataset. The multi-source nature of the novel datasets benefits from augmentation of the color space compared to strictly spatial augmentation.
Keywords: image classification, deep learning, OpenStreetMap, dataset creation

Abstrakt

Nabídka dat z dálkového průzkumu Země (DPZ) vzrůstá a stejně stoupá poptávka po metodách umožnííjící automatickou analýzu těchto dat, jednou z těchto metod je klasificka. Nicméně přes vzrůstající poptávku, neexistuje žádný vhodný dataset pro učení neuronových sítí na těchto datech. Cílem této práce je navrhnout, implementovat a experimentálně vyhodnotit hlubokou neuronovou sít̆ for učení se klasifikace dat z DPZ za použití anotací z OpenStreetMap. Z tohoto důvodu tato práco představuje nový dataset leteckých snímků, který byl anotován za pomoci OpenStreetMap. Pomocí tohoto nového datasetu byla vyhodnocena vhodnost a chování několika moderních architektur pomoci přenosového učení z *ImageNet.* Chování sítí naučených pomoci přenosového učení bylo porovnávno s chováním hluboké sítě naučené jen za pomoci nového datasetu. Dále byla vyhodnocena vhodnost dvou augmentačních technik pro navržený dataset. Bylo zjištěno, že ResNet50 je z testovaných architektur nejvhodnější pro tento dataset a že přenosové učení je méně výhodné než učení od nuly na navrženém datasetu. Dále bylo ukázano, že vzhledem k více zdrojové povaze dataset použítí augmentace s transformací barev je výhodnější než použití čisté prostorové augmentace.
Klíčová slova: klasifikace, hluboké učení, OpenStreetMap, vytváření datasetu

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis describes an aerial image dataset creation using the OpenStreetMap (OSM) [141] as the source of annotations and then evaluates several deep neural network architectures that were trained using the obtained data. First, a general introduction is provided in this chapter, then Ch. 2 describes related works for the classification (section 2.1) and segmentation (section 2.2) using remote sensing data and also related works regarding a remote sensing dataset creation (section 2.3). The Ch. 3 then provides an introduction into deep learning — section 3.1 describes the early neural networks while the section 3.2 provides a short summary of the state-of-the-art deep learning. The section 3.3 describes the basic building blocks that constitute a neural network. The architectures that were used in this work are described in Sec.3.5. The used data are then described in Ch. 4, the data sources are introduced in section 4.3 (the aerial imagery) and section 4.4 (annotating the data using the OSM). The obtained data are described in section 4.1 and data sampling in section 4.5 and augmentation in section 4.6.

The Ch. 5 describes the method used and the individual experiments:

- comparing different architectures with transfer learning (section 5.1)
- learning from scratch of the most successful architecture from transfer learning (section 5.2) — same architecture is used for experiments below
- evaluation of the used data augmentation using networks trained from scratch (section 5.4)
- evaluation of the use of different activation functions using networks trained from scratch (section 5.5)

The Ch. 6 briefly describes technologies and software used for implementation of the work and also describes the implementation itself. Finally, the Ch. 7 provides the results for the experiments mentioned above. The Ch. 8 concludes the work and the Ch. 9 presents several direction for possible extensions of this work.

1.1 Motivation

The need to observe the Earth is as old as mankind but only with the invention of the camera were people able to capture images of the Earth. Images were taken from the surface in the beginning of photography but it has not taken long for the first aerial photos to emerge — the very first aerial photo was taken in 1860 from a balloon. However, it was not until the October 24, 1946 when the first images of the Earth were taken from the space — from 105 km above the surface [10]. These images were taken from a V-2 rocket; the first satellite images were taken by the U.S. Explorer 6 on August 14, 1959. A decade later, the Landsat program[1] started to provide up-to-date Earth imagery [127] for use in many fields — e.g. for studying the urban growth, climate change or biodiversity studies.

The Earth imagery was sparse and rare at the beginning and also available to only a few as it has been very expensive and of strategic value. The recent technological advances have, however, led to lower manufacturing and launch costs of the satellites and also allowed the satellites to be smaller and equipped with better sensors. This has led to an increase in bot demand and supply of satellite imaging and its analysis.

When remote sensing images were sparse and rare, a manual analysis was the norm, however manual analysis is quite costly for the amount of satellite images that need to be analyzed nowadays. This led to a rising demand for tools that were able to analyze the imagery automatically or, at least, simplify and speed up the analysis.

Many different methods were proposed in the field of computer vision but deep neural networks have emerged as the state-of-the-art tool for many tasks such as image classification [27, 57, 164, 170, 179], image segmentation [7, 165], and speech recognition [163].

1.2 Objectives and scope

The main goal of this thesis is to design, implement and experimentally evaluate a deep neural network for learning classification of satellite images using labels from OpenStreetMap (OSM) [141]. Most of the available satellite imagery datasets are manually labeled and thus are rather small, which makes training of deep networks rather difficult. Thus as the available datasets of remote sensing imagery are small or not suitable for some reason, the goal of this thesis is also to create a new dataset and annotate it using the OpenStreetMap platform [141]. OpenStreetMap is a crowd–sourced open, editable map built in *wiki* (e.g. Wikipedia [191]) style by many volunteers. This thesis describes the whole pipeline of the learning — obtaining the imagery, image preprocessing, creation of training, validation, and testing datasets with annotations, and experimental evaluation of selected state-of-the-art architectures.

The main contributions are:

- a new aerial dataset created remedying the problems with current datasets in the field

[1]viz `https://landsat.gsfc.nasa.gov/`

  - using annotations from OSM thus allowing simple extensions (more data from all over the world or, more classes) of the dataset in future
  - annotations for both classification (singe class per image) and labeling (multiple classes per image)

- a proposed and evaluated augmentation pipeline for multi-source data
  - the created dataset is multi-source (See section 4.3)
  - the augmentation brings 2 pp accuracy gains compared to the regular augmentation (validation accuracy 68.19% vs 71.80%, viz section 7.2.1)
- the performance evaluation of several pre-trained architectures and top layer configurations
  - pretrained on the ImageNet [158] and via transfer learning used on the created dataset
  - state-of-the-art architectures
- a trained network on the dataset
  - the selected network architecture is based on the results from the transfer learning experiment
- evaluation of suitability of several different activation functions for the provided network

Chapter 2

Related works

This section targets mostly the recent works focused on aerial image labeling. While this field is not recent, the early works utilized ad–hoc and knowledge–based approaches using quite small datasets and these works are not very relevant as the field has shifted significantly. A few examples of early works are [6, 11, 34, 55, 56, 73, 87, 117]. A review of classification approaches in remote sensing data that are not covered here is provided in [18, 46]. More details about remote sensing image scene classification method are available in a recent review [23] that has appeared concurrently with this thesis. To be even more specific, this work focuses mainly on high–level methods such as convolutional neural networks (CNNs) as these methods mostly dominate in classification of remote sensing images [23, 194].

2.1 Classification

One of the first classification tasks using remote sensing data was [56], where several small datasets were used. However the contribution of this work was several manually–crafted textural features as the individual classes had significantly different textures.

The neural networks (NNs) were used for classification of SAT-4 [8] and SAT-6 [8] datasets in [8, 153]. Both datasets are very large (SAT-4 has 500,000 samples and SAT-6 has 405,000 samples), however, they give almost no context and contain only a small number of classes, SAT-4 has 4 classes and SAT-6 has 6 classes [8]. The datasets consist of patches 28×28 pixels at 1 m per pixel resolution manually labelled and each patch has 4 spectral bands — red, green, blue, and near-infrared. The SAT-4 dataset consists of 400,000 training and 100,000 testing images as either barren land, trees, grassland or background class. The SAT-6 dataset consists of 324,000 training and 81,000 test images of barren land, trees, grassland, roads, buildings and bodies of waterclasses [8, 153]. The *DeepSat* [8] is an architecture where deep features are extracted first and then fed into a Deep Belief Network (DBN). The author of [153] has used a CNN for classification of the two datasets and has reached slightly better accuracy than the *DeepSat* (97.95% vs 99.54% on SAT-4 and 93.92% vs 99.44% on SAT-6) [153].

A CNN pre–trained on ImageNet [158] was used to classify the UC Merced Land Use dataset [197] and Brazilian Coffee Scenes dataset [145] by using *transfer learning* (for *transfer learning* viz section 5.1). Three CNN architectures learned on the ImageNet were used, two *OverFeat* [164] models (*OverFeat accurate* and *OverFeat fast*) and *Caffe* [81] pretrained *AlexNet* [96]. The *Caffe* based network had accuracy 93.42% $\pm$ 1.00 on the UC Merced Land Use dataset, while the *OverFeat* networks had 90.91% $\pm$ 1.19 (*OverFeat fast*) and 90.13% $\pm$ 1.81 (*OverFeat accurate*) and all three CNNs were more accurate than other evaluated descriptors. However, the networks were beaten by *color autocorrelogram* [71] and *Border–Interior Pixel Classification* [175] global descriptors on the Brazilian Coffee Scenes dataset [145].

A *CaffeNet* based on the *AlexNet* [96] was used for land use classification also using the UC Merced Land Use dataset [197] and Brazilian Coffee Scenes dataset [145]. The pretrained *CaffeeNet* on the ImageNet [158] was fine-tuned on UC Merced Land Use dataset and have reached even better accuracy than [145] with accuracy of 97.10%. The *CaffeNet* was learned from scratch on the much larger Brazilian Coffee Scenes dataset and the author reported accuracy of 91.8% — the authors have also tried using the pretrained network with fine-tuning but the performance was slightly worse (90.75%). A combination of both *OverFeat* and the *CaffeNet* have reached good results on the UC Merced Land Use dataset of 99.36% $\pm$ 0.63 (*OverFeat accurate + CaffeNet*) and 99.43% $\pm$ 0.27 (*OverFeat fast + CaffeNet*) [145].

A CNN network called *PatreoNet* was used in [129] for aerial scene classification using the UC Merced Land Use dataset [197] and Brazilian Coffee Scenes dataset [145] and reaching accuracy of roughly 90% on both datasets. An evaluation of different network architectures and transfer learning is in [130] where the authors took several different network architectures (*PatreoNet* [129], *AlexNet* [96], *CaffeNet* [81], *GoogLeNet* [179], *VGG16* [170], and both *OverFeats* [164]) and evaluated them on the UC Merced Land Use [197], Brazilian Coffee Scenes [145], and *WHU-RS19* [195] datasets. The authors have found that transfer learning with fine–tuning performs better than learning the networks from scratch [130]. However, that might have been caused by quite small datasets (each of these has less than 3000 samples) that might not have been sufficient to learn good feature extractors. This seems likely because both fine-tuned and learned from scratch networks have performed comparably on the Brazilian Coffee Scenes dataset that is larger compared to the other two datasets. The authors have also tried to use an SVM classifier using the deep features extracted by the individual networks and have found that it works slightly better than pure CNNs for both fine–tuned and fully trained networks[130].

A two–way bottom up approach is used in [105] for finding objects of given class on aerial images, however, details about the aerial dataset were not provided thus the performance is rather unclear.

A different approach was used in [154] where novel *quaternion based features* were proposed and then used for classification of the UC Merced Land Use [197] and Brazilian Coffee Scenes [145] datasets using an SVM classifier which resulted in quite good results — the accuracy was 92.29% on the UC Merced Land Use dataset and 90.75% on the Brazilian Coffee Scenes datasets.

This thesis focuses on the use of deep learning for classification in aerial imagery but, unlike most of the approaches in the published literature, it learns the deep network from scratch using the aerial imagery only — the other papers mentioned use either shallow networks or deep features extracted using networks pre-trained on other datasets or based on a random initiliaization.

2.2 Segmentation

While strictly speaking this work focuses on the classification and not on the segmentation, the segmentation approach is still relevant for several reasons. First, the early works in remote sensing were doing per-pixel classification using low-resolution data that were more connected to classification than segmentation. Second, the segmentation approaches more often used deep CNNs similar to the ways used in this thesis as used in this thesis (e.g. both this thesis and [170] use VGG network).

There are many published papers that focus on road or building segmentation. For example, road segmentation is done in [14, 36, 45, 122–125]. Another example of segmentation is in [91] where authors used covariance descriptors with a random forest to create a segmentation mask for 5 classes — *Building*, *Waterbody*, *Grassed Area*, *Tree*, and *Streetlayer*. The work was later extended in [92]. Note that both works have used aerial data together with height information reconstructed from triplets of aerial images with overlaps for the segmentation [91, 92]. The single class segmentation of roads was done in [36] where the main idea was the use of edge detection. Building segmentation was done in [110, 115, 128, 150]

A support vector machine was used for pixel–wise classification in [70, 172].

Neural networks for road segmentation were used in [123]. An early work featuring neural networks and comparing them with a Naive Bayess classifier is [34] — however, the used networks were very simple with at most 10 neurons in a single layer and at most two hidden layers. Both types of classifiers were used in [11] with the use of hyperspectral data, however, both the networks and dataset were very small — the network had 56 neurons and the dataset consisted of 2019 pixels . Pixel–wise classification was also done in [13], where a neural network was used to classify hyperspectral pixels into several classes. However,the datasets seem to be not very challenging as these pixels were low–resolution and covering many quite big areas with a single class thus the classification was able to work quite well with 98% accuracy [13].

A hallow neural network was also used in [143]. Multi–temporal data and a simple NN were used in [104] for cloud segmentation. A 4 layer feedforward NN was used for cloud classification from a single channel visible light data in [103].

A Fully Convolutional Network (FCN) for segmentation of the ISPRS Potsdam benchmark dataset [74] (6 classes: *impervious surfaces*, *building*, *low vegetation*, *tree*, *car*, *clutter/background*)is presented in [167]. This network presents a novel *no–downsampling* approach to preserve the output dimension and have finer segmentation [167]. A VGG network [170] pre-trained on the ImageNet [158] was modified for use in the FCN architecture and was used for segmentation over the *color infrared* CIR channels instead of RGB as input and a network trained from scratch was used for the the *digital surface model* (DSM) that contains the depth information — the outputs of the two networks were connected just before the fully-connected layers that were used for final output [167].

A CNN named *SatCNN* was used in [207] for classification of the SAT-4 and SAT-6 datasets. Object detection in aerial–images using the *contextual hierarchical model* was used for detection of trees, cars, roofs, roads, and parking lots in [149].

While segmentation is often closely related to classification in remote sensing applications, this thesis deals with the classification into many more classes than were used in the related segmentation work.

2.3 Dataset creation

Many works have created adataset for remote sensing but many of those are not suitable for all tasks. This section focuses mostly on classification datasets, although a few others are discussed as well. A brief summary of remote sensing image classification datasets in the literature is in table 2.1. A custom dataset for road segmentation was created in [125]. The author has used OpenStreetMap (OSM) [141] data together with MapBox Studio [113] to create segmentation labels for aerial images obtained using [52]. OpenStreetMap was also used for the creation of building segmentation dataset in [122], where the author has created the Massachusetts Buildings datasets using labels taken from the OSM using satellite imagery. The dataset's labels are quite accurate because the city of Boston has provided the building footprints to the OSM project [122]. The author has also used the same approach to create Massachusetts Road dataset for road segmentation [122]. A small segmentation dataset created using Google Maps [52] for both imagery and segmentation mask was created in [36].

A hand labeled object detection dataset using Google Earth was created in [149] — the dataset consists of 196 manually labelled images that contain a larger number of objects: 10477 cars, 973 roofs, 202 roads, 584 parking lots, and 555 tree regions [149], and it can be considered one of the first larger datasets. Another hand labeled dataset for object detection was presented in [24] and it contains 10 diffent classes. Two datasets, SAT-4 and SAT-6, are created in [8] — while these datasets are quite large (500,000 and 400,000 patches), they contain only 4 and 6 classes respectively [8].

A small dataset using imagery from IKONOS satellite is used in [198], it consists of 10 classes by 100 panchromatic images 64×64 px with a spatial resolution of 1 m that were manually extracted from the original IKONOS images [198].

Two small *SPOT5* datasets were created for the purpose of [111], each dataset contains 304 panchromatic images in three classes (97 images in *Building* class, 97 images in *Vegetation* class and 110 images in *Farm* class). The two datasets differ by the spatial resolution and size of their images, the first dataset has a spatial resolution of 2.5 m and 512×512 px patches, the second dataset has a spatial resolution of 10 m and 128×128 px patches [111].

A *ORNL-I* dataset was first introduced in [185] which consists of 850 512×512 px images distributed into 5 classes (*agricultural*, *large-facility*, *commercial*, *suburban*, and *wooded*) [185]. An extended version of *ORNL-I* dataset called *ORNL-II* is presented in

[25], it consists of 512×512 px patches having cca 1 m spatial resolution. The dataset is binary and consists of *large–facility* class (153 samples) and the *background* class (974 samples) [25].

The UC Merced Land Use dataset was developed in [197] for the evaluation Bag-of-visual-words approach on the land-use classification, thus the dataset have a smaller number of examples for each class than the datasets usually have for learning CNNs. It consists of 100 256×256 px RGB patches with a spatial resolution of 0.3 m for each of its 21 classes.

An RGB–NIR dataset (*In-House dataset*) was created in [5] for testing multispectral descriptors. The dataset consists of 850 RGB and NIR 80×80 px images of spatial resolution of 1 m manually labeled into 5 classes (*grayfield*, *greenfield*, *houses*, *river*, and *woods*) [5]. Another version of *In-House dataset* was introduced in [155], where the RGB image (4500×6000 px) was divided into 606 128×128 px tiles that were manually classified into one of 6 classes (*houses*, *cemetery*, *industry*, *field*, *river*, and *trees*)[155]. However both datasets are a bit problematic as all the images are from a single scene that was cut into the patches, thus the patches do not exhibit great variance.

The Brazilian Coffee Scenes dataset was created for the purposes of [145] and it consists of 64×64 px patches of three classes — *coffee* ($> 85\%$ coffee pixels), *non–coffee* ($< 10\%$ coffee pixels) and *mixed* (patches not fitting into the previous descriptions). The dataset is quite large as it has 36,577 samples of *non–coffee*, 1,438 samples of *coffee*, and 12,989 samples of *mixed* [145]. The images consists of three spectral bands — red, green and near infrared [145]. A similar dataset, hte *Brazilian Cerrado-Savanna Scenes Dataset*, was published with [131] for vegetation classification. It consists of 1,311 64×64 px images with 3 spectral bands (red, green, near infrared) with a spatial resolution of 5 m distributed in 4 classes (*agriculture*, *arboreal vegetation*, *Herbaceous vegetation*, *Shrubby vegetation*).

A dataset with 11 classes was created in [206] for evaluation of a *multibag-of-visual-words* model. The dataset was extracted from Google Earth and manually labeled, it consists of 1232 RGB images of size 512×512 px and spatial resolution of 0.2 m divided into 11 classes each represented by roughly 100 images. This dataset was referenced as *RSC11 dataset* in [23].

This work has also made use of a *WHU-RS19* dataset created in [195]. However, it is unclear what dataset was used as the original data are no longer available and the description in the original work [195] differs from the description in following works [20, 166, 206]. Furthermore, it is unclear who originally created the dataset as two different works claims its authorship [32, 195]. The "original" works do not provide any details about the dataset except that it consists of 12 different classes each having 50 images, while the rest of the works describe the dataset as having 19 classes each consisting of 50 images 600×600 px [21, 23, 166, 206] at spatial resolution up to 0.5 m [206]. It seems that the authors of [32] or [195] had published an extended version of the dataset used in [32] or [195] without making it clear that the dataset used in the work differs from the published dataset that was later used in the following works. This possibility is supported by the work in [23] which states that there are two versions of this dataset.

Another classification dataset was presented in [205], which, similar to the other datasets discussed here, contains 2,400 images obtained from Google Earth and manually labeled. It contains 12 classes, each with 200 images of size 200 × 200 px and a spatial resolution of 2 m. While the dataset is referred to only as the *Google dataset* in the original work, it appears named as *SIRI-WHU* in [23].

All of the previous datasets suffer from at least one weakness for classification using NNs — too few examples, too few classes, low variation, or too small patches — which is the reason why this thesis and [23, 194] introduce new datasets. The *Aerial Image Dataset* (AID) is created in [194]. It contains 10,000 600 × 600 px images with spatial resolution ranging from 0.5 m to 8 m per pixel — the data were manually labeled by "the specialists in the field of remote sensing image interpretation" [194]. The dataset contains 30 classes each having from 220 to 420 images and the data were sampled from selected countries worldwide — mainly from China, the United States, England, France, Italy, Japan, Germany [194].

Concurrent with our work, a new dataset has been created in [23]. It is the *NWPU-RESISC45 dataset* for classification. The authors of [23] have created a dataset of 256 × 256 px patches with spatial resolution ranging from 30 m to 0.2 m. The dataset consists of 31,500 images divided into 45 classes of 700 images each. While this dataset solves all of the mentioned weaknesses by providing a large number of samples with a large number of classes, it consists of images of different spatial resolution as the author wanted to make the dataset challenging [23]. The authors review existing datasets but they seem to miss some of the relevant datasets in the field — they do not mention the AID [194] dataset from a previous year that already solves most of the weaknesses of other datasets and both its size and number of classes is comparable to their NWPU-RESISC45 dataset (AID has 30 classes while NWPU-RESISC45 dataset has 45 classes).

However, both NWPU-RESISC45 and AID do not have imagery with a single spatial resolution but rather have images with different spatial resolutions. This might not be the best idea as the aerial/satellite imagery is obtained usually with known spatial resolution and large batches of images at given spatial resolution are usually provided. Thus it makes more sense to provide individual datasets for different spatial resolutions as it is a more frequently used case — unlike for the "ground" image datasets such as ImageNet [158] where objects naturally occur at different scales [167].

This thesis, similar to mentioned publications, deals with the creation of a remote sensing image dataset but it differs from related work in several points:

- the images are annotated using OSM, not manually labelled
- the size of created datasets is significantly higher than the existing dataset
  - with the exception of the NWPU-RESISC45 dataset that has comparable size (60% compared to the created E44 dataset) and that was published concurrently with our work in [23]
- compared to the two larger datasets (NWPU-RESISC45 and AID), the spatial resolution is fixed

year	name	use	images	classes	bands	size [px]	res. [m]	source	link
2008	IKONOS	C	1000	10	P	64×64	1	[198]	NA
2008	ORNL-I	C	850	5	P	512×512	1	[185]	NA
2010	UC Merced Land Use	C	2100	21	RGB	256×256	0.3	[197]	[199]
2010	WHU-RS19	C	1005	19	RGB	600×600	up to 0.5	[166, 195]	NA
2011	In-House (RGB)	C	606	6	RGB	128×128	1	[155]	[4]
2013	SPOT5 (2.5)	C	304	3	P	512×512	2.5	[111]	NA
2013	SPOT5 (10)	C	304	3	P	128×128	10	[111]	NA
2014	ORNL-II	C	1127	2	P	512×512	1	[25]	NA
2014	In-House (RGB–NIR)	C	850	5	RGB,NIR	80×80	1	[5]	[4]
2015	Brazilian Coffee Scenes	C	36,577	3 (2)	RG,NIR	64×64	NA	[145]	[146]
2015	SAT-4	C	500,000	4	RGB,NIR	28×28	1	[8]	[9]
2015	SAT-6	C	405,000	6	RGB,NIR	28×28	1	[8]	[9]
2016	RSC11	C	1232	11	RGB	512×512	0.2	[206]	NA
2016	SIRI-WHU	C	2,400	12	RGB	200×200	2	[205]	NA
2016	Brazilian Cerrado-Savanna Scenes	C	1,311	4	RG,NIR	64×64	5	[131]	[146]
2016	AID	C	10,000	30	RGB	600×600	0.5 – 8	[194]	[193]
2017	NWPU-RESISC45	C	31,500	45	RGB	256×256	0.2 – 30	[23]	[22]
2017	**CLS20** (proposed)	C	**25,551**	20	RGB	400×400	0.7	—	not yet
2017	**CLS44** (proposed)	C	**52,596**	44	RGB	400×400	0.7	—	not yet
2017	**LAB20** (proposed)	L	**25,551**	20	RGB	400×400	0.7	—	not yet
2017	**LAB20S44** (proposed)	L	**52,596**	20	RGB	400×400	0.7	—	not yet
2017	**LAB44** (proposed)	L	**52,596**	44	RGB	400×400	0.7	—	not yet

Table 2.1: List of remote sensing classification datasets. The column **use** contains "C" for strictly classification datasets (one possible class per image) and "L" for labeling datasets (more possible labels per image). The column **bands** contains "P" for panchromatic images, "R","G", and "B" for red, green and blue bands, and "NIR" for near–infrared band.

2.3.1 Use of *OpenStreetMap*

The OpenStreetMap [141] (viz section 4.4) is created and maintaned by many volunteers all over the world (similarly as Wikipedia [191]) and it was already used as source of data for creation of segmentation ground truth mask in [125]. The OpenStreetMap was used together with MapBox Studio [113] for styling the mask to create road segmentation *Prague dataset* [125]. Somewhat similar approach has been undertaken in [76] where both Google Satellite (actually aerial) Map and classical Google Map were used as source of data. However, the goal was to show the abilities of the *Conditional Adversial Network* [76] rather than serious segmentation task — the network should produce a good looking map from the aerial map (and vice versa). It is unclear how well could this approach work as the authors used it only as a proof-of-concept with only small datasets that contained only the most common objects — the network failed on less frequent object such as lakes, etc.

Chapter 3

Neural networks and deep learning

3.1 Beginnings

The field of neural networks (NNs) is very broad as NNs have been around for several decades and have experienced a boom in the last two decades, thus this chapter will provide only a shallow overview of only one sub–field — the Deep Learning (DL) in Neural Networks. The DL has become very popular few years ago as it can accurately solve many problems that were very hard to solve before.

It is hard to pinpoint the first neural networks as the field has evolved gradually and the earliest NNs were based on linear regression methods which were around for an even longer time [160] — the firsts works with linear regression appeared in the beginning of the 19th Century [160]. The early NNs were not able to learn from the data [116, 160] — McCulloch has introduced neural networks as a logical calculus in [116]. The first learnable network was presented in 1949 in [60] (reference from [160]), where Hebb introduced the idea of unsupervised learning for the NNs. Then approaches to supervised learning in NNs were introduced (e.g. the perceptron algorithm in 1958 [156], more examples in [160]).

Some authors consider the networks trained by the *Group Method of Data Handling* (GMDH) in 1965 [77, 78, 119] to be the first general working learning algorithm for supervised deep feedforward multilayer perceptron [160]. Another example of an early deep learning architecture is *Neocognitron* which also first introduced the idea of convolutional NNs (CNNs) [42, 160] that are nowadays used very widely in pattern and image recognition tasks — the network presented in [42] has seven layers. While the *Neocognitron* resembles the modern NNs architectures, it does not feature a supervised learning algorithm and the weights were rather set using local unsupervised learning rules [42] or were pre–wired [160].

The use of backpropagation (BP) in NNs first appeared in 1981 [160] in [190] even though the BP was first described in 1970 [109, 160]. During the 1980s, the BP became quite popular, e.g. [99, 100, 157], though it seemed that BP was usable only for shallow networks [160]. Many improvements to the steepest descent used in BP were proposed in the following years — e.g. the momentum to accelerate the BP [157, 160], even though the core idea of momentum can be traced back to 1964 to [148]. Another improvement was the well–known algorithm *RProp*, which is a variant of BP that takes into account only the sign of the error derivatives.

The convolutional networks were first trained with BP in 1989 when LeCun applied it to a network *LeNet* similar to *Neocognitron* to recognize handwritten digits of the MNIST dataset [99, 160], which was the beginning of the massive use of CNNs — the state-of-art networks are still very similar to LeCun's network [160].

However, most of the used networks at the time were still shallow as the deep networks suffered from the vanishing (or exploding) gradients — also known as the *Fundamental Deep Learning Problem* [51, 160] — which was first described in [63] (ref. from [160]).

3.2 Winning competitions

Various competitions help to find better algorithms/approaches in machine learning and the NNs gained much more attention once they started to win these competitions often — especially in the field of pattern recognition. Even though the NNs won several competitions back in the 90s [160], the breakthrough was made by deep network *AlexNet* in 2012 when the network won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge [158]) while significantly outperforming the second best entry (top-5 test error rate of 15.3%, compared to 26.2%) [96]. The *AlexNet* had 60 million parameters and 650,00 neurons in 8 layers — it had reached the limit of the contemporary hardware and it had to be trained simultaneously using 2 GPUs.

The ILSVRC 2013 confirmed the position of CNNs and was won by *ZF Net* which is a CNN based on the *AlexNet*. While it reached an even lower error rate (11.2%), the network otherwise remained very similar to the *AlexNet* and the main contribution of the paper, in which it was introduced, was a technique for visualization of feature maps and analysis of the network to various transforms of the input data [203].

One of the advantages of DCNNs is that they can be relatively simple and still achieve great performance if they are deep enough as was shown by the VGGs networks in [170] — the VGGs networks used only 3×3 filters (the smallest possible still capturing the positional information) with a stride and pad of 1. The networks also contained 2×2 maxpooling layers after some of the convolutional layers and the last three layers were fully connected. The authors have proposed various networks of this architecture from 11 to 19 layers [170]. The previous networks have often used larger convolutional layers (e.g. 7×7 in [203] or 11×11 in [96]) but [170] shows that similar effects can be achieved using more 3×3 convolutional layers with the advantage of having a lower number of parameters to optimize.

The DCNNs also do not have to consist of several convolutional, maxpooling and dense layers stacked after each other in a simple chain. A diversion from this scheme was shown in [179] where the network *GoogLeNet* was introduced — the network contains parallel connections instead of simply chaining all layers. The *GoogLeNet* contains *Inception* units (a network-in-network) which have introduced parallel connections. Otherwise the architecture remains sequential. Each *Inception* unit consist of several different parallel streams with dimensionality reduction as shown in fig. 3.1, the dimensionality reduction is necessary because otherwise the depth of a single unit would be unmanageable. The whole

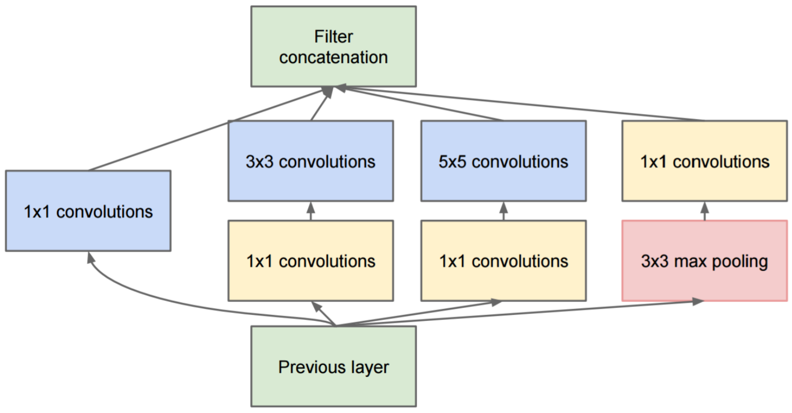


Figure 3.1: The Inception unit. It contains parallel connections that are concatenated together. The 1×1 convolutional layers are for dimensionality reduction. Adapted from [179].

architecture also focuses on computational efficiency — the network does not contain any dense layers as they were replaced by average pooling layers and the whole network has $12\times$ fewer parameters than the *AlexNet* [179].

Another famous network is the *ResNet* [57] which won the *ILSVRC 2015 classification competition* with a great error rate of 3.57% (actually, this error rate was achieved by an ensemble of *ResNet* networks). Aside from being the winner of the prestigious ILSVRC 2015 [158] (and also *ILSVRC 2015 ImageNet detection*, *ILSVRC 2015 ImageNet localization*, *COCO 2015 detection*, and *COCO 2015 segmentaion* [108] competitions), the ResNet architecture is also unique for its depth — one of the networks in the paper had 152 layers which is unprecedented, and surprisingly, the network still has lower complexity than the VGG networks [57]. To facilitate the learning of such deep network without degradation, the authors propose the *deep residual learning* framework viz section 3.5.4. Despite the novel framework, the core idea can be easily used as it can be implemented by simply adding a *shortcut connection* (also called *skip connection*) between layers which forces the layers to learn a residual mapping, viz fig. 3.2 for comparison with the VGG network architecture. Even though shortcut connections have been used before — e.g. with multi-layer perceptrons (MLPs)([152] ref. from [57]) or an equivalent of connection intermediary layers to an auxiliary classifier (e.g. *GoogLeNet* [179], or *deeply–supervised nets* (DSN) in [102]) — the ResNets were the first architecture where significant increase in depth led to accuracy gains [57].

3.3 Building blocks

Deep neural networks are a very modular approach and they can be broken into several blocks which are made from individual neurons and the complexity of an neural network architecture can vary from very simple (e.g. a multilayer perceptron — MLP [157]) to

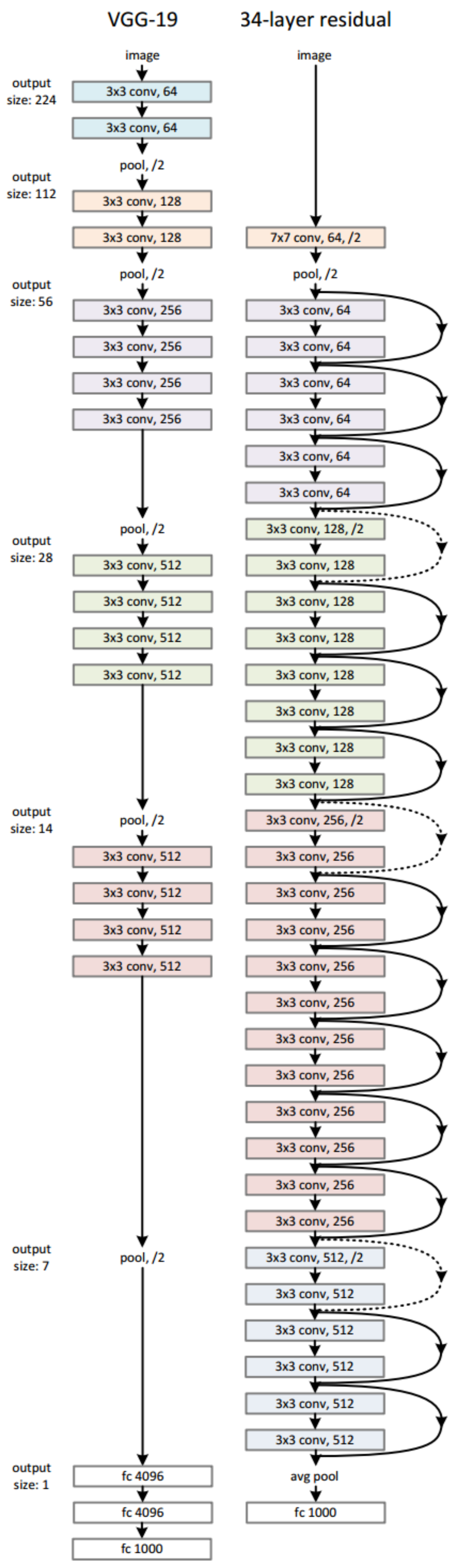


Figure 3.2: Comparison of VGG19 architecture and ResNet34. The ResNet34 is almost twice as deep as the VGG16 but it has fewer filters and lower complexity. Adapted from [57].

very complex (e.g. GoogLeNet [179]). The NNs are best explained using simple examples at first and only then more complex ones. The goal of this section is to provide a brief overview of the deep learning; a more detailed overview is available in [51, 86].

An artificial neural network (NN) is a computational model that is represented by a weighted graph — nodes are individual neurons and edges show paths by which a signal flows. A single neuron takes all of the inputs and combines them into a single output signal which is then sent to all of the outputs of the neuron. The individual inputs are usually combined using weighted summation where weights are the defined by the connections by which the signal came into the neuron. More specifically, let us have a single neuron i with input signals $x_0, x_1, \dots, x_n$ which flows through input connections with weights $w_0, w_1, \dots, w_n$, a bias term b and an activation function $f(x)$, then the output x_i of the neuron at time t is

$$x_i[t] = f\left(b_i + \sum_{j=0}^{n} w_{i,j} x_j[t-1]\right) \tag{3.1}$$

as described in [41]. Furthermore, when the time domain is not needed, the time indices are dropped [61] ref. from [41]. The Eq. 3.1 is often written in a matrix form:

$$x_i = f\left(\mathbf{w}_i^{\mathrm{T}} \mathbf{x}\right), \tag{3.2}$$

where

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_0 \\ \vdots \\ x_n \end{bmatrix} \tag{3.3}$$

and

$$\mathbf{w}_i = \begin{bmatrix} b_i \\ w_{i,0} \\ \vdots \\ w_{i,n} \end{bmatrix} \tag{3.4}$$

Some authors consider the activation function be applied after combination of the inputs (as above) while others consider the activation function be applied directly on the inputs (and thus the input combination is part of the function). Since the most common approach in pattern recognition is combining the inputs by summation, most of the researchers in the field define the activation function as a function on combined inputs and therefore the naming used here considers the activation function to be $f(x)$.

The most simple activation is a *signum*-like function which is used in perceptron:

$$y = \begin{cases} 1 & b + \sum_{j=0}^{n} w_j x_j > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where b is the bias term, $x_0, \dots, x_n$ are real valued inputs and $w_0, \dots, w_n$ are weights.

A slightly more complex example is shown in fig. 3.3, which depicts neurons in layers. Note that for historical reasons, the intermediary layer between the input and output

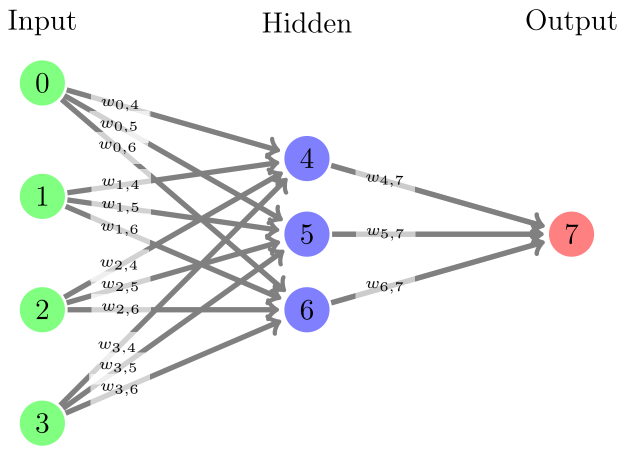


Figure 3.3: An example of a simple feedforward neural network with 3 input neurons, 2 neurons in the hidden layer and 1 output neuron.

layer is usually called hidden, however, this notation is less common in the case of neural networks that have many hidden layers. The feedforward (i.e. no loops in the graph) NNs usually have used (more about currently used activation functions in section 3.3.2) the *sigmoid* activation functions — often–used candidate was the *logistic function*:

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{3.6}$$

The output$y$ of the example network from fig. 3.3 for given inputs $x_0, \ldots, x_3$ is:

$$y = \mathrm{S}\left(b_7 + w_{4,7}\mathrm{S}\left(b_4 + \sum_{j=0}^{3} w_{j,4}x_j\right) + w_{5,7}\mathrm{S}\left(b_5 + \sum_{j=0}^{3} w_{j,5}x_j\right) + w_{6,7}\mathrm{S}\left(b_6 + \sum_{j=0}^{3} w_{j,6}x_j\right)\right) \tag{3.7}$$

where S is the sigmoid activation function and $w_{i,j}$ describes the weight of the connection between neurons i and j. Note that if the network had more layers or more outputs, there would be elements in the formula that occur several times — only the inputs $x_0, \ldots, x_3$ occur repeatedly in the example — thus the computation would be done in stages to avoid repetitions.

3.3.1 Layers

The neural networks are usually organized into layers as in fig. 3.3 and neurons in these layers are usually of the same type. This section introduces several common layers that are frequently used in NNs — note that this list is not exhaustive and focuses especially on the layers that are of particular interest to this work.

3.3.1.1 Dense layer

Dense layers (also called *fully connected*) were already introduced in the example in fig. 3.3. A dense layer consists of regular neurons as described in Eq. 3.1 whose inputs are connected to all outputs of the preceding layer (hence the name *dense*). The early NNs usually used only this type of layer, however, *dense* layers have an enormous number of parameters and are not suitable if the number of neurons is large. This is one of the reasons why these layers cannot be used alone for image classification. Furthermore the learning of such networks can be quite difficult as such networks are rather susceptible to overfitting (several possible ways to address this issue were proposed such as *dropout* [174], L1 and L2 regularization or soft weight sharing [133]). Modern networks for image pattern recognition usually use *dense* layers only as the top layers that utilize the features extracted by other types of layers (e.g. [57, 170, 179]).

3.3.1.2 Dropout layer

As was mentioned above, *dropout* is a technique for limiting the overfitting — it approximates the combining of exponentially many different neural network architectures efficiently [174]. The method randomly temporarily drops out neurons from the network along with all its incoming and outgoing connections. Each neuron is given probability p that it remains active, otherwise it is dropped out during the training phase. All neurons when using dropout as described in [174] are active during the testing thus the outputs from the neurons need to be scaled down in order to average the outputs and reach similar values to those used during the training phase [174].

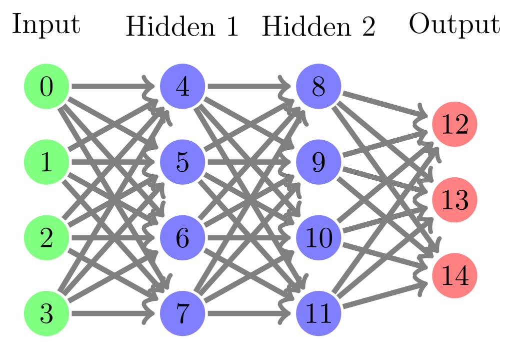
Most applications however use the *inverted dropout* which, unlike classical dropout from [174], scales the outputs during the training phase and thus no scaling is necessary during the test phase [86]. A similar concept is *DropConnect* which drops out individual connections instead of whole neurons [188]. The *DropConnect* achieves slightly better result than classical dropout [188], furthermore, its modified version *Sparse DropConnect* seems to achieve even better results [107]. Examples of different dropout approaches are shown in fig. 3.4.

The dropout is a type of regularization of the optimization [174] — regularization typically leads to a preference of certain types of weight over another [51].

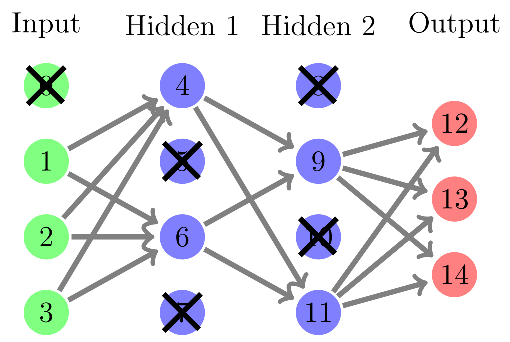
While technically *dropout* layer is not a layer but rather a method of regularization, it can be implemented by a layer that randomly switches off a fraction of outgoing connections during training phase — e.g. the Keras implementation [26].

3.3.1.3 Convolutional layer

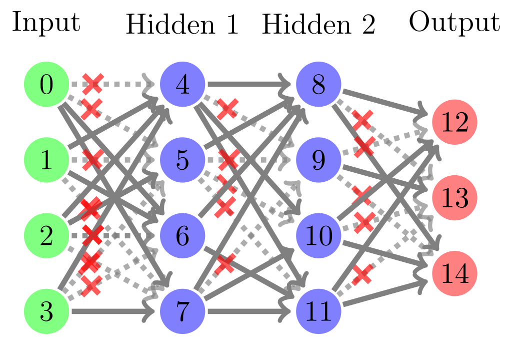
The convolutional layer might be the most important type of a layer for image pattern recognition as it allows extraction of spatial features in the image without enormous computational costs. Convolutional layers are based on the ideas of *parameter sharing*, *sparse interactions*, and *equivariant representations* [51]. Convolutional layers are used in applications where arrangement of inputs carries spatial or temporal information — e.g. video



a without dropout



b with dropout



c with DropConnect

Figure 3.4: Example of dropout regularizations

(3D) [20, 169] images (2D) or natural language (1D) [88]. This section focuses on the use of the 2D convolution in image processing however the same principles apply in other fields.

The convolutional layer basically applies filters to the input data[1] and thus it adds a another dimension to the image data as several different convolutions are usually used and each of them results in a different feature map as shown in fig. 3.5. The fig. 3.5 shows a convolution layer with 9 filters being applied to an input image. Note that the fig. 3.5 also shows *pooling* layer (viz section 3.3.1.4) and another convolutional layer, this time with 36 filters.

The convolutional layer is defined by several parameters — width w and height h of the kernel, number of filters d, padding p , and stride s. The width and height of the kernel determine the size of the matrix defining the kernel; most common kernels in image pattern recognition are square kernels (but e.g. Inception 3 uses two layers of 3×1 and 1×3 convolutions instead of a single 3×3 convolution, viz section 3.5.2) . The width w and height h define the *receptive field* [86] of the layer. The parameter d determines the number of filters that will be applied to each input and thus it represents the depth of the output volume, a single feature map at a certain depth is called *depth slice*.

The padding p determines whether and how will be the input volume padded by zeros. The example in Fig. 3.5 has no padding thus the width and the height of the output are lower than the width and height of the input due to the convolution — no padding is sometimes called the *valid* padding (e.g. [26, 51]) and the convolution without padding is sometimes called the *narrow* convolution (e.g. [85]). The padding is most often used to keep the output spatial dimension same as of the input, this type of padding is often called the *same* (e.g. [26, 39, 51]) or the *half* padding (e.g. [39]). For example, if the layer had stride $s = 1$ and $h = w$ then the *same* padding keeping the spatial dimensions would be $p = \frac{w-1}{2}$. An extreme case is the *full* padding which pads the input with enough zeros such that every input pixel is visited equally — the border pixels in the case of the *same* padding are underrepresented in the model [39, 51].

Stride s sets how often the filter will be applied to an input pixel, $s = 1$ means that the filter will be applied to every pixel and, for example, $s = 3$ means that the filter will be applied to every third pixel. The stride determines downsampling of the input of the convolution in the spatial dimensions — it is equivalent to a regular convolution with stride $s = 1$ followed by downsampling. This is, however, less computationally efficient than a convolution with a stride [39, 51].

The output volume of a convolutional layer $H_o \times W_o \times D_o$ is determined by the parameters in following way [86, 125]:

$$H_o = \frac{H_i - h + 2p}{s + 1} \tag{3.8}$$

$$W_o = \frac{W_i - w + 2p}{s + 1} \tag{3.9}$$

$$D_o = d \tag{3.10}$$

[1]Many implementations of convolutional layers often do not really use *discrete convolution* but rather *cross-correlations* which is technically slightly different but the difference is only an implementation detail, for further details see [51, p. 333].

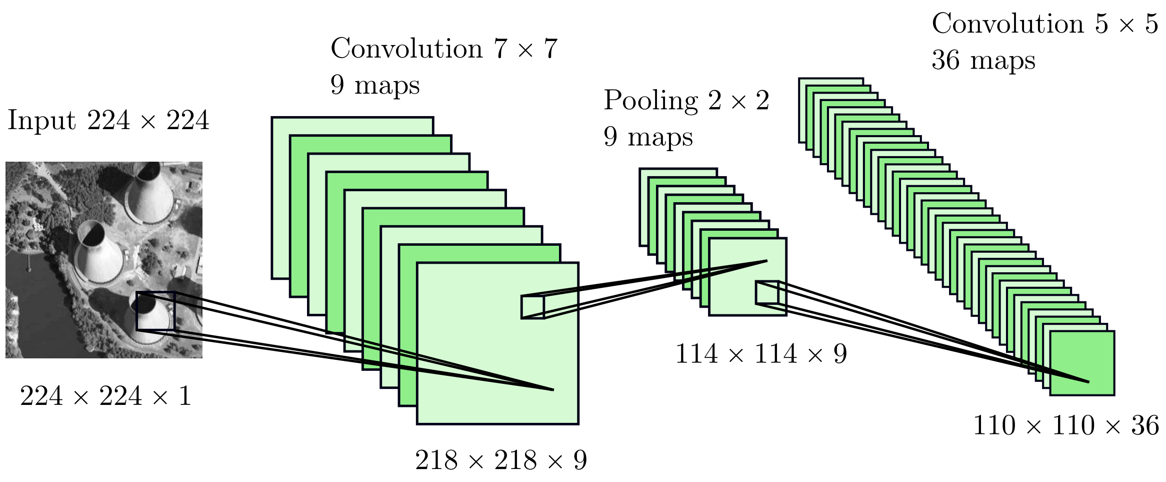


Figure 3.5: Example of usage of convolutional and pooling layers. Note that this example has no padding and thus the convolutions also reduce dimension. More about pooling layers in section 3.3.1.4.

where h, w, p and s are defined above and H_i and W_i is the spatial dimension of the input volume.

The convolutional layers utilize *weight sharing* as the same filter is applied to different pixels while the traditional *dense* layer has an individual weight for each of its inputs. This leads to enormous drop in number of parameters if the kernel size is sufficiently smaller than the input spatial dimension. The convolutional layer has one set of weights for each depth slice — all neurons within the slice have the same weights [86].

Sparse interactions (*sparse connectivity*) between neurons are another idea utilized in convolutional layers [51, 86]. The interactions are defined by the size of the kernel of the convolutional layer — e.g. if the kernel size is $n \times n$ then the size of *receptive field* of each neuron in the layer is $n \times n$, i.e. the neuron is not connected to all neurons in the preceding layer but only to $n^2 d_i$ neurons where d_i is the depth of the input. The *sparse connectivity* also leads to lower number of parameters of a neuron, for example, if the input volume has size $224 \times 224 \times 3$ and the filter size is 5×5, then each neuron in the convolutional layer will have $5 \cdot 5 \cdot 3 + 1 = 76$ parameters while a neuron in a dense layer would have $224 \cdot 224 \cdot 3 + 1 = 150529$ parameters.

A very important property of convolutional layers is the *equivariance* to a translation [51] which means that shifting the output of the convolutional layer is equivalent to shifting the input and then applying the convolution (except for border areas). This is a desirable property when we want to detect features that appear at different locations in the input which happens in typical image applications quite often (not always, the images might be centered) [51]. If the *equivariance* to translation is not a desired property, the convolutional layer might be replaced by *locally–connected layer* which is basically a convolutional layer without strict weight sharing [86].

3.3.1.4 Pooling layer

A pooling layer is a layer reducing the spatial dimension of the input and it is very often used together with convolutional layers [51, 86]. The core idea is similar to convolutional layers but instead of convolving close pixels with kernel a pooling function is applied instead.

The most common pooling is *max pooling* where the pooling function outputs the maximum value of the close pixels [39, 51, 86, 209]. Examples of other used pooling functions are *average* [39, 51, 86], *L2–norm pooling* [51, 86], or weighted average based on the distance from the central pixel [51].

Not only the pooling layer reduces the number of parameters and thus reduces the possible overfitting, it also helps the network to be invariant to small translations of the input [51]. Pooling layers have similar parametrization as convolutional layers — width w, height h, stride s and pooling function f. The width w and height h determine the size of a rectangular neighbourhood that is pooled together by function f. Stride s has the same meaning as in convolutional layers and determines how often the pooling will be applied [51]. The pooling is usually applied to non-overlapping patches hence the stride is used more often than in convolutional layers, for example, very common pooling layer has $w = 2$ $h = 2$ (filter 2×2) together with $s = 1$ and max pooling function [86, 125] which results in downsampling and getting rid of 75 % outputs [86]. The dimension reduction makes the network less computationally demanding.

The output volume of pooling layer $H_o \times W_o \times D_o$ is determined by the parameters in following way [86]:

$$H_o = \frac{H_i - h}{s + 1} \tag{3.11}$$

$$W_o = \frac{W_i - w}{s + 1} \tag{3.12}$$

$$D_o = D_i \tag{3.13}$$

where h, w, and s are defined above and H_i, W_i, D_i is the dimension of the input volume.

The *pooling* layers are not necessary and can be efficiently replaced by convolutional layers with a stride which might be often useful because some architectures and approaches do not work well with pooling [86, 173]. For example, visualization by deconvolving the output requires saving *switches* of *max pooling layer* during forward pass and thus the visualization is conditioned by the forward pass — a network with convolutional layers does not require switches thus it allows visualization unconditional on a image from a forward pass [173]. A simple *all convolutional* network without any pooling layers was shown to work well or even at state-of-the-art performance on various image classification datasets (CIFAR-10, CIFAR-100 [95], ImageNet [158]) [173].

3.3.1.5 Activation layer

Sometimes an activation function of individual neurons is considered to be an individual layer [26, 51]. While this distinction is not always used, it is useful when describing more complex architectures. Further distinction is made in [51], where two main conventions are described — *complex layer terminology* and *simple layer terminology. Complex layer terminology* considers set of convolution (*convolution stage*), activation function (*detector stage*), and pooling (*pooling stage*) to be a single layer [51]. On the other hand, the *simple layer terminology* considers it to be three separate layers — consequently, not all layers have parameters to be learnt [51]. The *simple layer terminology* seems to become prevalent as the pooling layer is not used after every convolution and sometimes even completely omitted [173].

The activation layer only applies an, usually non–linear, *activation function* to its inputs, commonly used activation functions are described in Sec. 3.3.2.

3.3.2 Activation functions

Activation function is used to control what the neurons outputs (*fires*). The choice of an activation function significantly influences what the network is able to model and how difficult is the network to train. The non–linear functions are what makes deep network useful for non–trivial problems — it can be easily shown that classical multi–layer perceptron with linear activation function is equivalent to a single layer perceptron — just a linear combination of its inputs.

The non–linearity of activation function (used to be sometimes called *squashing function*) is what makes theoretical representational power of neural networks — it was shown that any continuous function on compact subsets of $\mathcal{R}^n$ can be approximated by a *feed–forward* NN with a single hidden layer [68]. It was first proven for *sigmoid* function only [31] and then for any continuous, bounded and nonconstant activation function [67]. However this only describes theoretical representative power not its practical usability and trainability and it does not make deep learning obsolete (even though some scientists in the 90s presented this theorem as an argument why deep networks are *not* needed [160]). It is worth noting that not all activation functions are static in terms of learning — some might have parameters that are to be learned during the network training (e.g. *soft exponential* [49] or *adaptive piecewise linear* unit [1]).

3.3.2.1 Logistic function

Logistic function used to be one of the most used activation functions in NNs [86]. It is a function of *sigmoid* type and often when activation function is called *sigmoid*, it actually means that the activation function is the *logistic* function. The logistic function is defined as [51, 86]:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{3.14}$$

The function has, however, two undesirable properties. First, *sigmoids* in general have gradients very close to zero when the activation saturates for large values at either 0 or 1 [86] — such gradient limits the signal flowing to the weights of neuron and in turn makes learning much harder [86]. This is important during initialization which should avoid having too many neurons saturated or the learning will go very slow or it will not converge at all. Second undesirable property is that the logistic function is not centered at zero and is strictly positive which might make the learning a bit slower compared to other sigmoidal function — *tanh* [86]. The logistic function is usually not used anymore in feedforward networks [51, 86] where it was overtaken by *rectified linear units* (ReLU) and its modifications.

3.3.2.2 Tanh function

The tanh function is another sigmoidal activation function, similarly as the logistic function, it also gradients close to zero when saturated [51]. It squashes a real-valued number into range $(-1, 1)$ — unlike logistic function, it is centered around zero thus it is a better choice than the logistic function [51, 86]. The tanh function is actually just a transformation of the logistic function $\sigma(x)$ as shown in Eq. 3.15 [51, 86].

$$\tanh(x) = 2\sigma(2x) - 1 \tag{3.15}$$

3.3.2.3 Rectified linear function (ReLU)

Rectified linear unit (ReLU)[126] is probably the most popular activation function in the state-of-the art feedforward networks [48, 51, 86]. It have been found that the ReLUs can significantly accelerate the convergence of stochastic gradient descent [96]. Furthermore, the traditional ReLU is much less computationally expensive than activation functions such as the logistic or tanh functions [86] and it often outperforms sigmoidal activation functions [48]. The only problem of ReLUs is that might get disabled during training – i.e. they won't be ever activated again for any input and the output gradient will permanently be zero [86]. This might happen after a weight update after large gradient flows through the unit [86]. Many ReLUs modification and derivations were proposed [120, 121] — e.g. *Leaky ReLU*, *very leaky ReLU*, *parametric ReLU* [58], *randomized leaky ReLU* [196] or *S-shaped ReLU* [82]. Smoothed modifications are, for example, *exponential linear unit* [29] and *SoftPlus* [48]. Most of the modifications solve the problem of dying out neurons as they allow for gradient flows for any input. A ReLU is defined as [48, 121]:

$$f(x) = \max(0, x) \tag{3.16}$$

The ReLU is recommended as default choice for feedforward networks as it usually works better than sigmoidal functions and is fast to compute [86]; furthermore, it works comparably to its modifications [121].

3.3.2.4 Softmax function

Unlike activation functions mentioned previously, the *softmax* function is a function of the neuron's inputs and not of their weighted sum [51, 86]. It is often used as an output layer because its output can be interpreted as an estimate of probability. It takes a real valued vector and squashes its element in a such way that they are in range $[0, 1]$ and they add up to 1 [51, 86].

The *softmax* function is defined as [51]:

$$\operatorname{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}, i = 1, \ldots, n \tag{3.17}$$

3.3.3 Classification

The classification is usually done using a layer with the *softmax* activation function where the output is interpreted as a probability distribution over a discrete variable. If a single value instead of a set of estimated probabilities is needed, a label corresponding to maximum probability is returned [51, p. 183].

3.3.4 Optimization

The optimization (training) of a neural network is usually done using *backpropagation* and consists of three steps — *forward propagation*, *loss optimization*, and *error backpropagation with parameter update* [51]. The *forward propagation* computes the output of the NN for a given input, the *backpropagation* allows to compute the gradient of the optimized function and the actual update is done using a gradient descent method [51].

3.3.4.1 Loss function

Loss functions measure the quality of a particular parameter assignment after the forward pass [86]. The forward pass outputs class score for an input and the loss function measures how much is the score consistent with the ground truth [86]. The loss function is the function that is optimized during training. Loss functions commonly used are *crossentropy*, *mean squared error* (sometimes without the average — *sum of squared errors* (SSE) or called the L_2 loss [80, 86]), *hinge loss*, and others [26, 80].

3.3.4.2 Backpropagation

Backpropagation is a method for computing gradients based on iterative use of the *chain rule of differentiation* [51]. For $x \in \mathcal{R}$, $f(x) : \mathcal{R} \to \mathcal{R}$, $g(x) : \mathcal{R} \to \mathcal{R}$, $y = g(x)$, and $z = f(y) = f(g(x))$, then the chain rule is [51]:

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{3.18}$$

The vector case is more relevant for neural networks. Let $\mathbf{x} \in \mathcal{R}^m$, $\mathbf{y} \in \mathcal{R}^n$, $z \in \mathcal{R}$, $g(\mathbf{x}) : \mathcal{R}^m \to \mathcal{R}^n$, $f(\mathbf{y}) : \mathcal{R}^n \to \mathcal{R}$, $\mathbf{y} = g(\mathbf{x})$, and $z = f(\mathbf{y})$, then the chain rule is [51] is:

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{3.19}$$

The backpropagation in NNs is usually with tensors — the tensor notation of the chain rule is [51]:

$$\boldsymbol{\nabla}_{\mathbf{X}} z = \sum_j \left(\boldsymbol{\nabla}_{\mathbf{X}} Y_j\right) \frac{\partial z}{\partial Y_j} \tag{3.20}$$

where $\mathbf{X}$, $\mathbf{Y}$ are tensors, $\mathbf{Y} = g(\mathbf{X})$ and $z = f(\mathbf{Y})$ and j is a tuple of indices [51].

The backpropagation uses the chain rule iteratively as the NNs can be thought of as a compound function. The backpropagation is usually done for computational graph which describes the compound function — each node applies a function to the set of arguments that are the values of previous nodes [51]. For more detailed overview about backpropagation and computational graphs see [51, 86].

3.3.4.3 Gradient descent

The gradient descent is probably the most used approach for optimizing deep NNs [86]. It is an algorithm for finding local optima that works by going down in the loss function landscape [86]. The vanilla version computes the gradient at current point and takes a step of fixed length (*learning rate*) in the direction of the steepest descend. Since NNs usually have many parameters and are trained using many examples, the gradient descent is usually replaced by *online gradient descent* [86] which takes into account only a single example, or by *minibatch gradient descent* using small batchs of examples per weight update. Both modifications are often called *stochastic gradient descent* (SGD) even though SGD is only a different name for the *online gradient descent* [86]. Other modification of gradient descent includes *momentum* that allows to overcome some local optima and speeds up the convergence in case of long, narrow passes in the loss function landscape. The *momentum methods* takes into account the history of the descent — the negative gradient only changes the velocity of the particle with respect to its momentum, the new position is then calculated using only the current position and the velocity [86]. Similar concept is the *Nesterov momentum* which works exactly the same as classical momentum except that the gradient is computed at the position after momentum update [86]. Also several different methods for setting dynamically the *learning rate* were proposed as a single learning rate rarely works well. These methods range from single formula for the learning rate (e.g. *step decay*, *exponential decay* or $\frac{1}{t}$ *decay* [86]) to tuning the learning rate differently for individual parameters based o training history (e.g. *adagrad* [38], *Adadelta* [202], *RMSprop* [183], *ESGD* [33], *Nadam* [37], or *Adam* and *AdaMax*[89]). See [51] for more details about gradient descent including second order methods.

3.3.4.4 Hyperparameters

Hyperparameters are the parameters that influence learning of NNs and are not trained during learning of the network. The most common hyperparameters for optimization are the *(initial) learning rate*, *learning rate schedule*, *regularization strength* and *batch size* [16, 51, 86]. When discussing hyperparameters, some authors only consider the hyperparameters for the optimization (e.g. [125]) while others consider even the architecture to be a hyperparameter [16]. For analysis of individual parameters see [16, 121].

The *learning rate* parameter is sometimes difficult to set correctly while being tremendously important for the learning [51] — some optimization techniques mitigate this issue by automatically setting the learning rate (e.g. optimizers *adagrad* [38] or *Adam* [89], viz section 3.3.4.3) but they do so at the cost of introducing other hyperparameters [51].

The *batch size* determines how many samples will be used in *mini-batch optimization* for one gradient update [51]. While it influences the learning, it also significantly affects the training time as it determines how many operations will be done at a time. For Image pattern recognition, the recommended values are 128 and 256 as shown in [121]. The batch size, however, is often limited by the memory of the GPU used for training of the network and thus it is recommended to set the batch size as large as possible and reduce the learning rate proportionally to the batch size [121].

The hyperparameters do not have to be set manually but often a parameter search is used — the common approaches include *grid search* and *random search* [51, p. 436]. More complex techniques of hyperparameters optimization are sometimes used [51] but they are often impractical due to their computation complexity.

3.4 Network types

Over the years, many different neural networks architectures and approaches were proposed (not all are always considered to be a NN) — to name just a fraction: *perceptron* [156], *RBF* networs [17], *Hopfield* networks [66], *Boltzmann Machines* [62], *Restricted Boltzmann Machines* [171], *Auto–Encoders* (AE) [15], *Sparse Auto–Encoders* (SAE) [151], *Variational Auto–Encoders* (VAE) [90], *Denoising Auto–Encoders* [186], *Deep Belief networks* (DBN) [12], *convolutional neural networks* CNNs [42, 101] (viz. Ch. 3), *Deconvolutional networks* [204], *Deep convolutional inverse graphics networks* (DCIGN) [97], *Generative Adversarial Networks* [50], *Recurrent neural networks* (RNNs) [40], *LSTM networks* (networks with *Long short-term memory* units) [64], *GRU networks* (networks with *gated recurrent units*) [28], *Neural Turing Machines* (NTM) [54, 201], *bidirectional recurrent neural networks* (BRNNs) [161], *Echo State networks* [79], *Extreme Learning Machines* (ELMs), *Liquid State Machines* [112], *Recombinator networks* [65], and *Self-organizing maps* (SOM) [93]. This short list mentions only several main types of networks (some are subsets of others) and each of the named approaches contains many finer improvements in subsequent works. The NNs are a very general field with very different applications. A quite good visual comparison of most of the listed architectures is presented in [184]. This work focuses only on a very small subset of networks — *convolutional neural networks* for image pattern recognition.

3.5 Used Architectures

3.5.1 VGG16 and VGG19

The VGG16 and VGG19 are well known architectures presented first in [170]. They are very often used because they have very simple architecture while still being able to have impressive performance. As mentioned in section 3.2, the VGG network consists only of 3×3 convolutional layers and 2×2 max–polling layers with three densely connected top layers as shown in fig. 3.6. VGG networks are used not only for classifications but also for

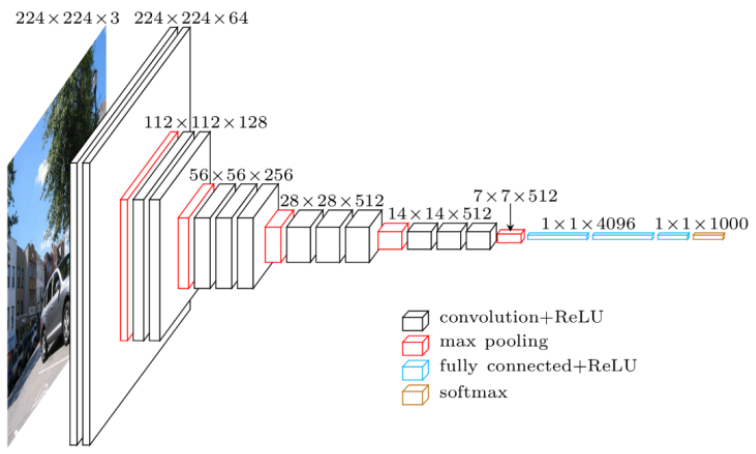


Figure 3.6: A visualization of the VGG16 architecture. Note the simplicity when compared to the Inception architecture (section 3.5.2) — only three types of layers are used in pyramidal settings. Reused from [30].

segmentation where similar but deconvolutional network is connected on top of the VGG [132].

3.5.2 Inception v3

Another used architecture is Inception v3 which is an improved version of the original *GoogLeNet* with inception units (viz fig. 3.1) [179] that was proposed in [180]. The Inception architecture allows for very good results on the ILSVRC2012 ImageNet challenge [158] with Top-5 error of only 3.58% (an ensemble of networks) [180] while being much less computationally costly than the VGG architecture [180]. The Inception v3 contains also different inception units compared to [179] as the 5×5 convolutions were replaced by a mini–network of two layers of 3×3 convolutions — this led to a lower number of parameters per inception unit and this saving allowed an increase in filter-bank sizes [180]. Furthermore, some of the convolutions were replaced by a mini-networks of asymmetrical convolutions, e.g. a 3×3 convolution was replaced by 3×1 convolution followed by a 1×3 convolution.

3.5.3 Xception

The *Xception* architecture is inspired by the Inception v3 where the individual inception units were replaced by depthwise separable convolutions [27]. The author interprets his architecture as equivalent to an inception unit with "a maximally large number of towers" [27] and shows that while both architectures behave similarly on the ImageNet dataset (Xception is slightly better than Inception v3 — top-1 accuracy 79.0% vs 78.2%), the Xception is significantly better on a very large dataset with 350 million images and 17,000

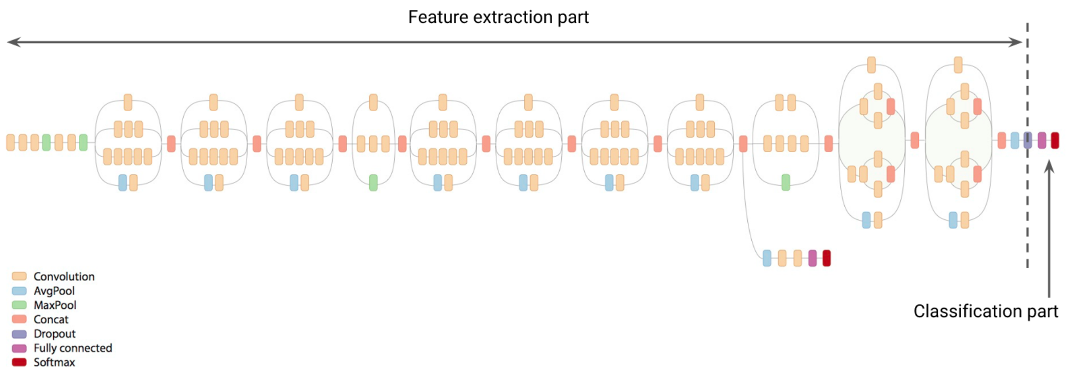


Figure 3.7: Visualization of the Inception v3 architecture. The Inception v3 has slightly different inception units than the GoogLeNet (the original inception unit shown in fig. 3.1). Reused from [75].

classes (4.3% relative improvement over the Inception v3) while having similar number of parameters.

3.5.4 ResNet Family

It is a family of residual networks ranging from moderately deep networks with 18 layers [57] through networks with 152 layers [57] to networks with 200 or even 1001 layers [59]. The ResNet ensemble won the *ILSVRC 2015* contest (viz section 3.2) and it was regularly used as part of ensembles in the top entries in the *ILSVRC 2016* contest [159]. The core idea of residual networks are skip connections (shortcuts) that carry the information to the output while skipping some layers — the skipped layers thus learn residual mapping instead of the whole mapping [57], the ResNet with 34 layers is visualized in fig. 3.2. A single ResNet branch with a skip connection is shown in fig. 3.8. The 50 layer version of ResNet was used in this thesis as the main network architecture, see section 5.2.

The ResNet has started a boom of residual networks and ResNet-like architectures as a simple addition of residual connections allowed learning of very deep networks that were before untrainable [106]. A theoretical analysis of residual networks is provided in [106] where it was explained why only the shortcuts of depth 2 works best as empirically found in [57] — authors of [106] shows that deeper shortcuts have the Hessian at the *zero* initial point to be a *zero matrix* which makes it a high-order stationary point that is hard to escape [3, 106], and the Hessian's condition number at the *zero* initial point for shortcuts of depth one grows unboundedly for deep architectures [106], the shortcuts of depth two, on the other hand, works very well because the *zero* initial point for such networks is a *strict saddle point* [44, 106] and such points can be easily escaped from using either second order method or stochastic first order method such as stochastic gradient descent [44].

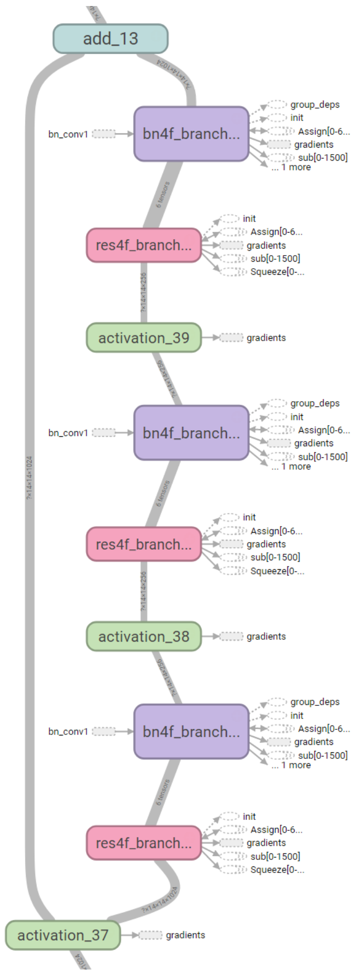


Figure 3.8: Visualization of ResNet branch with shortcut connection using TensorBoard. More about TensorBoard in section 6.2.1.

Chapter 4

Data

This chapter presents the created proof-of-concept datasets that are one of the main contributions of this thesis and also discusses the methodology of dataset creation:

- dataset description (section 4.1 and section 4.1.2)
- obtaining the imagery (section 4.3)
- annotating the imagery (section 4.4)
- sampling the data (section 4.5)
- data augmentation (section 4.6)

The available remote sensing image datasets, as described in Ch. 2 Sec. 2.3 and summarized in Tab. 2.1, are not sufficient for learning CNNs from scratch. They are usually very small (especially in comparison to ImageNet [158] that has 1.2 million images) or they they have a very small number of classes (e.g. the Brazilian Coffee Scenes dataset has only two classes) which makes them less interesting. The classifiers in the field were limited for a long time only to rule based descriptors and hand crafted features [23] as there were not available datasets that would allow learning more complicated features from the data. The few datasets that have provided sufficient amount of data (e.g. *Brazilian Coffee Scenes* dataset [145] or *SAT-4* and *SAT-5* datasets [8]) have only a few classes — while low number of classes is not harmful for learning, these classes tends to be rather general and most of the difficulties of classification comes from images that, to some extent, contain both classes rather from complex classes as in the ImageNet.

Furthermore, the CNNs trained on the ImageNet can be often used in other tasks through transfer learning as they have likely learned good features that are necessary for classifying complex classes (the features can be also applied to different spectral bands than they was trained on, for example, a VGG network pretrained using RGB data (ImageNet) was used for classificaton of CIR data in [167]). Furthermore, CNNs learned on the mentioned datasets are much less likely to be useful for transfer learning. While two datasets solving the weaknesses have emerged recently (*AID* [194] in 2016, *NWPU-RESISC45* [23] concurrently with this thesis in 2017), both do not have a constant scale. While this might be useful for some tasks, it is not suitable for many remote sensing applications because the scale is usually known in such applications [167]. We believe that several datasets with different scales but constant scale within the dataset are a better approach — especially as there are objects whose definition is based on their scale.

4.1 Created datasets

Several different datasets were created for the purpose of this thesis. The reasons for creation of several datasets instead of one are listed in section 4.1.2 while the individual classes present in the dataset are shortly introduced in section 4.1.1. Examples of individual classes are shown in fig. 4.1 and chapter A.

4.1.1 Individual classes

Total of 44 classes (viz table 4.1 and fig. 4.1) were selected but not all of them are of the same quality — some of the classes are very common and can be easily distinguished (e.g. *road*, *building*, or *forest*), others are very rare and also almost impossible to distinguish from each other (e.g. *power_ oil*, *power_ coal*, and *power_ nuclear*). The goal of this section is to provide a short description of the selected classes.

airport
: The class *airport* describes "a place from which flight operations take place" [138]. The objects vary a lot within this class — it contains objects from local airfields to international airports. This object is often the *Airport reference point* [176]. Imagery examples shown in Fig. fig. 4.1a and fig. A.1.

baseball
: This class describes baseball pitches, stadiums and places where baseball is played [138]. Imagery examples shown in fig. 4.1b and Fig. fig. A.2.

bridge
: This class describe any bridge — it describes a situation "when a road, railway, path canal, pipeline or similar is leading over a bridge" [138]. Imagery examples shown in fig. 4.1c and fig. A.3.

building
: Any buildings are members of this class. Imagery examples shown in fig. 4.1d and fig. A.4.

castle
: A castle is considered to be a "residential and often fortified buildings often from medieval times" [138]. The class *fort* describes more modern forts. Imagery examples shown in fig. 4.1e and fig. A.5.

cemetery
A class describing a (larger) place for burials [138]. It often appears near churches. Imagery examples shown in fig. 4.1f and fig. A.6.

coastline
The border of land and sea. It is defined as "the mean high water spring line between the sea and land" [138]. Imagery examples shown in fig. 4.1i and fig. A.9.

cooling_tower
A cooling tower is large and distinctive tower in the landscape. It describes towers that "use evaporation of water to reject heat from processes such as cooling the circulating water used in oil refineries, chemical plants, power plants" [138]. Imagery examples shown in fig. 4.1j and fig. A.10.

dam
A dam is any "wall built across a river or stream to block and regulate the flow of the river" [138]. It can vary from very small objects built on streams to large dams spanning hundreds of meters. It also describes walls of a pond. Imagery examples shown in fig. 4.1k and fig. A.11.

farmland
A farmland is "an area of farmland used for tillage and pasture (animals, crops, vegetables, flowers, fruit growing)" [138]. It differs from the class *meadow* by its purpose and plants growing on it. Imagery examples shown in fig. 4.1l and fig. A.12.

forest
A forest is a "managed woodland or woodland plantation" [138]. However, there is a discord between OSM users about how forests and woodlands should be tagged. The class definition used here in this thesis is the most prevalent within Europe. If obtaining data for other regions, a change of definitions might be necessary, viz [135] for details. Imagery examples shown in fig. 4.1m and fig. A.13.

fort This class describes historical military fortresses that are more modern than *castle* [138]. Imagery examples shown in fig. 4.1n and fig. A.14.

fuel_station
A fuel station is "retail-type facility where motor vehicles can be refueled" [138]. This class describes only fuel stations that are used by road vehicles. Imagery examples shown in fig. 4.1o and fig. A.15.

golf This class describe a golf course where golf can be played. Imagery examples shown in fig. 4.1p and fig. A.16.

greenhouse
This class describe the "land used for growing plants in greenhouses" [138]. A greenhouse is "a glass or plastic covered building used to grow plants" [138]. Imagery examples shown in fig. 4.1q and fig. A.17.

harbour
This class defines the "area of water where ships, boats, and barges can moor" [136]. Unlike the class *marina*, it should be used for larger boats, however, OSM users do not always make this distinction. Imagery examples shown in fig. 4.1r and fig. A.18.

helipad
A helipad is "a landing area or platform for helicopters" [138]. Imagery examples shown in fig. 4.1s and fig. A.19.

highway
This class describes the most important roads within a country. It represents objects defined as `motorway` and `trunk` in the OSM [138]. Imagery examples shown in fig. 4.1t and fig. A.20.

chimney
A chimney is "a tall distinctive vertical conduit for venting hot gases or smoke, normally found near power stations or large factories" [138]. Imagery examples shown in fig. 4.1g and fig. A.7.

church
This class describes buildings that were built as churches [138], these object do not have be used as churches anymore. Such buildings are often distinctive by their architecture or location. Imagery examples shown in fig. 4.1h and fig. A.8.

lake A lake is "a body of relatively still fresh or salt water of considerable size, localized in a basin that is surrounded by land" [137]. Imagery examples shown in fig. 4.1u and fig. A.21.

landfill
A landfill is "a place where waste is dumped" [138]. It is often distinctive in the landscape. Imagery examples shown in fig. 4.1v and fig. A.22.

marina
A marina is "a facility for mooring leisure yachts and motor boats" [138]. It is often mistaken for *harbour* that should be used for facility for larger ships. Imagery examples shown in fig. 4.1w and fig. A.23.

meadow
A meadow is "an area of land primarily vegetated by grass and other non-woody plants, usually mowed for making hay" [138]. Imagery examples shown in fig. 4.1x and fig. A.24.

orchard
This class is used for "intentional planting of trees or shrubs maintained for food production" [138]. Imagery examples shown in fig. 4.1y and fig. A.25.

parking
This class represents car parks [138]. However, sometimes it is used for very small places for parking of several cars only. Imagery examples shown in fig. 4.1z and fig. A.26.

pipeline
This class describes major pipelines [138]. The use of the class is limited only to pipelines that are overground. Imagery examples shown in fig. 4.1aa and fig. A.27.

power_coal
This is a small class describing places where electricity is produced using coal generators. Imagery examples shown in fig. 4.1ab and fig. A.28.

power_hydro
This is a class describing places where electricity is produced using water. Imagery examples shown in fig. 4.1ac and fig. A.29.

power_nuclear
This is a small class describing places where electricity is produced using nuclear reactors. Imagery examples shown in fig. 4.1ad and fig. A.30.

power_oil
This is a small class describing places where electricity is produced using oil generators. Imagery examples shown in fig. 4.1ae and fig. A.31.

power_solar
This is a class describing places where energy is produced using solar energy. This class describes both large solar parks and smaller generators placed on roofs of buildings. Imagery examples shown in fig. 4.1af and fig. A.32.

power_wind
This is a small class describing places where electricity is produced using wind turbines. Due to the recent support of renewable resources, many new wind turbines have been built recently — not all of them are captured by the aerial imagery available in Google Maps. Imagery examples shown in fig. 4.1ag and fig. A.33.

quarry
This class describes places for surface mineral extraction [138]. Imagery examples shown in fig. 4.1ah and fig. A.34.

raceway
A raceway is "a course or track for (motor) racing" [138]. Imagery examples shown in fig. 4.1ai and fig. A.35.

rail This class represents rails for "full sized passenger or freight trains in the standard gauge for the country or state" [138]. Imagery examples shown in fig. 4.1aj and fig. A.36.

river

This class describes rivers that are large enough to be defined as an area in the OSM [138] as narrow streams are often almost indistinguishable in the aerial imagery. Imagery examples shown in fig. 4.1ak and fig. A.37.

road

This class describes any road that is not in the class *highway*. Imagery examples shown in fig. 4.1al and fig. A.38.

runway

A runway is "a strip of land kept clear and set aside for airplanes to take off from and land on" [138]. It can be just a strip of grass. Imagery examples shown in fig. 4.1am and fig. A.39.

snow_fence

A snow fence is "a solid fence-like structure built across steep slopes to reduce risk and severity of (snow) avalanches" [138]. However, it is the only class from the used classes that appears mostly in mountains which can lead to a detection of mountains instead of snow fences. Imagery examples shown in fig. 4.1an and fig. A.40.

stadium

A stadium is "a major sports facility with substantial tiered seating" [138]. As created datasets are target on Europe, the overlap with class *baseball* is minimal. For other regions one of the class should not be used or the labeling datasets should be used. Imagery examples shown in section 4.1.1 and fig. A.41.

taxiway

This class represents paths "on an airport connecting runways with ramps, hangars, terminals and other facilities" [138]. Imagery examples shown in fig. 4.1ap and fig. A.42.

train_station

This class describes train station buildings. This class should be used mostly for the labeling tasks as there is an overlap with classes *building* and *rail*. Imagery examples shown in fig. 4.1aq and fig. A.43.

vineyard

A vineyard is "a piece of land where grapes are grown" [138]. Imagery examples shown in fig. 4.1ar and fig. A.44.

4.1.2 Division into several datasets

One common problem with remote sensing datasets is that a single image can contain several objects and thus it is hard to classify the image as a single class. Authors of individual datasets usually solve the problem by careful selection of the images and possible

name	**20cls**	**total**	**train**	**test**	**val.**	**key**	**value**	**radius**	**area**	**weighted**
baseball	✓	999	799	100	100	sport	baseball	—	✓	✗
bridge	✓	1350	1080	135	135	man_made	bridge	—	✗	✗
building	✓	1350	1080	135	135	building	—	1000	✓	✓
cemetery	✓	1347	1077	135	135	landuse	cemetery	—	✓	✓
coastline	✓	1349	1079	135	135	natural	coastline	—	✗	✗
farmland	✓	1350	1080	135	135	landuse	farmland	1000	✓	✓
forest	✓	1350	1080	135	135	landuse	forest	1000	✓	✓
golf	✓	1350	1080	135	135	leisure	golf_course	—	✓	✓
harbour	✓	810	647	82	81	harbour	—	—	✓	✗
highway	✓	1350	1080	135	135	highway	motorway,trunk	1000	✗	✗
marina	✓	1350	1080	135	135	leisure	marina	—	✓	✗
meadow	✓	1350	1080	135	135	landuse	meadow	1000	✓	✓
power_solar	✓	1350	1080	135	135	generator:source	solar	—	✓	✗
power_wind	✓	1336	1067	135	134	generator:source	wind	—	✗	✗
quarry	✓	1350	1080	135	135	landuse	quarry	—	✓	✓
rail	✓	1350	1080	135	135	railway	rail	10000	✗	✗
river	✓	1350	1080	135	135	waterway	riverbank	—	✗	✗
road	✓	1350	1080	135	135	highway	primary, secondary, ↲ tertiary, residential	1000	✗	✗
runway	✓	1350	1080	135	135	aeroway	stadium	—	✗	✗
stadium	✓	1350	1080	135	135	leisure	stadium	—	✓	✓
20cls total		**25551**	**20869**	**2342**	**2340**					
airport	✗	1350	1080	135	135	aeroway	aerodrome	—	✓	✗
castle	✗	1350	1080	135	135	historic	castle	—	✓	✗
cooling_tower	✗	332	265	34	33	tower:type	cooling	—	✓	✗
dam	✗	1350	1080	135	135	waterway	dam	—	✓	✗
fort	✗	1178	940	119	119	historic	fort	—	✓	✗
fuel_station	✗	1350	1080	135	135	amenity	fuel	—	✗	✗
greenhouse	✗	1350	1080	135	135	landuse	greenhouse_horticulture	—	✓	✓
helipad	✗	1350	1080	135	135	aeroway	helipad	—	✗	✗
chimney	✗	1350	1080	135	135	man_made	chimney	—	✗	✗
church	✗	1350	1080	135	135	building	church	—	✓	✓
lake	✗	1350	1080	135	135	water	lake	—	✓	✓
landfill	✗	1350	1080	135	135	landuse	landfill	—	✓	✓
orchard	✗	1350	1080	135	135	landuse	orchard	—	✓	✗
parking	✗	1350	1080	135	135	amenity	parking	8000	✓	✗
pipeline	✗	1350	1080	135	135	man_made location l	pipeline overground, overhead	—	✗	✗
power_coal	✗	283	226	29	28	generator:source	coal	—	✓	✗
power_hydro	✗	1347	1078	134	135	generator:source	hydro	—	✓	✗
power_nuclear	✗	132	105	14	13	generator:source	nuclear	—	✓	✗
power_oil	✗	67	53	7	7	generator:source	oil	—	✓	✗
raceway	✗	1350	1080	135	135	highway	raceway	—	✗	✗
snow_fence	✗	216	172	22	22	man_made	snow_fence	—	✗	✗
taxiway	✗	1350	1080	135	135	aeroway	taxiway	—	✗	✗
train_station	✗	1350	1080	135	135	building	train_station	—	✓	✓
vineyard	✗	1350	1080	135	135	landuse	vineyard	—	✓	✓
full total		**52596**	**42068**	**5266**	**5262**					

Table 4.1: Summary of used classes. The column **20cls** shows whether a class is selected for the smaller dataset with 20 classes. The columns **train**, **test**, and **val.** represents the number of samples of given class in the training, testing and validation data respectively. The columns **key** and **values** describes the tags that describe the given class in the OSM. The column **radius** describes the radius in which objects are obtained when using the *random search sampling* (section 4.5.1), if empty, the data for given class were obtained using the *bulk download sampling* (section 4.5.2). Columns **area** and **weighted** describes whether the object is considered to be an area and whether area weighted sampling should be used (viz section 4.5.3).

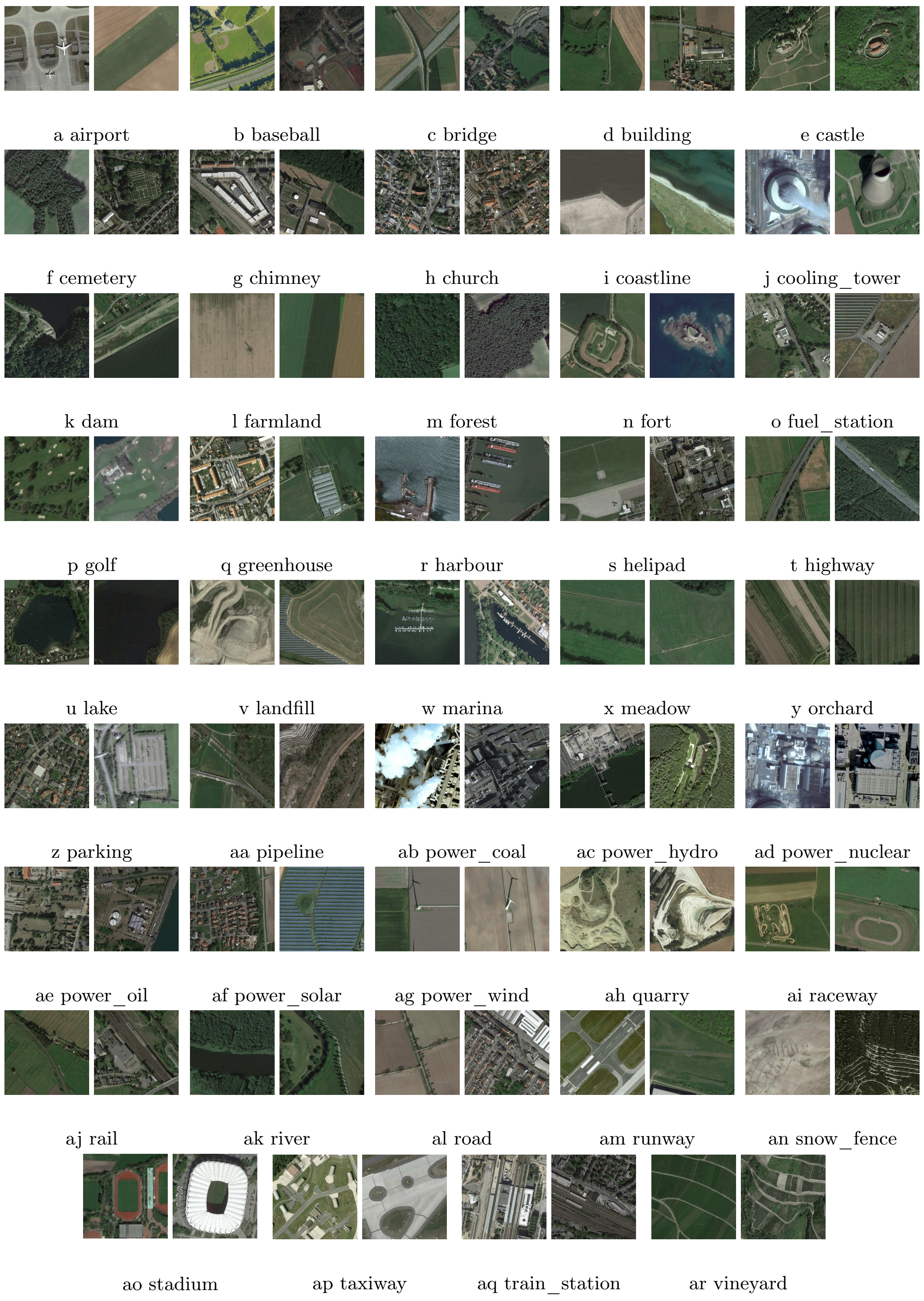


Figure 4.1: Examples of imagery for all 44 classes.

classes in a such way that the ambiguity is removed (e.g. *UC Merced Land Use* dataset [197] have only very specific classes). Another problem is that sometimes the classes are hierarchical and not mutually exclusive, e.g. a class *stadium* or *church* are subsets of more general class *building*. This work takes two different approaches to the latter problem — one dataset (*classification*) creates a hierarchy and more specific class has priority over a general class (e.g. if a object is both *building* and *stadium*, its final class is *stadium*) in order to make the classes well defined and disjoint, the other dataset (*labeling*) allows an image to belong to several classes, for example, if both *stadium* and *road* are present, both labels should be assigned. Both types of datasets consist of a single set of imagery, however, they differ in the annotations. The *classification* datasets assign the class to the patch based on the object in the center of the patch, while the *labeling* datasets assign the labels by object present anywhere in the patch.

The datasets also differ by the included classes as a smaller dataset was created for use in the classification tasks as it minimize hierarchical overlaps of classes (e.g the class *taxiway* is usually part of class *airport*). Thus the datasets were also divided by two different set of classes — one with 20 classes and one with all 44 classes. The classes that were selected for the dataset with 20 classes are shown in table 4.1 where they have checkmark in the column **20 cls**. These classes were selected for minimal overlaps in their definitions and thus this dataset is suitable for the strict classification task where only **one** class is assigned per image. The 20 classes in the smaller datasets also are more clear and better distinguished from each other and the distinction between them is usually clear — unlike in the full dataset (named *44cls*) where classes such as *castle* and *fort* are harder to classify even for a human subject.

This two divisions lead to a total number of 5 datasets where the tasks are distinguished by a prefix *LAB* for the labeling task and *CLS* for the classification task — *CLS20*, *LAB20*,*CLS44*, and *LAB44*. The fifth dataset is used only for the labeling task and while it contains imagery from the full dataset, the annotations contain only the 20 classes as the smaller datasets. The fifth dataset is denominated as *LAB20S44*. The fifth dataset is suitable for labeling as it contains the set of better defined classes and it still utilizes all the obtained imagery which results in higher number of samples. It however, deepens the imbalance between classes as, for example, many of the classes not included in the set of 20 classes are subclasses of a class *building* and thus the class *building* will be much more frequent in the dataset *LAB20S44* compared to the dataset *LAB20*.

4.2 Obtaining the dataset

The whole pipeline for obtaining the dataset is shown in fig. 4.2 — the parts that use the Google Static Maps API for obtaining the aerial data (viz section 4.3) and OSM (viz section 4.4) are highlighted. The process of sampling places with given class is described in section 4.5, the process for downloading the imagery and annotations is shortly described in chapter 6, and the data augmentation in section 4.6.

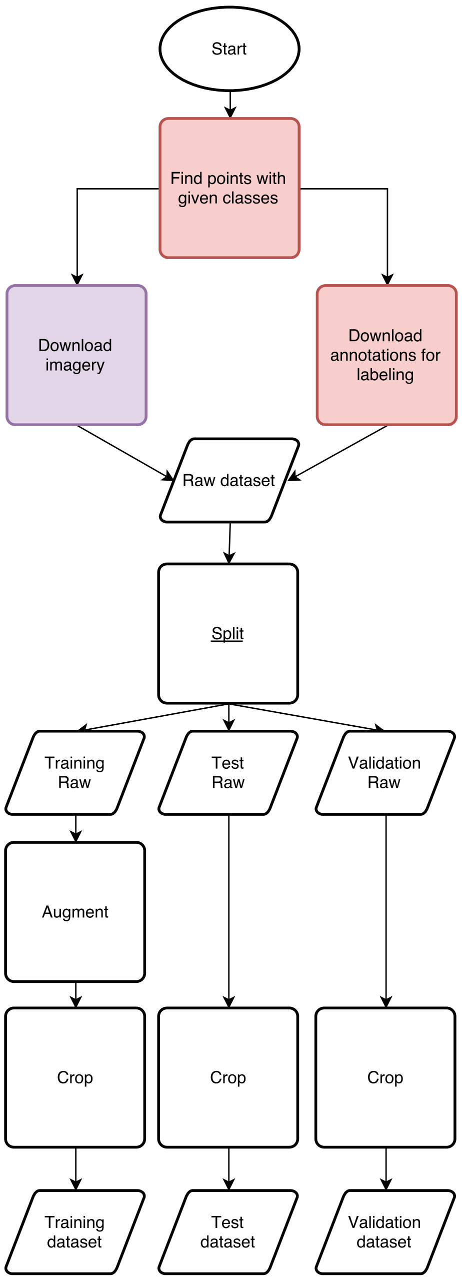


Figure 4.2: A diagram describing the dataset creation pipeline. Parts in red use the OSM [141], while parts in purple use the Google Static Maps API[52].

4.3 Aerial data

The proposed dataset consists of aerial imagery with three spectral bands (red, green, and blue). The data were obtained from Google Maps [52] that provide aerial map images stitched together from various providers. Use of images from several sources is quite common for constructing remote sensing datasets — for example, Google Earth was used in creation of datasets in [23, 194, 195, 205, 206]. Furthermore, Hu et al. analyze the suitability of Google Earth imagery for land use mapping in their work *Exploring the Use of Google Earth Imagery and Object-Based Methods in Land Use/Cover Mapping* [69] and they have come to conclusion that such imagery is suitable for application where the red, green, and blue spectral bands are sufficient. Thus that even though the Google Earth images are post-processed from the original image sources, the imagery is still suitable for many remote sensing tasks [69, 194].

Another reason for the use of Google Maps imagery is that the requirement of this thesis was to use open imagery source and the [52] provides free access to aerial/satellite imagery all over the world unlike other open sources that are often limited to small areas (e.g. [134]). Furthermore, the Google Maps API [52] allows easy access to the imagery by simply supplying the coordinates and thus alleviate many problems with image alignment. The Google Earth/Maps imagery is a multi-source imagery and thus have also quite a high variability across the world. This is beneficial for the purposes of the dataset creation as it contains images with different quality, post-processing, and, to some extent, different weather conditions thus allowing to learn better generalizing features.

4.3.1 Weaknesses

The main weakness of Google Maps imagery is the interval of updates — the imagery tends to be a bit outdated. While this is not a significant problem in most cases as the physical features changes only slowly, the problem might be accentuated for several classes (e.g. *power_wind*). This problem is not present when manually annotating the images but only when external annotations are used because such annotations will not be typically synchronized with the aerial data.

4.4 Annotating the data using OpenStreetMap

Most of the datasets described in this work are manually annotated by expert in remote sensing interpration (e.g. [5, 23, 145, 194, 195, 197, 205]), however, this process is quite costly for larger datatets and it might be one of the reasons why most of the datasets have modest size. Furthermore, the labels are usually known as the classes are most of the times physical features that are noted in maps (e.g. roads, rivers, buildings, etc.), thus it makes sense to use a map for annotating the images and avoid duplicate work. The use of such annotations might, however, lead to a problem with gap between times of acquisition of annotations and the aerial imagery, see section 4.3.1. While there are many maps in the world, the OpenStreetMap (OSM) [141] is probably most suitable because is both relatively up-to-date and with open access. An example of rendered OSM is shown in fig. 4.3.

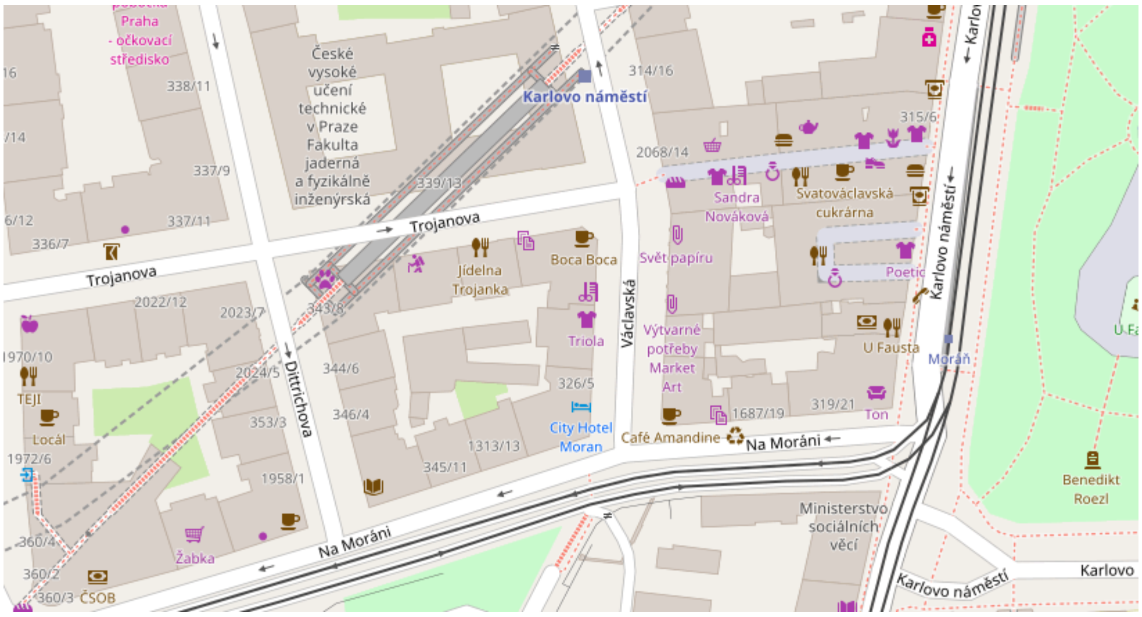


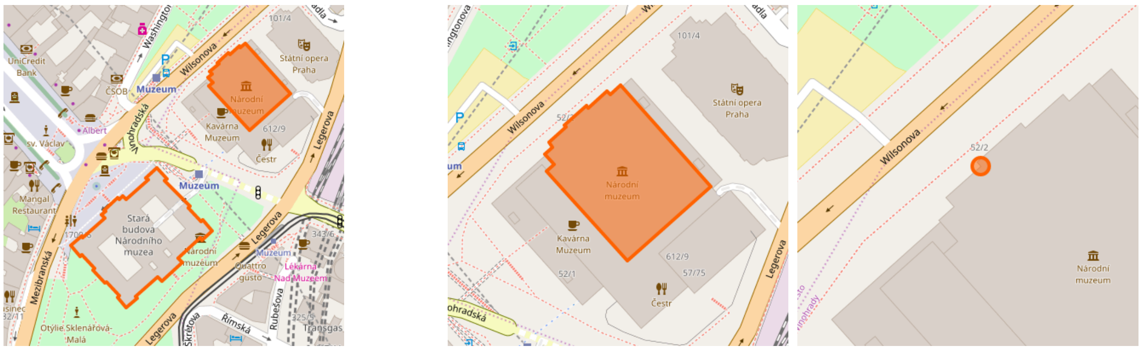
Figure 4.3: An example of a rendered OSM. Obtained using [140].

This thesis uses OpenStreetMap [141] for both labeling of obtained imagery and selecting the location at which the imagery should be obtained. In order to have more or less balanced dataset, the list of classes is created first and then based on this list, a set of images is obtained for each class (More details in section 4.5). Then the area covered by the image is determined and OpenStreetMap is again used for annotating the image.

The OSM consists of three different types of objects — nodes, ways, and relations that can have tags consisting of a key and possibly a value to them [138], example shown in fig. 4.4. A way is a sequence of nodes and it is used for describing both lines (e.g. road) or closed polygons (e.g. building), while relation is a set of nodes, ways, and other relations (e.g. stadium building with open area in the middle might consist of an outer way and an inner way).

4.4.1 Weaknesses of OpenStreetMap

The OpenStreetMap, however, suffers occasionally from several weaknesses which is hardly unexpected as it is created by many users with different levels of expertise. The most notable weakness is connected to the OSM's strength — the OSM depends heavily on its community and the quality and accuracy of a map is often proportional to the quality of the local community that creates it. The OSM contains regions of different quality — some are up-to-date and mapped with rich features and objects while others are outdated and contain deprecated tags.



a relation b way (closed) c node

Figure 4.4: Types of objects in the OSM. The *relation* can contain *ways*, *nodes* but also other *relations*. A *way* is an ordered sequence of *nodes* and can be either closed (a polygon) or open. Each object may contain tags. Imagery obtained using [140]

Another weakness is inconsistent map tagging and object annotation — the map consists of many different types of objects and each of them can have different tags and in spite of the effort of the OSM community, inconsistent tags occurs. For example, a tag `amenity=parking` is often assigned to a single node placed in the middle of the real parking area or the whole parking are can be traced with a polygon with the assigned tag — this results in problems with segmentation mask creation and, in our case, with area weighted sampling as a single node has no area and thus zero weight. Moreover, the OSM maps have also tags `parking:lane` (reserved for parking areas along roads) and `amenity=parking_space` (reserved for tagging a single parking spot within a parking area) and as the mapping is done by users with varying expertise, it sometimes happens that a user tags an object with a wrong tag.

Some tags also depend on subjective decision of individual users and such tags also are often inconsistent — for example, a tag `man_made=chimney` should be used only for mapping huge chimneys but it is often not the case. Furthermore, the OSM are still under development and new features and tags are being proposed and, occasionally, old ones are being deprecated. The implementation of these changes depends again on the activity of the local community and the old tags can survive in the map for a long time. For example, there was a confusion whether water reservoir should be tagged as `landuse=reservoir` or using a new way with two tags `natural=water` and `water=reservoir` [138]. The confusion might have resulted in inconsistent tagging. There are also many optional tags that could be very useful for dataset creation (e.g. tags `surface` or `height`) but their usage varies heavily and thus they are not reliable.

However, the OSM is very accurate for regions with active and large local community as it is often made by people who live in the area and know their surroundings very well. This, together with the richness of features, makes up well for the weaknesses mentioned above and thus the OSM are our choice as the source of annotations.

4.5 Sampling the Earth

Despite the coverage of the OpenStreetMap and the Google Map, this dataset covers mostly Europe only — and even then most of the samples are from Germany. The focus of the dataset was determined by several reasons. Since up-to-date and as exact as possible labels are needed, the Europe and especially Germany were selected as the target region as the OpenStreetMap has strong community there and is probably, by our subjective view, most accurate there. The OpenStreetMap community has come up with two main explanations why the OSM lacks in the U.S. — first, the OSM in the U.S. obtained a large import of map data from a database *Tiger* [19] in 2007 [192] which might have discouraged early users from contributing and second, the U.S. residents have open access to mapping project such as *The National Map* [177] which might have lowered the demand for OSM. The OSM lacks even more in the rest of the world even though the situation is getting slowly better.

Despite this bias towards Europe and Germany in particular, the imagery should come more or less uniformly sampled from the target region and not from just a particular subregion. The dataset should be, at least partially, balanced in order for the network to have enough of data of each classes to learn the individual classes well — if, for example, the patches were sampled randomly and unconditionally of the class, most of the data would be covered by farmlands and forests, occasionally with several buildings and roads and almost none of the complex classes would be present. A network would have then seen almost no samples of certain categories and would fail to learn those as the training would be steered by the preeminent classes. Furthermore, the most common classes (e.g. *farmland*, *forest*, *building*) are not very interesting as they can be quite easily classified using multi-spectral, but low resolution data (e.g. [13]). Moreover, if the goal was to learn the most common classes, the SAT-4 and SAT-6 datasets [8] could be used instead. Therefore the imagery of this dataset was sampled with respect to a given class to have the dataset at least partially balanced. Namely, 1350 images were attempted to obtain for each of the classes. It was not possible to obtain the full number of images for certain classes as they do not occur in such number in Europe (e.g. *cooling-tower* class).

Two different approaches were used for sampling imagery for different classes — a *random search* and *bulk download* — based on the frequency of occurrence of the sampled class. The two different approaches were used in order to reduce technical difficulties of the sampling. The *random search* was used for sampling very common classes (e.g. *forest*, *meadow*, *farmland*, *road*, . . .) and it was used in order to have great variety in the data and sample the Germany more or less uniformly. The *bulk download* approach was used for classes that occur very rarely and would not be efficiently found by the *random search* or, in most cases, the classes were so infrequent that the whole region contained less then desired number of images of the class (e.g. classes *airport*, *baseball* or *cooling-tower*).

4.5.1 Random search sampling

This approach was used for sampling of very common classes the within target region (Germany). First, a point inside the region was generated at random from the uniform distribution, then a local search around the point was made and set S_l of all objects of given

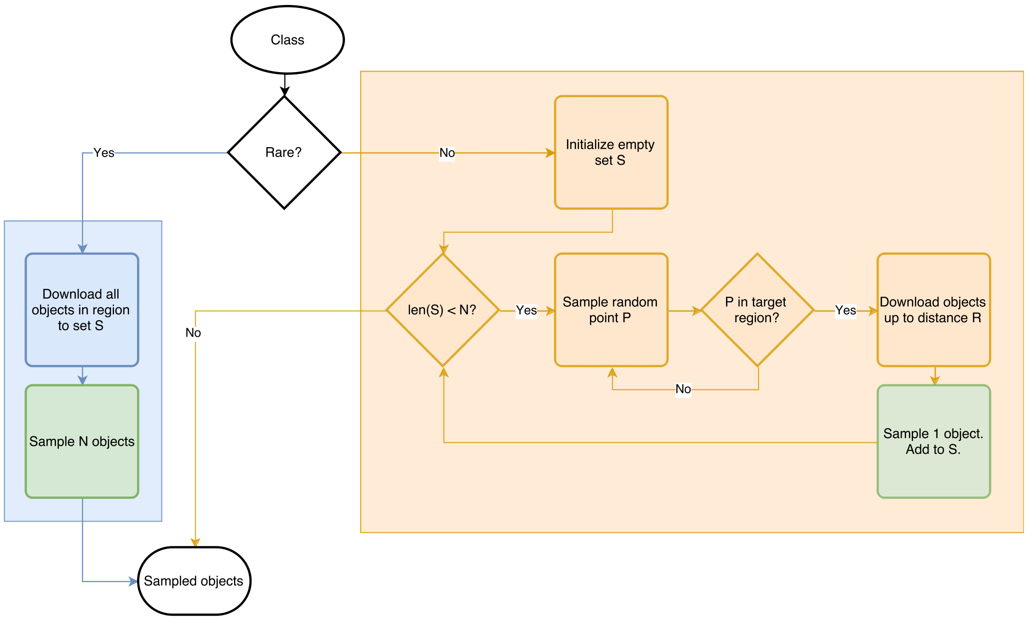


Figure 4.5: A diagram describing the sampling proces for a given class. The blue part describes the bulk sampling, the orange describes the random search sampling. The green shows the sampling method that is common for both parts and that is shown in fig. 4.6.

class within a certain radius r was obtained. An object was randomly selected from the set S of obtained objects and its coordinates were returned. The radius r is set differently for each class as it is based on the frequency of occurrences of the class — this was motivated by technical reasons as large radius r is costly as too many objects have to obtained while too small radius r is costly as it results in many empty sets. The random object from the set was obtained in a same way as in the *bulk download*, see section 4.5.3. This approach could not be used for the more rare classes as it would take a very long time to find them if there was only a few of them and, more importantly, if the region contained less objects of the given class than desired, it would be impossible to determine whether all appropriate objects were obtained. Thus it would be impossible to stop the sampling. The random search sampling procedure is summarized in Listing 4.1.

Listing 4.1: Pseudocode of the random sampling for a single class

```
samples = []
while len(samples) < N:
    pt = random_point()
    if pt in boundary:
        objects = obtain_object_in_radius(pt,r)
        sample = weighted_sample(objects,1)
        samples.add(sample)
```

4.5.2 Bulk download sampling

This approach was used for the more rare classes — a set S_g of all objects of given class was obtained at once and then sampled (see section 4.5.3 for details of the sampling). While theoretically this approach is suitable for all classes and not only for the rare ones, practical difficulties limited its use — the most common classes were computationally very costly to obtain and also the set of objects would not fit in the memory.

4.5.3 Sampling of obtained objects

The objects were sampled from the sets of objects — in case of the *random search* sampling, a single object was sampled from the local set S_l and in case of the *bulk download* sampling, a set of n object was obtained from the set of all objects S_g, where n is the desired number of images of the particular class. The sampling was done on the OSM nodes and thus it was different than would be the uniform sampling of the Earth as there was bias for certain objects. If the object of the class was defined using only *nodes* or *ways* (viz section 4.4), the sampling was done uniformly over the set of the nodes. This sampling was used, for example, for *road*, *rail*, *coastline*, and *river* classes. This type of sampling exhibits bias for objects that consists of more nodes than the others — e.g. if user A of the OSM created roads consisting of many nodes, while user B tagged the roads using much fewer nodes, the sampling would be biased towards the roads of user A. Despite the theoretical bias, we have observed no significant variations in complexity of objects of most classes.

Two different samplings were used if the object could be interpreted as an area. First, if the object was defined as area only for some instances but as a node for other instances, the objects defined as an area would be treated as if they were a single node (e.g. *airport*, *baseball*, *harbour*, *cooling-tower*). If most of the objects of the given class were defined as an area, then the rest of the objects was discarded and the objects defined as an area were sampled with respect to their size (such classes have ✓in column **weighted** in table 4.1). If the size had not been taken into account, then the sampling would have been strongly biased towards small objects that are more numerous but that do not represent most of the class occurrence on the Earth — for example, a huge forest defined as a single area would have much lower probability of being selected than that one of several small groves would be selected even though a point sampled uniformly would have higher probability to end up in the larger forest than in one of the small groves. In order to obtain higher variety in the dataset, the objects were not, however, sampled proportionally to their areas but rather to square root of their areas — if some of the classes were sampled proportionally to their areas, we would end up with images that are completely covered with the object and such images are not as useful for learning. After several experiments, the weighting by square root of the area seems as the best trade-off between reducing the bias to very small objects and keeping variety of the dataset. While the weighting could be elaborated and analyzed in more detail, the decision is not as crucial for the dataset and the analysis was left for future works.

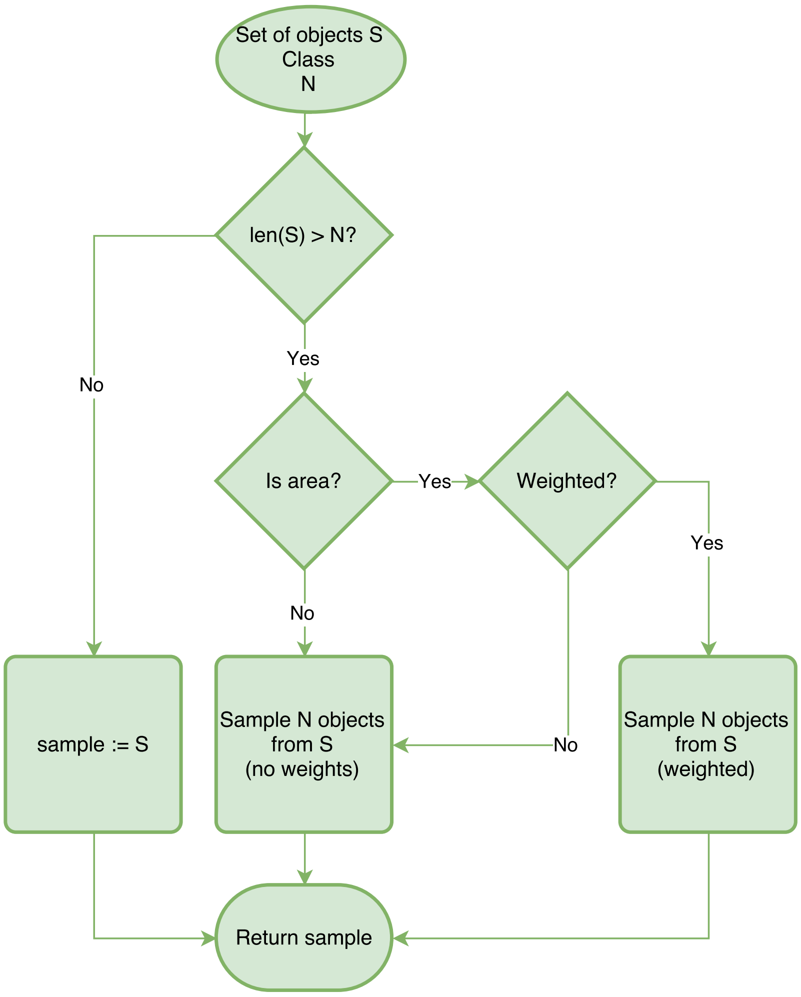


Figure 4.6: A diagram describing the sampling of obtained objects. The inputs are a set of objects S, a class that is being sampled (to know whether the sampled objects represent areas and whether weighted sampling should be used), and a number N of objects to be sampled.

4.6 Augmentation

Image augmentation is a process of enlarging the dataset by creating new artificial samples [51]. More samples usually helps with generalization [51], thus augmentation is used when there is not enough data and a NN is overfitting. The augmentation modifies the data and creates a new samples that somewhat resembles the samples within the dataset. While this migh be problematic for certain tasks, it is relatively simple for image recognition tasks [51, p. 240] — the input image is usually transformed by, for example, shifts by several pixels or rotations.

The augmentation used in [26] is limited to rotations, shifts, flips, shear transformation and zoom transformation. While shear and zoom transformation are often useful for general image recognition, they are not used in this thesis as we deal with aerial images that were rectified and have constant scale. Two different augmentation configurations were tested — *simple* and *complex*.

4.6.1 Used transforms

4.6.1.1 Shift

Shift can be horizontal or vertical, this transform shifts the input image by random number of pixels. The unknown pixels after shift are usually either filled with a default value or by the nearest known pixel. The shifting range might be defined relative to the size of the image — e.g. 5% vertical shift of image with width 400 px shifts the image by 20 px to right.

4.6.1.2 Flips

Flips can also be horizontal or vertical, flipping transforms flip the image over the horizontal or vertical axis.

4.6.1.3 Rotation

Rotation rotates the image by a random number of degrees. The unknown images are filled in same way as for shifting transforms.

4.6.1.4 Sharpening

The implementation from [84] was used for sharpening. The sharpening process is controlled by two parameters α and lightness λ. The sharpening operation is defined as convolving matrix $\mathbf{S}$ with the image matrix (per channel). The matrix $\mathbf{S}$ is defined as [84]:

$$\mathbf{S} = (1-\alpha)\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix} + \alpha\begin{bmatrix}-1 & -1 & -1\\ -1 & 8+\lambda & -1\\ -1 & -1 & -1\end{bmatrix} \tag{4.1}$$

4.6.1.5 Blurring

A Gaussian blur was used for blurring, the only parameter being the standard deviation σ of the filter. See [84] for the implementation.

4.6.1.6 Brightening

Two different brightening modes were used, adding a random value to each of the pixel (additive mode) or multiplying the values of each pixel by a random constant (multiplicative mode). The output value is clipped to interval $[0, 255]$ and discretized.

4.6.1.7 Contrast Normalization

This changes the contrast of the image and it is parametrized by a single parameter α. The new pixel intensity (per channel) $\overline{p_{i,j}}$ before clipping is defined as:

$$\overline{p_{i,j}} = \alpha \cdot (p_{i,j} - 128) + 128 \tag{4.2}$$

where $p_{i,j}$ is the original intensity. The values are then cropped to interval $[0, 255]$ and discretized.

4.6.2 Simple augmentation

The *simple* augmentation contains only simple transforms that allows it to be realtime, thus every time an image is used in training, it gets augmented differently. The *simple augmentation* is summarized in the table 4.2.

transform	**range**	**probability**
horizontal shift	$-5\% - 5\%$	1
vertical shift	$-5\% - 5\%$	1
rotation	$-20° - 20°$	1
horizontal flip	—	0.5
vertical flip	—	0.5

Table 4.2: Transforms in the *simple augmentation.* The column **probability** represents the probability with which the augmentation will be applied to an image.

4.6.3 Complex augmentation

The *complex augmentation* includes several computationally more costly transforms, thus the training images were augmented before training with a factor 10, i.e. a single image have been augmented into 10 new images. This augmentation is used because the obtained aerial data are from multiple sources and thus the individual images differ substantially in colors and sharpness. The *complex augmentation* was designed so the output images mimic the variation in the data as visually close as possible. The *complex augmentation* includes contrast and brightness transforms, rotations, flips and also sharpness modifications. Furthermore, the contrast and brightness transforms were applied also in per channel form in order to simulate the color variations in the data. This augmentation was made to be a part of preprocessing pipeline and was run on the raw images obtained from the Google Maps before they were cropped to the required size. Thus no method for filling the unknown pixels after rotation and shifts was required as the margin between the raw images and cropped images was sufficient to cover all rotations (if smaller images were used, the unknown pixels would be filled by the nearest known pixels).

Summary of used all transforms in the *complex augmentation* is provided in table 4.3.

transform	**range**	**probability**	**per channel**
horizontal flip	—	0.5	✗
vertical flip	—	0.5	✗
sharpen	α: 0 – 0.8 λ: 0.75 – 1.3	0.4	✗
blur	0.25 – 1.5	0.4	✗
brighten (additive)	-15 – 15	0.6	✗
brighten (multiplicative)	0.75 – 1.25	0.5	✗
contrast normalization	0.8 – 1.2	0.5	✗
contrast normalization	0.9 – 1.1	0.6	✓

Table 4.3: Transforms in the *complex augmentation.* The column **probability** represents the probability with which the augmentation will be applied to an image. Column **per channel** describes whether a new random value for the parameters was drawn for each channel.

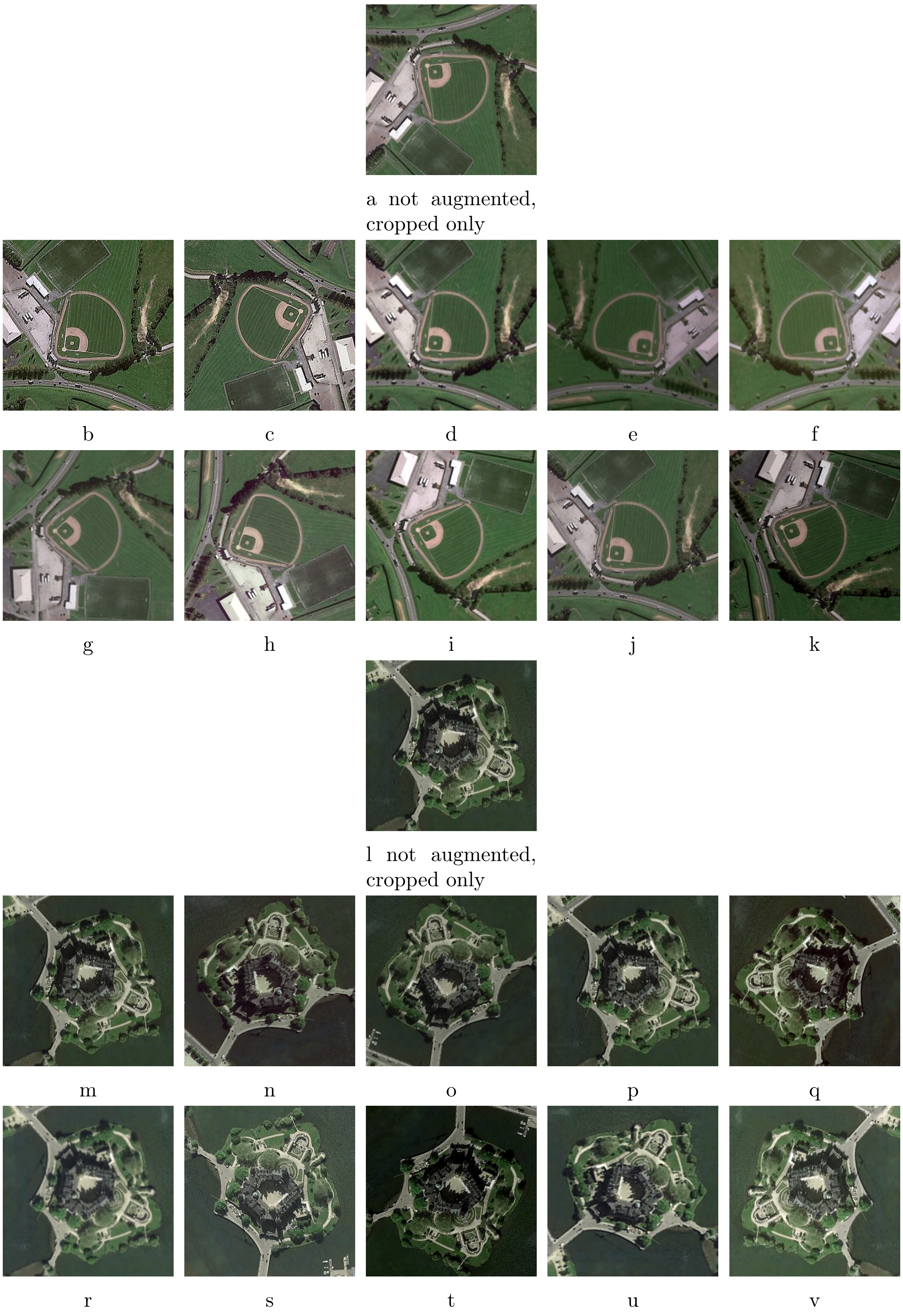


Figure 4.7: Examples of augmented imagery.

Chapter 5

Methods

This thesis is divided into three distinctive parts — creation of datasets, using them for training deep neural networks and evaluation. While the dataset creation was described in previous chapter, the training and evaluation is described here.

5.1 Transfer learning

A several architectures with good performance were selected for the transfer learning: VGG16 [170], VGG19[170], ResNet50 [57], Inception 3 [180], and Xception [27] (See section 3.5 for details). Transfer learning is a process where a network trained using some data is adapted for use on different data [51] —the pretrained network is usually used as feature extractor and only the top layers are trained again for classification on the new dataset. The transfer learning is based on the premise that the first layers usually learn only general filters that can be used if the dataset are similar enough (which is case for many image datasets).

The individual pretrained networks were used without their top layers for production of so called *bottleneck features* [2] which were then input to several different top layers configurations that were then trained, i.e. the top layers in the original network were replaced by different layers and the rest of the layers were frozen so no weights update occurred during training. For computational gains, the bottleneck features were precomputed on the datasets and only the shallow networks consisting only from the top layers were subsequently trained which is much computationally cheaper (at the cost of no online augmentation — however, the training data were augmented on the disk before the training to mitigate this issue, viz section 4.6).

The individual top layer configurations (section 5.1.1) and individual architectures (section 3.5) were then evaluated and the best architecture from the transfer learning was then used for learning from scratch using only the proposed dataset. The individual top layers configurations were used only in the transfer learning as the preceding layers were frozen, during the full learning, the default top layer configuration for the selected network was used as the gains by additional layers would be (if any) minimal. The transfer learning also provided a baseline for evaluation of the performance of the networks learned from scratch.

5.1.1 Top layer configurations

Several different top layer configuration were used — the configuration could either use *global average pooling* (GAP) or *global max pooling* (GMP) which were followed by output layer that used either *softmax function* for classification tasks or the *logistic sigmoid* function for the labeling task. Optionally, there could be an intermediary dense layer with 256 ReLU neurons. The configurations with the intermediary layers could also use a dropout of 0.5 to deal with overfitting. This resulted in 6 different top layer configuration for each of tasks as summarized in table 5.1.

name	**task**	**pooling**	**intermediary**	**dropout**	**output**
GAP-softmax	classification	GAP	✓	✗	softmax
GMP-softmax	classification	GMP	✓	✗	softmax
GAP-256dense-softmax	classification	GAP	✓	✗	softmax
GMP-256dense-softmax	classification	GMP	✓	✗	softmax
GAP-256dense-0.5drop-softmax	classification	GAP	✓	✓	softmax
GMP-256dense-0.5drop-softmax	classification	GMP	✓	✓	softmax
GAP-sigmoid	labeling	GAP	✓	✗	sigmoid
GMP-sigmoid	labeling	GMP	✓	✗	sigmoid
GAP-256dense-sigmoid	labeling	GAP	✓	✗	sigmoid
GMP-256dense-sigmoid	labeling	GMP	✓	✗	sigmoid
GAP-256dense-0.5drop-sigmoid	labeling	GAP	✓	✓	sigmoid
GMP-256dense-0.5drop-sigmoid	labeling	GMP	✓	✓	sigmoid

Table 5.1: Summary of used top layer configurations. The column **task** describes for which task the configuration was used. The column **pooling** describes which pooling was used, the column **intermediary** whether an intermediary dense layer with 256 ReLU neurons was used. The columns **dropout** shows whether a dropout of 0.5 has been applied to the intermediary layer and finally, the column **output** describes the used output layer.

5.1.2 Hyperparameters

The networks were trained using optimizer *Nadam* [37] with initial learning rate $l = 0.002$, $v = 0.999$,$\mu = 0.9$, $\epsilon = 1 \times 10^{-8}$, and schedule decay of 0.004 as recommended in [26] (the parameters have different names: $v = \beta_2$ and $\mu = \beta_1$ in [26]). The batch size for experiments with transfer learning was set to be 512 and the networks were trained for 100 runs over the whole dataset (epochs). The 100 epochs were more than sufficient to reach a stable performance over the validation set. The used loss function was *crossentropy* (named `categorical_crossentropy` in [26]) for the classification task and *binary crossentropy* for the labeling task.

5.2 Learning from scratch

The best architecture from the transfer learning was then selected for learning from scratch. The network with highest performance was the ResNet50 (viz section 7.1.1).The ResNet network with 50 layers was then learned from scratch on the proposed datasets in order to obtain a network capable of good classification performance. The network was trained using pre-augmented data (viz section 4.6.3).

5.2.0.1 Hyperparameters

The used optimizer was *Nadam* with the same parameters as in the transfer learning, the used batch size was 64 (the maximum size fitting into memory). The networks were trained until no significant improvement was observed (between 30 and 70 epochs) as the training was computationally very costly (up to several days on GeForce GTX TITAN X). The used loss functions were also the same as in the transfer learning.

5.3 *Accuracy metrics*

The *accuracy* was used as a metric of choice because it is the most often used metric in image classification. The used *accuracy* slightly differs for *classification* and for *labeling.* The accuracy for *classification* is defined as the fraction of correctly predicted classes to the number of predictions [51, p. 103]:

$$acc_{\text{CLS}} = \frac{\text{correct}}{\text{all}} \tag{5.1}$$

The *labeling* task actually consists of many binary classifications — each image can either be labeled or not. The published accuracy is actually an average accuracy for all the individual subtasks [26]:

$$acc_{\text{LAB}} = \frac{1}{n}\sum_{i=1}^{n} acc_{\text{CLS}_i} \tag{5.2}$$

where n is the number of possible labels and acc_{CLS_i} is the accuracy of a single classification subtask.

5.4 Evaluation of data augmentation

The data augmentation plays a crucial role in fighting the overfitting when the datasets are not huge [51]. The proposed augmentation pipeline (See section 4.6) was evaluated and compared to other augmentation techniques using the *CLS20* dataset. Four different augmentation configurations were tested:

1. only horizontal and vertical flipping augmentation (no degradation in the data)
2. default augmentation on the generated dataset (flips + random rotations $\pm 20°$ and random shifts $\pm 5\%$)
3. proposed complex augmentation on the raw data (viz section 4.6.3)
4. both complex and default augmentation

The hyperparameters were the same as in the section 5.2.0.1.

5.5 Evaluation of different activation functions

While the ResNet architecture used ReLU activations by default, many other different activations were proposed (See section 3.3.2). Use of different activation functions such as *exponential linear unit* (ELU), *leaky ReLU*, and *parametric ReLU* might bring accuracy gains [121]. Thus the ResNet50 implementation from [26] was modified to use also different activation functions. The *LAB20* dataset was used for learning, and the learning setup was same as in the section 5.2.0.1.

5.6 Visualization

To better understand the behavior of a network, several different techniques are usually used. The include *saliency maps* [168], *occlusion maps* [203], *class activation maps* (CAM) [208], and *gradCAM* [162]. This works used only *saliency maps* and *gradCAM* because the *occlusion maps* are basicaly equivalent to *gradCAM* while being computationally expensive and CAM requires suitable network architecture. The *gradCAM* produce similar results as the CAM method but is independent of network architecture [162]. While *saliency maps* were used for experiments, the *gradCAM* was chosen as the main method for visualization (viz figs. 7.8 to 7.10).

The *gradCAM* method is similar to backpropagation as it propagates the importance of individual neurons from a given layer back to the input to visualize the attention map. The *gradCAM* method can be used for class activation maps for any layer in the network. The usual choice is the last convolutional layer as these features are of the highest level [162] and the most interesting — such attention maps can be often used for object detection. The activation maps for earlier layers show more low-level features [162].

Chapter 6

Implementation

The work was implemented in python 2 and python 3 with *Ipython* [147], the neural networks were implemented using NN library *keras* [26] and computational framework *Tensorflow* [114]. Other used packages include *scipy* [83], *Scikit–learn* [144], *pandas* [118], *NumPy* [187], *matplotlib* [72], *seaborn* [189], *keras-vis* [94], *imgaug* [84], *overpy* [35], *gdal* [43], *shapely* [47].

The data were manipulated using scripts mostly based on code of *georasters* package [142].

The work can be divided into several parts that were implemented separately.

6.1 Dataset creation

The dataset creation (viz chapter 4) consists of two major parts — downloading the images and annotating the images. The *Data Obtainer* scripts first uses *Annotation Obtainer* to find the location for obtaining the imagery. The *Annotation Obtainer* takes a list of classes and number of desired samples per class and using process described in section 4.5 finds the coordinates of desired images. The *Annotation Obtainer* uses the Overpass API [139] that was designed for fast, read-only access to OSM. A python package *OverPy* [35] is used as wrapper for the OverpassAPI.

Then the *Data Obtainer* uses *Imagery Obtainer* which downloads the images for given locations and zoom level using the Google Static Maps API [52]. The images are returned as instances of (modified) *georaster*. A *georaster* is a class for manipulating geo-tagged images; the original code of *georasters* [142] was slightly modified for the purposes of this work (mostly minor modifications only).

Then another script is used for splitting the data into *training*, *testing*, *validation* datasets. The *training* data were then augmented using script for augmentation process described in section 4.6. This script uses the *imgaug* [84] python package for image augmentation.

6.2 Training neural networks

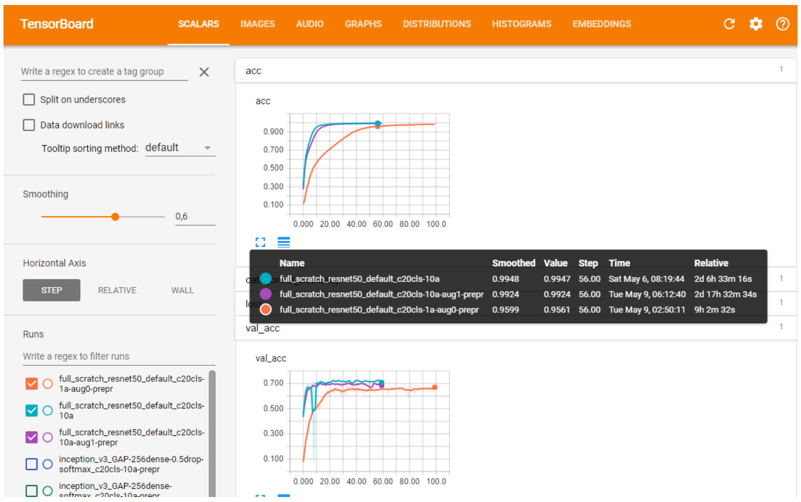
The training was implemented using *keras* [26] that is a library that utilize either *tensorflow* [114] or *theano* [182] backends for training neural networks. The backend *tensorflow* was used for the purposes of this thesis. *Tensorflow* is machine learning library that expresses computations as stateful dataflow graphs [114].

The *keras* provides high-level functional API for defining neural networks that allows to design even very complex architectures in several lines of code.

The *keras* also provides pre-trained networks on ImageNet, thus all the experiments with transfer learning (viz section 5.1) used networks provided by [26]. The modifications of ResNet50 using different activation functions (viz section 5.5) were based on code for ResNet50 provided by [26] that was slightly modified for the purposes of this thesis.

6.2.1 Other tools used during training

The *tensorflow* comes with TensorBoard [53] that allows to visualize the computational graphs and also the training process. The TensorBoard was utilized to monitor the progress of the more computationally expensive trainings to stop them if a plateau has been reached. An example of TensorBoard for monitoring training and validation accuracy is shown in fig. 6.1.



☐ Split on underscores
☐ Data download links
Tooltip sorting method: default

Smoothing: 0,6

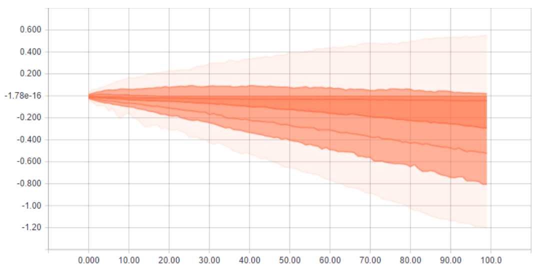
Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs
Write a regex to filter runs
✓ full_scratch_resnet50_default_c20cls-1a-aug0-prepr
✓ full_scratch_resnet50_default_c20cls-10a
✓ full_scratch_resnet50_default_c20cls-10a-aug1-prepr
☐ inception_v3_GAP-256dense-0.5drop-softmax_c20cls-10a-prepr
☐ inception_v3_GAP-256dense-softmax_c20cls-10a-prepr

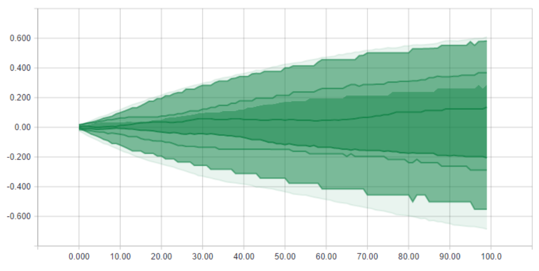
	Name	Smoothed	Value	Step	Time	Relative
●	full_scratch_resnet50_default_c20cls-10a	0.9948	0.9947	56.00	Sat May 6, 08:19:44	2d 6h 33m 16s
●	full_scratch_resnet50_default_c20cls-10a-aug1-prepr	0.9924	0.9924	56.00	Tue May 9, 06:12:40	2d 17h 32m 34s
●	full_scratch_resnet50_default_c20cls-1a-aug0-prepr	0.9599	0.9561	56.00	Tue May 9, 02:50:11	9h 2m 32s

Figure 6.1: An example of use of TensorBoard for monitoring the training process.

The TensorBoard also allows to visualize the distribution of weights for given layer, for example, fig. 6.2a shows the plot showing the development of the parameter distribution in time for biases in the dense layer with 256 neuron in transfer learning with ResNet50 feature extractor and shallow network consisting of GAP layer, 256 dense neurons with 0.5 dropout and softmax layer on the CLS20 dataset (viz section 5.1.1 for details). The fig. 6.2b shows the plot for biases in the softmax layer in transfer learning with Inception v3 feature extractor and shallow network consisting of GAP layer, 256 dense neurons and softmax layer on the CLS20 dataset.



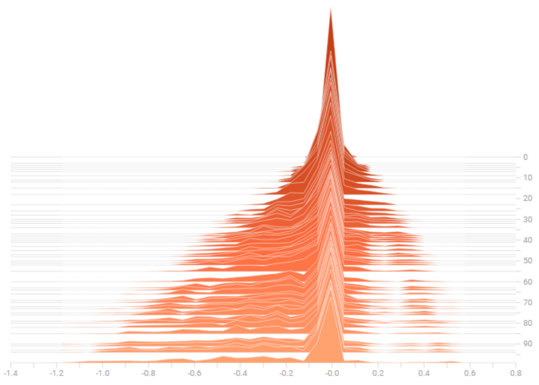
a bias for dense layer in ResNet50-GAP-256d-0.5drop-softmax



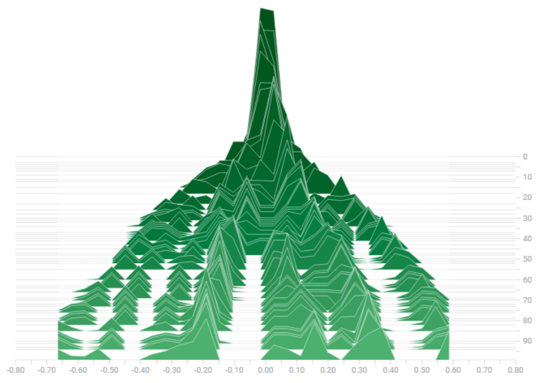
b bias for softmax layer in Inception v3-GAP-256d-softmax

Figure 6.2: Visualization of weight distribution using TensorBoard. The lines represent the percentiles: maximum, 93%, 84%, 69%, 50%, 31%, 16%, 7%, minimum from top to bottom [53]

Another possibility is to visualize the information using 3D surface plots as shown in fig. 6.3 for the same layers as described above.



a bias for dense layer in ResNet50-GAP-256d-0.5drop-softmax



b bias for softmax layer in Inception v3-GAP-256d-softmax

Figure 6.3: 3D isualization of weight distribution using TensorBoard.

It is possible to visualize a network architecture using the Tensorboard computational graph. This allows to understand even the complex architectures such as ResNets. For example, a Tensorboard visualization graph for a ResNet unit with shortcut is shown in fig. 3.8.

6.3 Evaluation

The evaluation used the library *scikit-learn* [144] for computing metrics and analyzing the results. The *gradCAM* attention maps and *saliency maps* were computed and visualized using the package *keras-vis*[94]. Other used packaged for evaluation and visualization include *matplotlib* [72], *seaborn* [189], and *pandas* [118].

Chapter 7

Results

This thesis consists of several experiments with the data in order to find the most suitable pipeline for the classification of the data. The first set of experiments consist of a transfer learning from the ImageNet [158] where several competition winning networks trained on the ImageNet were used for the classification of the proposed dataset. The second experiment focus on the data augmentation and its benefits. And finally, the best network architecture is selected and its performance evaluated.

7.1 Transfer learning

Several competition winning architectures (viz section 3.2) were selected for a transfer learning with weights learned on the ImageNet. The used networks are VGG 16 and VGG 19 [170], Inception v3 [180], ResNet50 [57] and also Xception [27] (not a winner of a competition but found to perform slightly better than the Inception v3 [27]). The goal of the transfer learning was not to produce the final classifier network but rather to obtain a baseline of the dataset and also to compare different architectures. The networks were used to produce a *bottleneck features* (viz section 5.1) and those were used as inputs to shallow networks. These shallow networks represent an equivalent to the top layer configurations stitched on top of the frozen architectures. According to preliminary experiments, the use of the *bottleneck features* sped up the learning at least by factor 10.

The general overview of the performance of all various combinations of top layer configurations and architecture is available in tables 7.1 to 7.4 for both training and validation accuracies on the CLS20 and LAB20 datasets. The results other datasets are available in section B.1.

While the situation is clear from overview tables tables 7.1 to 7.4 that shows the performance at the last epoch, plots of individual runs are available in sections B.2 and B.3. The ResNet50 pretrained network with GAP layers, dense layer with 256 neurons and 0.5 droput and output layer (dense with either softmax for classification or logistic sigmoid for labeling) performs generally the best. A further discussion of obtained results is available in section 7.1.1 (Comparison of architectures) and section 7.1.2 (Comparison of top layer configurations).

			Top layers					
			GAP soft	**GMP soft**	**GMP 256d soft**	**GAP 256d soft**	**GMP 256d 0.5drop soft**	**GAP 256d 0.5drop soft**
		Fig.	B.1d	B.2d	B.3d	B.4d	B.5d	B.6d
Architecture	Fig. B.7d	**ResNet50**	65.61	65.61	96.13	96.15	67.79	67.76
	Fig. B.8d	**VGG 16**	54.50	34.43	5.18	71.83	42.17	50.41
	Fig. B.9d	**VGG 19**	54.11	32.90	56.96	87.15	41.36	60.12
	Fig. B.10d	**Xception**	64.20	38.19	5.13	85.71	5.13	60.21
	Fig. B.11d	**Inception v3**	63.68	58.42	77.23	**98.66**	50.86	75.63

Table 7.1: The **training** accuracy of the classification task on *CLS2O* dataset. The accuracy is reported for the last epoch. The highest value for given architecture is in green, for the particular top layer in blue, and the overal highest value is in **bold**. The references to figures contains graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP soft	**GMP soft**	**GMP 256d soft**	**GAP 256d soft**	**GMP 256d 0.5drop soft**	**GAP 256d 0.5drop soft**
		Fig.	B.1d	B.2d	B.3d	B.4d	B.5d	B.6d
Architecture	Fig. B.7d	**ResNet50**	56.47	56.47	54.39	54.77	**59.75**	58.76
	Fig. B.8d	**VGG 16**	52.97	33.70	5.27	48.11	46.95	51.87
	Fig. B.9d	**VGG 19**	52.20	30.67	46.21	48.12	48.09	53.31
	Fig. B.10d	**Xception**	53.97	34.57	5.27	50.65	5.27	56.88
	Fig. B.11d	**Inception v3**	53.30	49.92	47.05	50.80	51.58	54.24

Table 7.2: The **validation** accuracy of the classification task on *CLS2O* dataset. The reported accuracy is the average over last 10 epochs. The highest value for given architecture is in green, for the particular top layer in blue, and the overal highest value is in **bold**.The references to figures contains graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP sig	**GMP sig**	**GMP 256d sig**	**GAP 256d sig**	**GMP 256d 0.5drop sig**	**GAP 256d 0.5drop sig**
		Fig.	B.1a	B.2a	B.3a	B.4a	B.5a	B.6a
Architecture	Fig. B.7a	**ResNet50**	88.23	88.23	92.10	**92.19**	88.98	89.04
	Fig. B.8a	**VGG 16**	87.25	84.76	87.92	89.95	86.58	88.07
	Fig. B.9a	**VGG 19**	87.23	84.82	87.96	89.96	86.68	88.05
	Fig. B.10a	**Xception**	88.07	86.65	88.12	89.81	86.11	88.19
	Fig. B.11a	**Inception v3**	88.06	87.51	89.52	91.25	87.76	88.72

Table 7.3: The **training** accuracy of the labeling task on *LAB20* dataset. The accuracy is reported for the last epoch. The highest value for given architecture is in green, for the particular top layer in blue, and the overal highest value is in **bold**. The references to figures contains graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP sig	**GMP sig**	**GMP 256d sig**	**GAP 256d sig**	**GMP 256d 0.5drop sig**	**GAP 256d 0.5drop sig**
		Fig.	B.1a	B.2a	B.3a	B.4a	B.5a	B.6a
Architecture	Fig. B.7a	**ResNet50**	87.73	87.73	86.16	86.09	**88.32**	88.23
	Fig. B.8a	**VGG 16**	87.39	85.02	87.22	86.74	87.06	87.88
	Fig. B.9a	**VGG 19**	87.32	84.73	87.27	86.54	87.18	87.94
	Fig. B.10a	**Xception**	87.62	86.15	87.47	87.16	86.60	88.09
	Fig. B.11a	**Inception v3**	87.72	87.15	86.74	85.82	87.78	87.98

Table 7.4: The **validation** accuracy of the labeling task on *LAB20* dataset. The reported accuracy is the average over last 10 epochs. The highest value for given architecture is in green, for the particular top layer in blue, and the overal highest value is in **bold**.The references to figures contains graphs of whole training for the given architecture or top layer configuration.

7.1.1 Comparing architectures

We can observe that the ResNet50 is generally performing the best — on all five datasets, it has reached the highest overall validation accuracy (the **bold** value in tables 7.1 to 7.4). Furthermore, ResNet50 was the best for all top layer configurations (blue values in tables 7.1 to 7.4) except for the configurations with dense layers without dropout where the networks with ResNet features tended to overfit a bit in comparison with the Xception. The Xception architecture seemed to produce more robust features than the other architectures as it had the best validation results with top layer configurations with dense layer without dropout. The networks with VGG feature extractors performed generally the worst — only the VGG19 was better than the Inception v3 on classification datasets when using the GMP pooling layer as both the Inception v3 and VGG16 failed in this case (accuracy $< 10\%$). The conclusion is clear — the ResNet50 pretrained network is the most suitable for transfer learning.

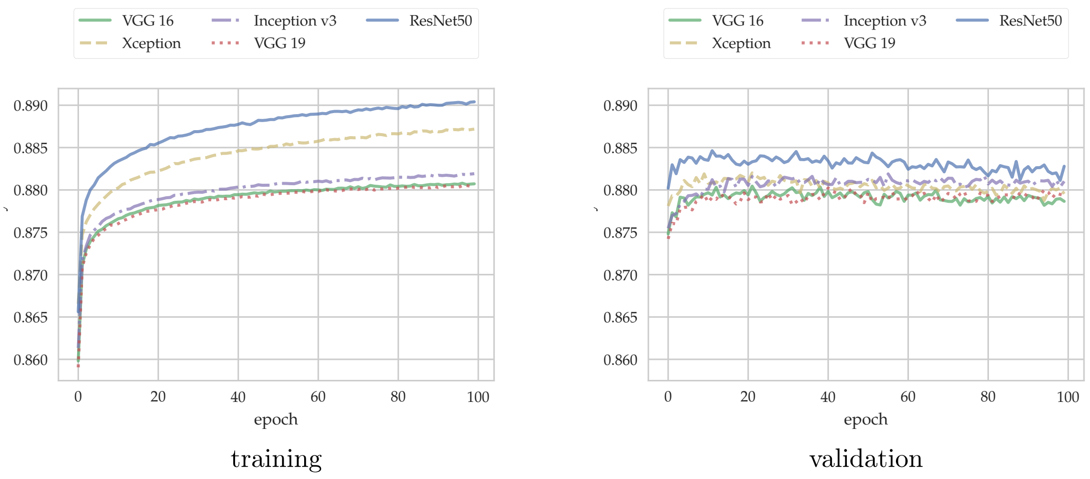


Figure 7.1: Accuracy of transfer learning using various architectures and the GAP-256dense-0.5drop top layer configuration on the *LAB20* dataset. See section B.2 for other top layer configurations and datasets.

7.1.2 Comparing top layers

Several different top layers were used for the classification after the features were extracted using the pretrained network. These top layers usually contain either a *global average pooling* (GAP) layer or a *global max pooling layer* (GMP). These layers behave similarly as the average pooling and max pooling layers but they are rather applied at the whole feature map instead of just to a neighbourhood of a cell. Note that sometimes their inclusion did not really make sense but they have been included anyway for the sake of completeness, e.g. fig. B.7, where both GMP and GAP behave almost identically —- the ResNet50 architecture contains already an average pooling layer and a repeated application of either

GAP or GMP does not change the output of the pretrained network. The usage of either GAP or GMP matters for other architectures though and it seems that the GMP leads to quite significantly lower accuracy than the GAP for both classification and labeling tasks.

The layer containing 256 densely connected neurons without any dropout overfits to the training set for all architectures and datasets. While not only the training error is significantly lower then the validation error, the validation error also decreases with more epochs of training. The networks that have a dense layer with 0.5 dropout also overfits in the sense that it performs significantly better on the training set than on the validation set but it does not exhibit significantly decreasing accuracy on the validation set which confirms that the dropout works well.

The combination of GAP with a dense layer with 256 neurons and 0.5 dropout seems to works best for all architectures (green values in tables 7.1 to 7.4) for transfer learning — the dense layer allows the network to learn quite complicated combinations of the extracted features while the dropout successfully limits the overfitting. Note that the final top layer was a dense layer with the *softmax* activation function for the classification task and the *sigmoid* activation function for the labeling task. The lesson learned is that when using transfer learning with frozen networks as feature extractors, it is suitable to add a dense layer with dropout to make the best from the extracted features.

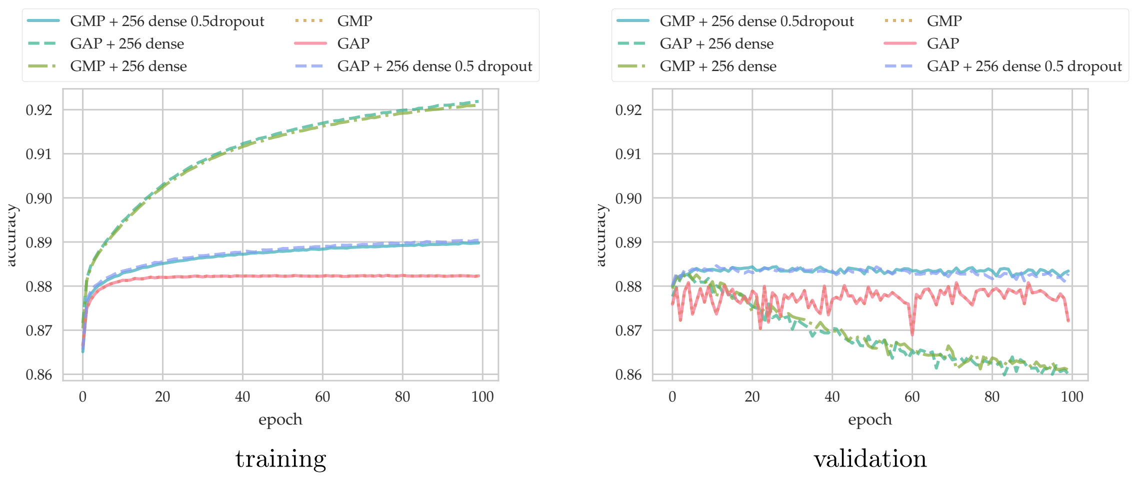


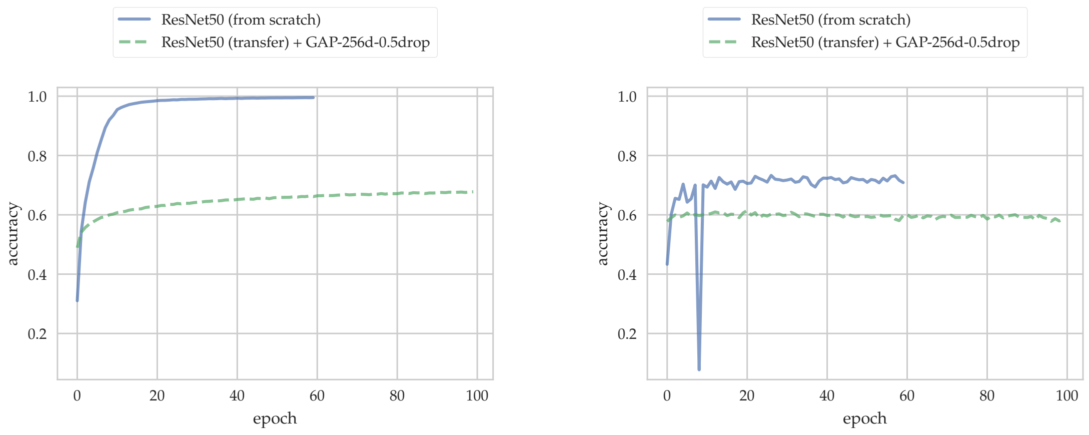
Figure 7.2: Accuracy of transfer learning using ResNet50 and various top layers configurations on the *LAB20* dataset. See section B.3 for other architectures and datasets.

7.2 Learning ResNet50 from scratch

Since the ResNet50 is state-of-art-network[1] and it performed the best in the transfer learning experiments (viz section 7.1), it was the architecture of choice for learning the weights

[1]it was regularly found in the winning neural networks ensembles in the prestigious competition ILSVRC 2016 [158, 159]

using only the proposed datasets from scratch. Unsurprisingly, the networks, that have learned all their weights and not only the top layers worked better than the transfer learning experiments as the dataset is of sufficient size to learn general features. Due to the computational complexity, the experiments with learning the full network has limited only to the smaller datasets (CLS20 and LAB20). A significant increase in accuracy compared to the transfer learning was observed as shown in fig. 7.3 for the classification task with *20cls* dataset.



training validation

Figure 7.3: Comparison between transfer learning and learning from scratch for ResNet50 on the CLS20 dataset. Note that the transfer learning network shown there is the best network from the transfer learning experiment.

	CLS20		LAB20		LAB44	
network	**train**	**val.**	**train**	**val.**	**train**	**val.**
ResNet50 (from scratch)	**98.43**	**71.80**	**99.92**	**91.59**	**99.91**	**95.10**
best from transfer	67.79	59.75	88.98	88.32	93.16	93.15

Table 7.5: Performance of training from scratch — comparison of the ResNet50 trained from scratch for given dataset and the best network from the transfer learning (viz section 7.1).

7.2.1 The benefit of augmentation

The classification task of the *20cls* dataset was also used for an investigation of the benefits of the *complex* data augmentation compared to a regular augmentation. Four different configurations were used — no augmentation, *complex* augmentation only, *simple* augmentation only and both augmentations.

The proposed complex augmentation pipeline (viz section 4.6) for aerial datasets performs the best — it even outperforms the combination of both *complex* and *simple* augmentations by a small margin. The *simple* augmentation performed better than the augmentation with flips only by a small margin — the random rotation seems crucial for the aerial data as most of the classes has no regular orientation in the landscape. However, the overall winner is the complex augmentation — it performed best on the validation dataset with a solid margin (2 pp). The only disadvantage of the complex augmentation is that it is quite costly. To partially alleviate the costs, this experiment used the augmentation as part of preprocessing the data, viz section 4.6.3. The training was then done without any live augmentation as the data that were fed to the optimizer were already augmented. The augmentation as part of preprocessing seems useful as the complex augmentation is costly and it could be possibly a bottleneck during training.

The complex augmentation comprises of several different transforms that are randomly applied to the input — besides random rotations and flipping, the complex augmentation also includes sharpening, blurring, additive and multiplicative brightness changes and also contrast transforms (channel wise) on the uncropped raw data (more details about the augmentation pipeline in section 4.6. This experiment shown that the complex augmentations is beneficial as it performs better than the simple by solid margins on the validation dataset. However, further analysis is needed to identify the individual gains from the individual transforms during the augmentation as it is possible that not all of the transforms are beneficial. The performance on the CLS20 dataset is shown in table 7.6.

augmentation	**training acc.** [%]	**validation acc** [%]
flips only	**98.51**	66.00
simple	98.05	68.19
complex	98.43	**71.80**
both	97.69	69.53

Table 7.6: Comparison of different augmentation methods (for description viz section 4.6). The complex augmentation performs the best, the combination of the simple and complex augmentation performs only slightly worse.

7.2.2 Different activations

The ResNet50 network performed very similarly for different activation functions (ReLU, ELU, PReLU) and its performance was almost identical for all three activation functions. The PReLU seems to learn faster on the training dataset which might be caused by the additional trainable parameter leading to earlier overfitting while the ELU overfits the slowest. The performance on the validation dataset is very similar. PReLU seems to be overfitting the most and the network with this activation had the deepest drop after initial peak. The results from [121] were not reproduced — the PReLU and ELU are comparable to ReLU and not superior. The difference in obtained results might be caused by using different networks and different datasets — the ResNet50 network is much deeper than the networks used in [121] and ResNets in general are able to learn complex classes well which might have undermined the influence of different activations.

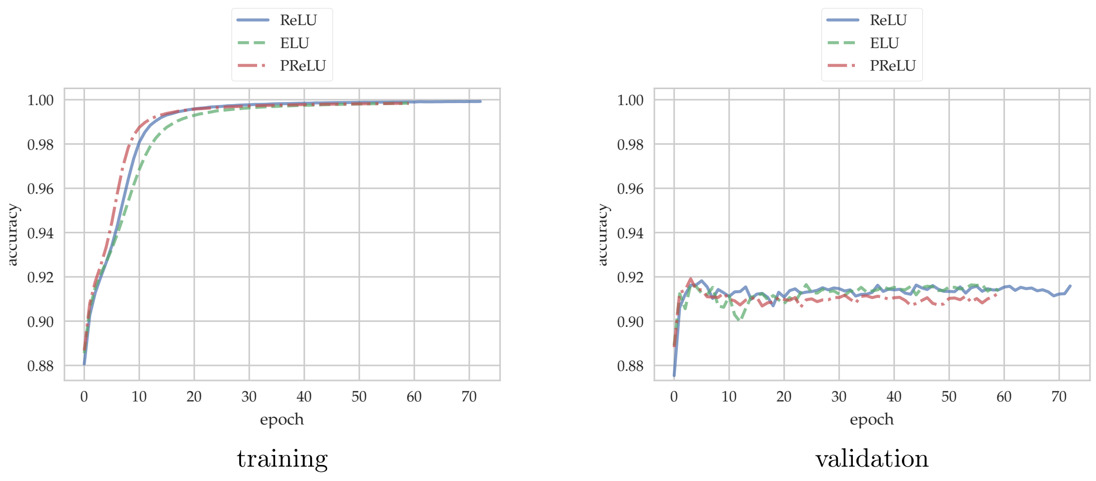


Figure 7.4: Testing different activation on the *LAB20* dataset.

7.2.3 Evaluation of the learned network

The network learned on the *CLS20* dataset was chosen for further analysis.

The confusion matrix is shown in fig. 7.7. We can observe that the network was able to learn most of the classes well even though there are some surprising patterns that are discussed bellow. Examples of nicely classified images are shown in fig. 7.8, examples of complete failure are shown in fig. 7.9. The fig. 7.10 contains examples that are wrongly classified when compared to the truth label but the mistakes are understandable and human expert would likely made the same mistakes as there is ambiguity to which class the image should belong (for the classification task, the labeling task solves this problem by allowing multiple classes per image).

While both gradCAM and saliency maps visualization were obtained for evaluating the performance (viz section 5.6), it was found out that the gradCAM visualization is superior and easier to understand to than the saliency map. An example of both saliency and gradCAM visualization is provided in fig. 7.5.

The provided gradCAM visualization are always for the layer after the last skip connection to visualize the high level features. However, to show that the gradCAM visualization depends on the used layer, the fig. 7.6 shows the visualization from the first convolutional layer to the same last layer as the other visualization. The 8 visualizations in the middle is randomly sampled as the number of layers is much higher and each can be visualized. The lower layers learn the low level features while the top layers learn higher level features.

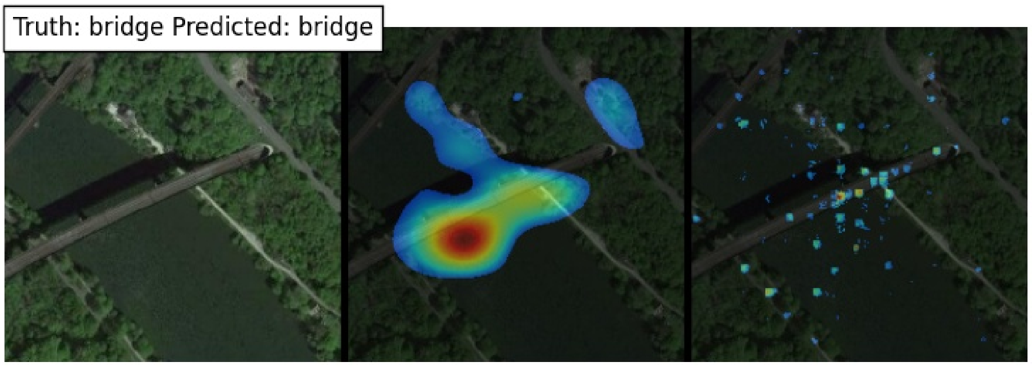


Figure 7.5: An example of a saliency map and a gradCAM visualization. The saliency map is inferior to the gradCAM as most of the spatial information is lost.

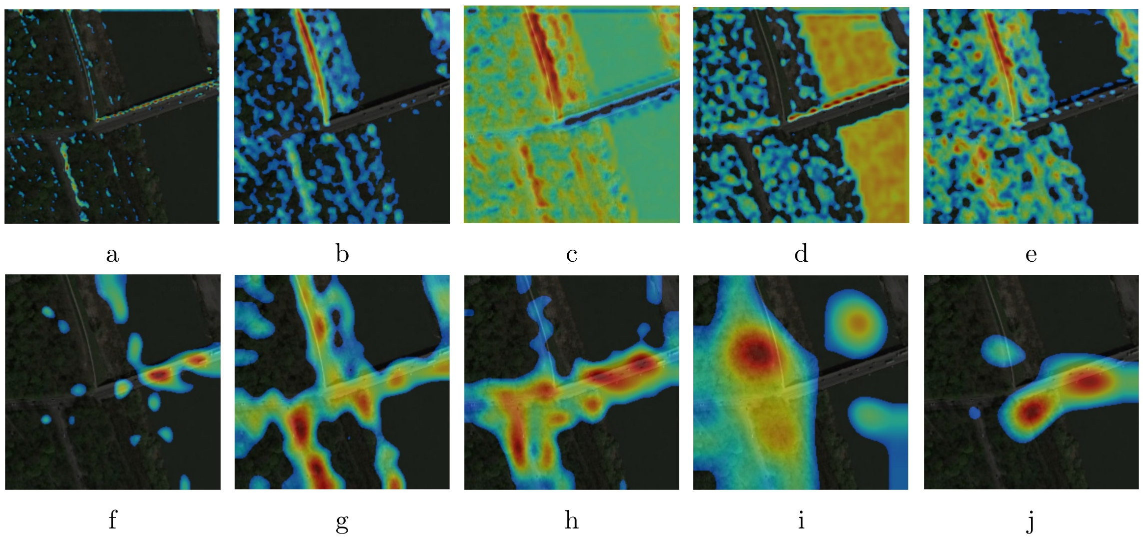
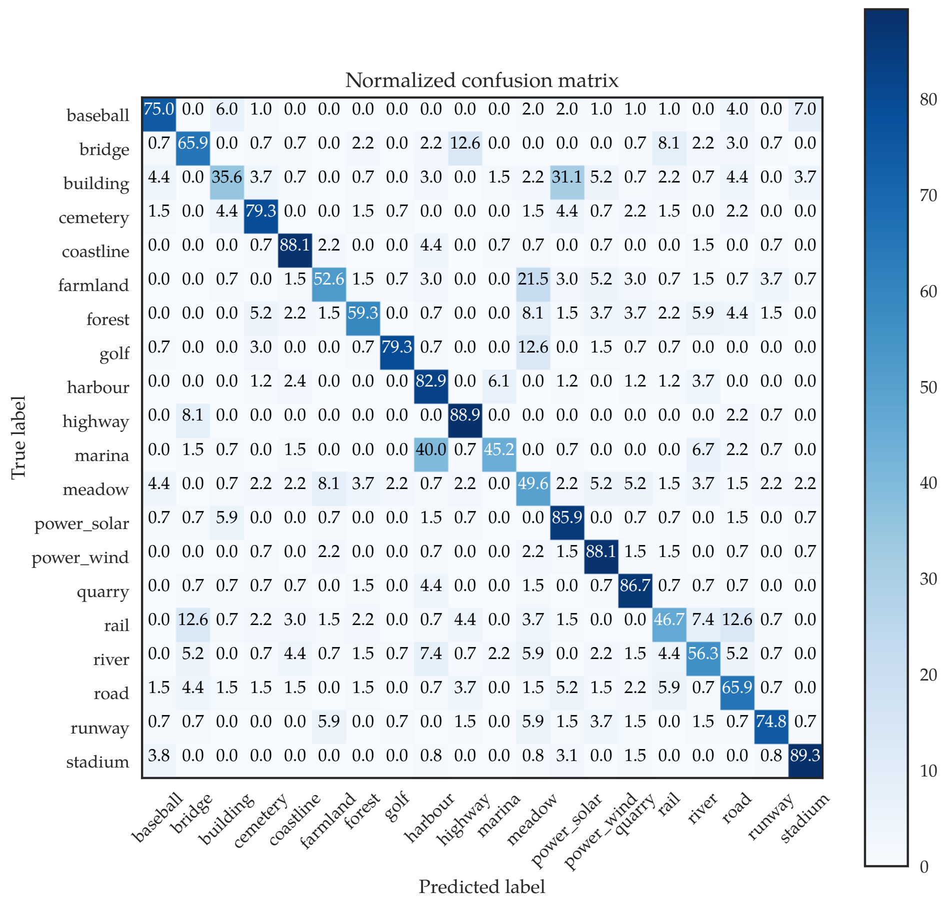
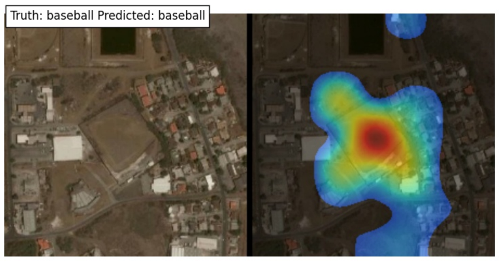


Figure 7.6: The gradCAM visualizatiosn differ for each target layer. Here are examples of gradCAM vizalization for class *bridge* for different layers in the ResNet50 network. The first visualization is for first layer while the last visualization is for the layer after the last skip connection. The other layers are sampled randomly and are ordered from the shallowest to the deepest.

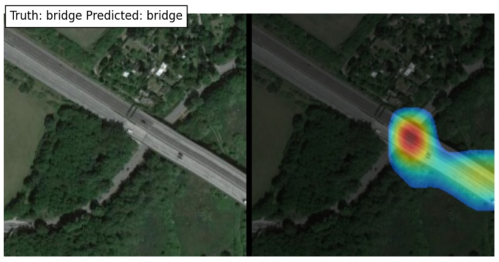


	baseball	bridge	building	cemetery	coastline	farmland	forest	golf	harbour	highway	marina	meadow	power_solar	power_wind	quarry	rail	river	road	runway	stadium
baseball	75.0	0.0	6.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	2.0	1.0	1.0	1.0	0.0	4.0	0.0	7.0
bridge	0.7	65.9	0.0	0.7	0.7	0.0	2.2	0.0	2.2	12.6	0.0	0.0	0.0	0.0	0.7	8.1	2.2	3.0	0.7	0.0
building	4.4	0.0	35.6	3.7	0.7	0.0	0.7	0.0	3.0	0.0	1.5	2.2	31.1	5.2	0.7	2.2	0.7	4.4	0.0	3.7
cemetery	1.5	0.0	4.4	79.3	0.0	0.0	1.5	0.7	0.0	0.0	0.0	1.5	4.4	0.7	2.2	1.5	0.0	2.2	0.0	0.0
coastline	0.0	0.0	0.0	0.7	88.1	2.2	0.0	0.0	4.4	0.0	0.7	0.7	0.0	0.7	0.0	0.0	1.5	0.0	0.7	0.0
farmland	0.0	0.0	0.7	0.0	1.5	52.6	1.5	0.7	3.0	0.0	0.0	21.5	3.0	5.2	3.0	0.7	1.5	0.7	3.7	0.7
forest	0.0	0.0	0.0	5.2	2.2	1.5	59.3	0.0	0.7	0.0	0.0	8.1	1.5	3.7	3.7	2.2	5.9	4.4	1.5	0.0
golf	0.7	0.0	0.0	3.0	0.0	0.0	0.7	79.3	0.7	0.0	0.0	12.6	0.0	1.5	0.7	0.7	0.0	0.0	0.0	0.0
harbour	0.0	0.0	0.0	1.2	2.4	0.0	0.0	0.0	82.9	0.0	6.1	0.0	1.2	0.0	1.2	1.2	3.7	0.0	0.0	0.0
highway	0.0	8.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	88.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.2	0.7	0.0
marina	0.0	1.5	0.7	0.0	1.5	0.0	0.0	0.0	40.0	0.7	45.2	0.0	0.7	0.0	0.0	0.0	6.7	2.2	0.7	0.0
meadow	4.4	0.0	0.7	2.2	2.2	8.1	3.7	2.2	0.7	2.2	0.0	49.6	2.2	5.2	5.2	1.5	3.7	1.5	2.2	2.2
power_solar	0.7	0.7	5.9	0.0	0.0	0.7	0.0	0.0	1.5	0.7	0.0	0.0	85.9	0.0	0.7	0.7	0.0	1.5	0.0	0.7
power_wind	0.0	0.0	0.0	0.7	0.0	2.2	0.0	0.0	0.7	0.0	0.0	2.2	1.5	88.1	1.5	1.5	0.0	0.7	0.0	0.7
quarry	0.0	0.7	0.7	0.7	0.7	0.0	1.5	0.0	4.4	0.0	0.0	1.5	0.0	0.7	86.7	0.7	0.7	0.7	0.0	0.0
rail	0.0	12.6	0.7	2.2	3.0	1.5	2.2	0.0	0.7	4.4	0.0	3.7	1.5	0.0	0.0	46.7	7.4	12.6	0.7	0.0
river	0.0	5.2	0.0	0.7	4.4	0.7	1.5	0.7	7.4	0.7	2.2	5.9	0.0	2.2	1.5	4.4	56.3	5.2	0.7	0.0
road	1.5	4.4	1.5	1.5	1.5	0.0	1.5	0.0	0.7	3.7	0.0	1.5	5.2	1.5	2.2	5.9	0.7	65.9	0.7	0.0
runway	0.7	0.7	0.0	0.0	0.0	5.9	0.0	0.7	0.0	1.5	0.0	5.9	1.5	3.7	1.5	0.0	1.5	0.7	74.8	0.7
stadium	3.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.8	3.1	0.0	1.5	0.0	0.0	0.0	0.8	89.3

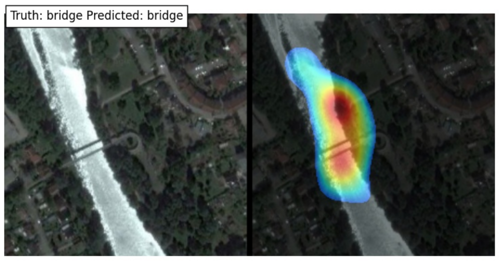
Figure 7.7: The confusion matrix over the **test** data of the ResNet50 network learned from scratch on the *CLS20* dataset.



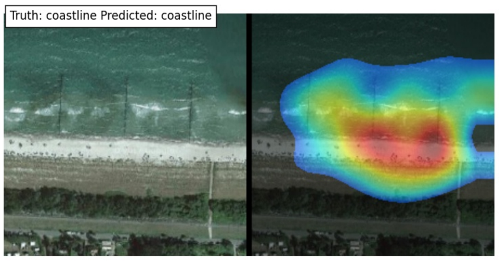
a



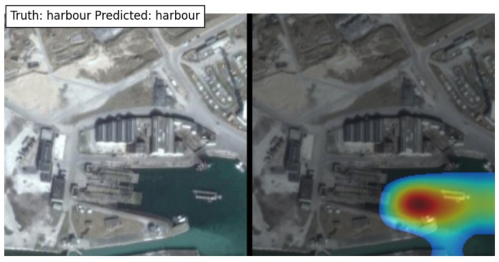
b



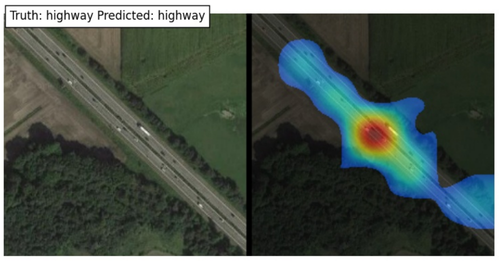
c



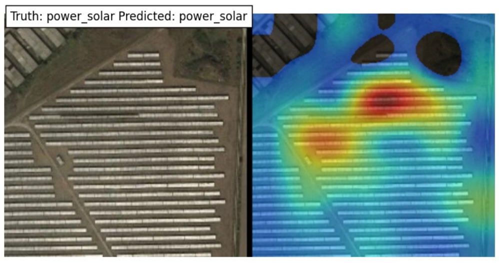
d



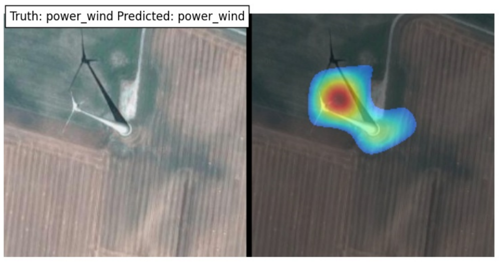
e



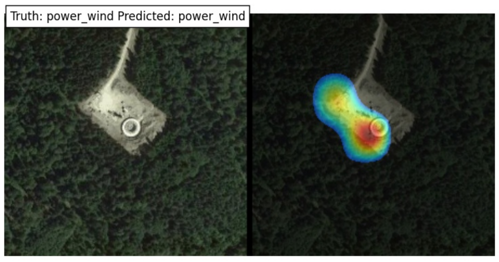
f



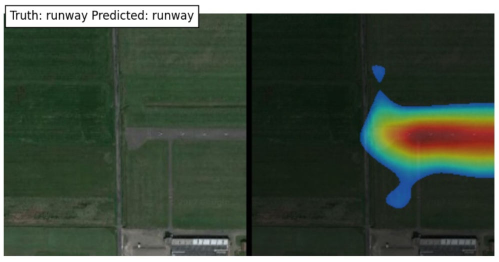
g



h

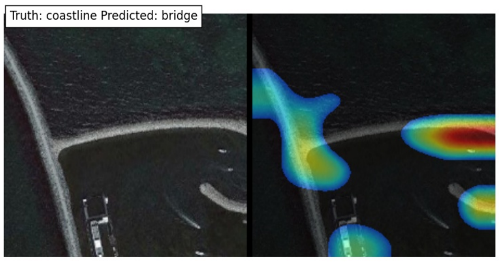


i

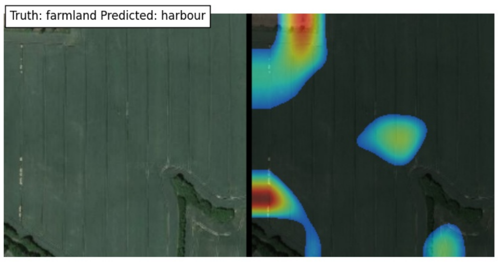


j

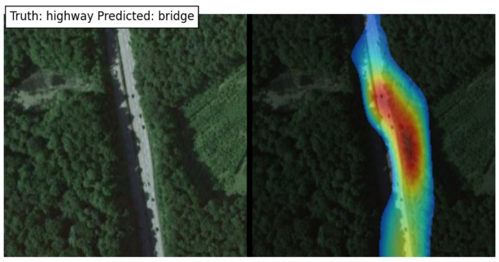
Figure 7.8: Visualization of correct classifications. The input image is on the left, gradCAM visualizations for the predicted class on the right.



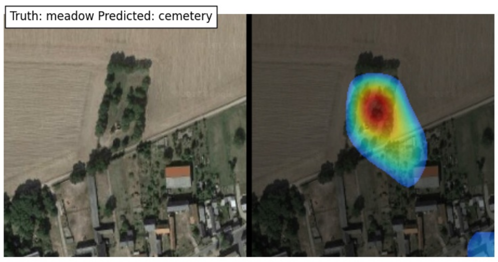
a



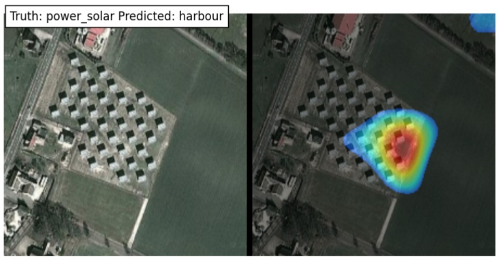
b



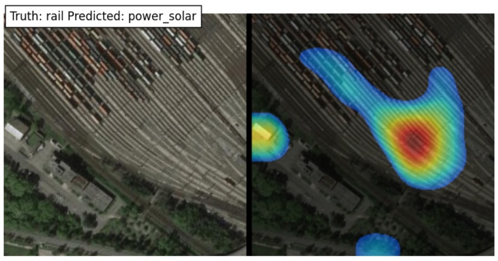
c



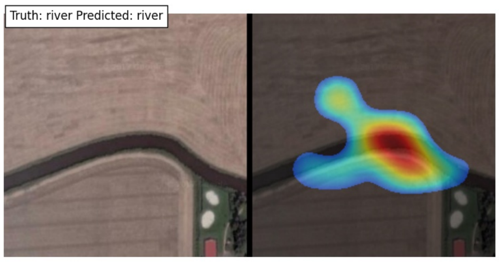
d



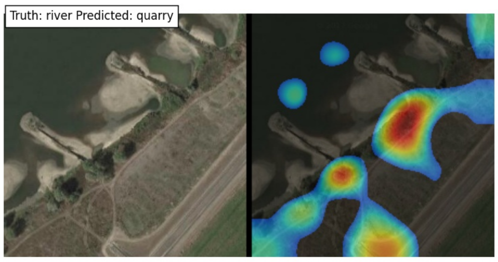
e



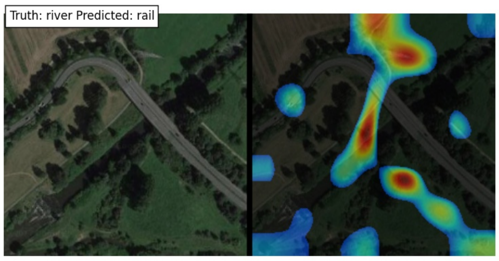
f



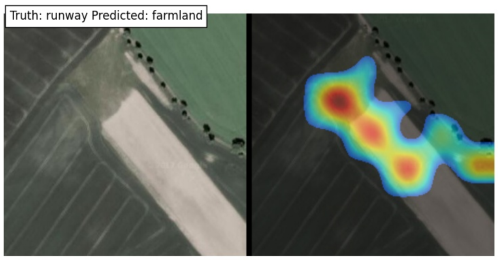
g



h

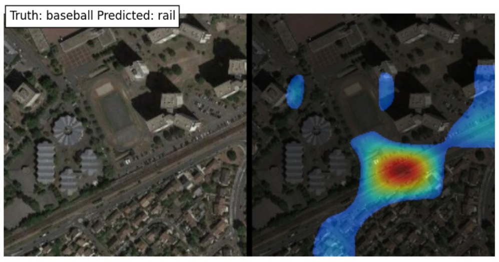


i

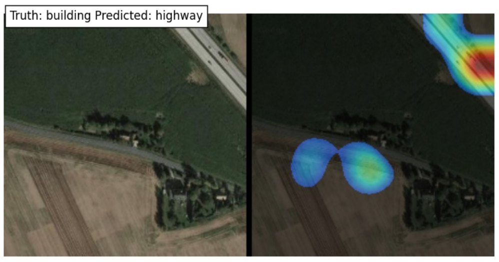


j

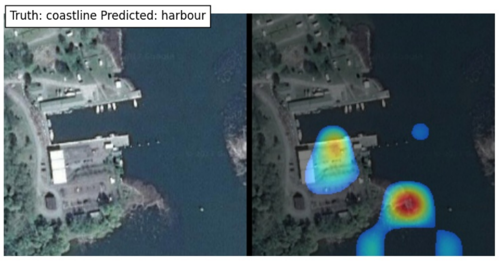
Figure 7.9: Visualization of the fail cases. The input image is on the left, gradCAM visualizations for the predicted class on the right.



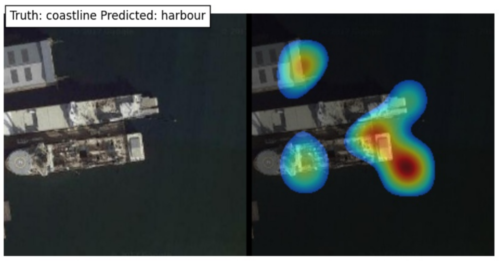
a



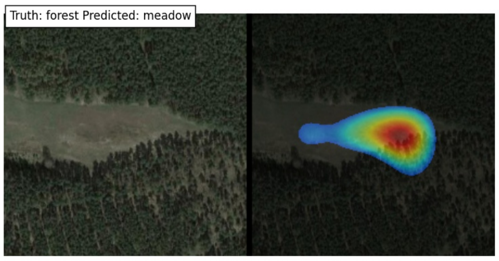
b



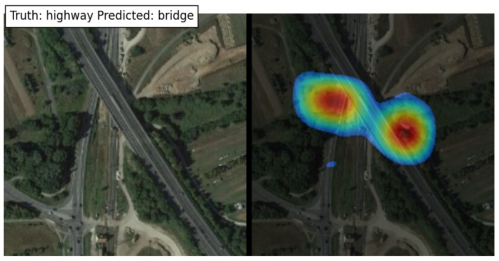
c



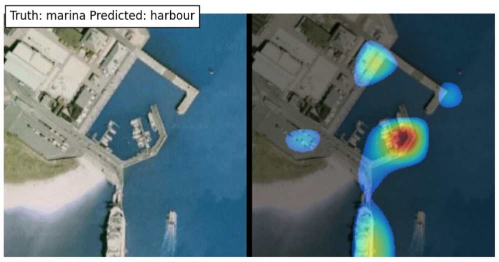
d



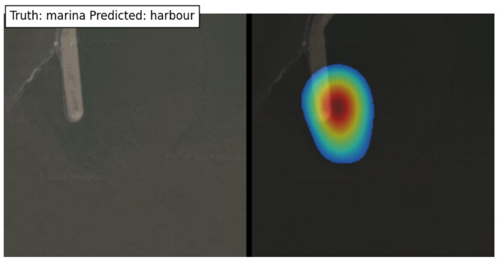
e



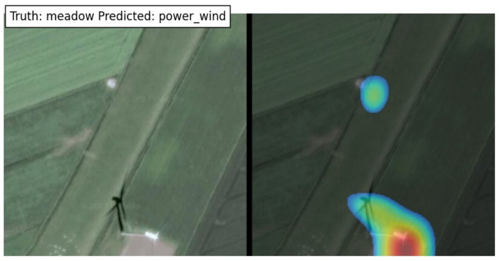
f



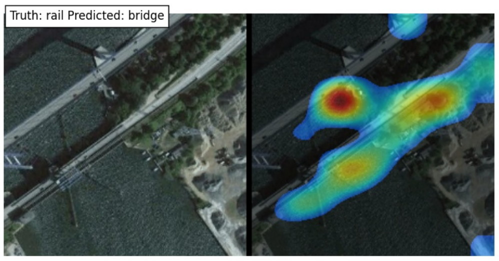
g



h



i



j

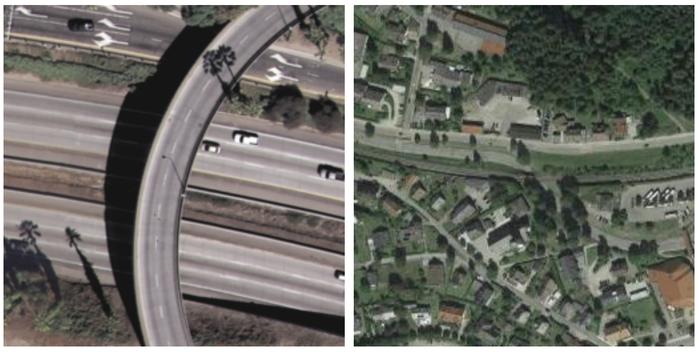
Figure 7.10: Visualization of the ambiguous cases. The input image is on the left, gradCAM visualizations for the predicted class on the right.

7.2.3.1 Mistaking *building* for *power_ solar*

While it seems that these two classes are very different, it is not the case — there are two most common cases where *power_ solar* class appears: either as a power farm or as small solar panels over the roofs of buildings (and mostly imperceptible in our data). The network thus learned that in some cases, the buildings are member of the *power_ solar* class. This behavior is unlikely to disappear unless the *power_ solar* data would be cleared. The *building* was classified as *power_ solar* in more than one third of samples (37%) which is quite bad considering that the building is usually very clearly defined in the image — it is, however, likely that the accuracy would be much higher if the network would be learned again but without the *power_ solar* class or using a manually cleaned dataset (part of possible future work).

7.2.3.2 *bridge* vs *road*, *river*, *highway*

The network make quite frequently mistakes on missing the bridge or classifying it as *bridge* when there is no bridge. This also might be partially caused by the difficulty of the dataset — it contains bridges that are very hard to distinguish in the aerial image. For example, the class *overpass* from the UCMerced Land Use dataset [197] is always visible at first glance while the bridges in our dataset might be distinguished only from the context (viz fig. 7.11).



a An example of class *overpass* [197]

b An example of class *bridge* from our dataset

Figure 7.11: xample of differences in difficulty of UCMerced Land Use dataset and proposed dataset

7.2.3.3 Frequent mistakes on classes *meadow* and *road*

These two classes are not frequently mistaken for a small set of similar classes but the mistakes are more uniformly distributed to other classes (viz rows in fig. 7.7). This is most likely caused by their frequent occurrence when other classes are present. Class *meadow* is a bit more often mistaken for *farmland* but otherwise the mistakes are quite uniform.

7.2.3.4 Good classification of class *highway*

The class *highway* is classified very accurately and the only classes that are mistaken with *highway* are *bridge*, *road*, *rail*, and *runway.* Furthermore, only the class *bridge* exhibits is mistaken with *highway* more frequently and this is caused by common occurrence of bridges in imagery of class *highway* and vice versa.

7.2.3.5 Other frequent mistakes

There are several frequent mistakes that are not really surprising and that are often caused by ambiguity in the data. First, *baseball* is quite often mistaken for *stadium*, *coastline* is mistaken for *harbour*, *golf* is mistaken for *meadow*, *harbour* for *marina*, *quarry* for *harbour* (some quarries are filled with water), and *river* for *bridge* and *harbour.*

7.2.3.6 Several visualizations of the labeling task

The labeling task did not differ much from the classification task, thus it is described only briefly. The labeling network behaves similarly as the classification networks in terms of mistakes. The mistakes of the network were of three types — missing a label, adding wrong label and mistaking one object for another. When mistaking one object for another, the network tended to make similar mistakes as the classification network. For example, the network has mistaken *bridge* for *rail* (other classes were labeled correctly) in fig. 7.12. Another type of mistake is omission when the network omits a label — for example, the network has omitted labels *highway*, *farmland*, and *meadow* in fig. 7.13. And the last type is addition when the network adds a label that should not be there, for example a label *meadow* has been added in fig. 7.14; this type of mistake was the least frequent.

And finally, a correct labeling is shown in fig. 7.15.

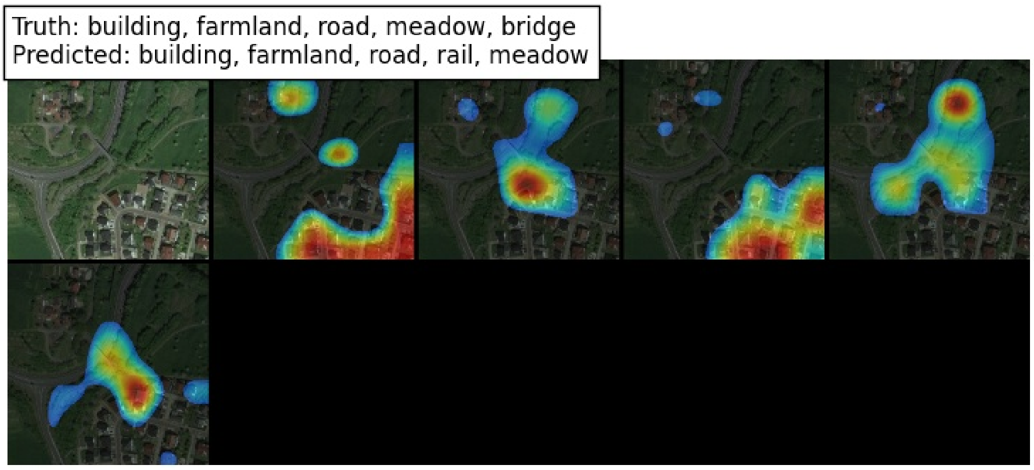

Truth: building, farmland, road, meadow, bridge
Predicted: building, farmland, road, rail, meadow

Figure 7.12: Example of output from the labeling task. The network has mistaken *bridge* for *rail*.

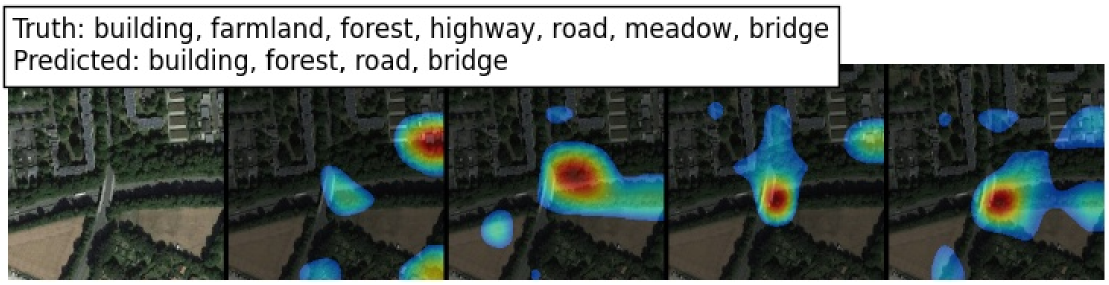

Truth: building, farmland, forest, highway, road, meadow, bridge
Predicted: building, forest, road, bridge

Figure 7.13: Example of output from the labeling task. The network has omitted labels *highway*, *farmland*, and *meadow*.

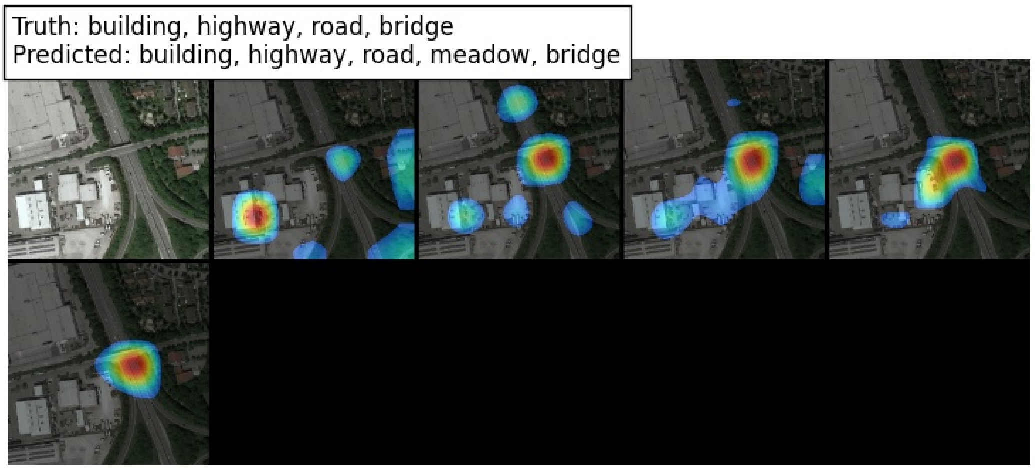

Truth: building, highway, road, bridge
Predicted: building, highway, road, meadow, bridge

Figure 7.14: Example of output from the labeling task. The network has added label *meadow*.

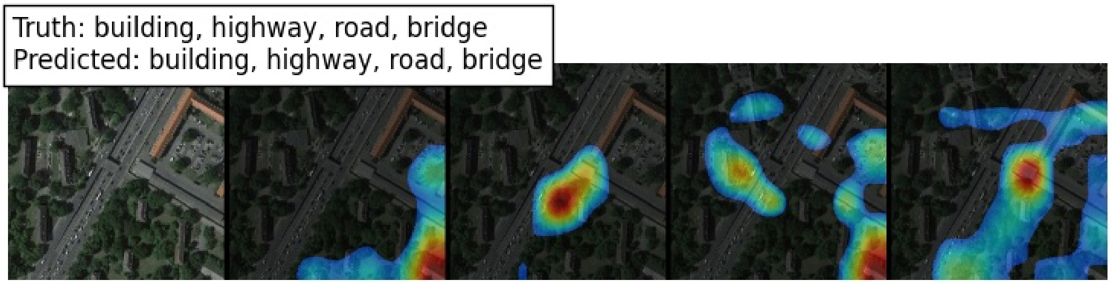

Truth: building, highway, road, bridge
Predicted: building, highway, road, bridge

Figure 7.15: Example of output from the labeling task. The network has correctly labeled all labels.

Chapter 8

Conclusion

This thesis evaluated the use of OpenStreetMap for obtaining the labels for aerial/satellite imagery and their subsequent use for network training by creating a new dataset that does not suffer from the weaknesses of other datasets in the field and that was used for training a convolutional network for aerial image classification. The contribution can be summarized into several points:

1. **Dataset creation with annotations from OSM**

   Several datasets was created using using imagery from Google Maps as they have been shown to be a suitable replacement for more costly satellite imagery for land use classification. The labels were obtained using the Overpass API from the OpenStretMap and several datasets were created. The created dataset address the weakness of other published datasets.

2. **Evaluation of augmentation suitable for the created dataset**

   A proposed augmentation pipeline was empirically evaluated and found to be suitable for use with the multisource data from Google Maps. The *complex* augmentation led to 2 pp gain in accuracy compared to *simple* accuracy.

3. **Transfer learning**A several experiments using obtained datasets were run in order to evaluate their quality and to establish a baseline classification performance for the datasets. These experiments included evaluating transfer learning from the ImageNet for several different architectures (VGG16, VGG19, Inception v3, Xception, and ResNet50).

   It was shown that the pretrained ResNet50 architecture is the most suitable for transfer learning and that adding a dense layer with dropout to the frozen network is beneficial.

4. **Evaluation of different activation functions**

   One of the created datasets was also used for the evaluation of use of several different activation functions (ReLU, ELU, PReLU) in the ResNet50 architecture. There were observed no significant differences in the use of ELU or PReLU in the ResNet50 architecture instead of the ReLU.

5. **Analysis of a network trained using one of the created datasets.**

   A performance of a ResNet50 network over a proposed *CLS20* dataset was then evaluated further and it was shown that the network was able to learn correct representations for most classes and that the most frequent mistakes were caused by the ambiguity in the data or unsuitable tagging in the OpenStreetMap.

The work describes the whole process from obtaining data through experiments with different architectures to final evaluation of selected neural network.

Chapter 9

Future work

While some of the work has been done there is plenty to research further. One of the ways to expand this work is examination of more CNNs architectures as many novel architectures has emerged recently (though they are mostly extensions of networks used in this work — e.g. *Inception v4* and *Inception-ResNet* [178], *Wide ResNet* [200], *ResNet-in-ResNet* [181], or *FractalNet* [98]). Other research directions include the analysis of the data augmentation for such multi-source data as are Google Maps — while the used augmentation pipeline was found beneficial, further analysis is needed to broke the gain into individual gains (losses) for individual augmentation steps in the pipeline. Yet another research direction concerns the dataset creation and annotating using the OpenStreetMap — the dataset could be manually cleaned to show how much of the error is caused by the mistakes in tagging in the OSM and also by the ambiguity in the data.

Bibliography

[1] F. Agostinelli, M. D. Hoffman, P. J. Sadowski, and P. Baldi. “Learning Activation Functions to Improve Deep Neural Networks”. In: *CoRR* abs/1412.6830 (2014). URL: http://arxiv.org/abs/1412.6830.

[2] T. Akilan, Q. M. J. Wu, and W. Jiang. “A Feature Embedding Strategy for High-level CNN representations from Multiple ConvNets”. In: *CoRR* abs/1705.04301 (2017). URL: https://arxiv.org/abs/1705.04301.

[3] A. Anandkumar and R. Ge. “Efficient approaches for escaping higher order saddle points in non-convex optimization”. In: *CoRR* abs/1602.05908 (2016). URL: http://arxiv.org/abs/1602.05908.

[4] A. Avramović and V. Risojević. *Aerial Image Classification.* 2014. URL: http://dsp.etfbl.net/aerial/.

[5] A. Avramović and V. Risojević. “Block-based semantic classification of high-resolution multispectral aerial images”. In: *Signal, Image and Video Processing* 10.1 (Oct. 2014), pp. 75–84. DOI: 10.1007/s11760-014-0704-x. URL: https://doi.org/10.1007%2Fs11760-014-0704-x.

[6] R. Bajcsy and M. Tavakoli. “Computer Recognition of Roads from Satellite Pictures”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.9 (Sept. 1976), pp. 623–637. DOI: 10.1109/tsmc.1976.4309568. URL: https://doi.org/10.1109%2Ftsmc.1976.4309568.

[7] A. Bansal, X. Chen, B. C. Russell, A. Gupta, and D. Ramanan. “PixelNet: Representation of the pixels, by the pixels, and for the pixels”. In: *CoRR* abs/1702.06506 (2017). URL: http://arxiv.org/abs/1702.06506.

[8] S. Basu, S. Ganguly, S. Mukhopadhyay, R. DiBiano, M. Karki, and R. R. Nemani. “DeepSat - A Learning framework for Satellite Imagery”. In: *CoRR* abs/1509.03602 (2015). URL: http://arxiv.org/abs/1509.03602.

[9] S. Basu, S. Ganguly, S. Mukhopadhyay, R. DiBiano, M. Karki, and R. R. Nemani. *SAT-4 and SAT-6 airborne datasets.* 2015. URL: http://csc.lsu.edu/~saikat/deepsat/ (visited on 05/15/2017).

[10] P. R. Baumann. *History of Remote Sensing, Aerial Photography.* 2014. URL: https://www.oneonta.edu/faculty/baumanpr/geosat2/RS%20History%20I/RS-History-Part-1.htm (visited on 05/15/2017).

[11] J. Benediktsson, P. Swain, and O. Ersoy. "Neural Network Approaches Versus Statistical Methods In Classification Of Multisource Remote Sensing Data". In: *IEEE Transactions on Geoscience and Remote Sensing* 28.4 (July 1990), pp. 540–552. DOI: 10.1109/tgrs.1990.572944. URL: https://doi.org/10.1109%2Ftgrs.1990.572944.

[12] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. "Greedy Layer-wise Training of Deep Networks". In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. NIPS'06. Canada: MIT Press, 2006, pp. 153–160. URL: https://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf.

[13] H. Bischof, W. Schneider, and A. Pinz. "Multispectral classification of Landsat-images using neural networks". In: *IEEE Transactions on Geoscience and Remote Sensing* 30.3 (May 1992), pp. 482–490. DOI: 10.1109/36.142926. URL: https://doi.org/10.1109%2F36.142926.

[14] J. E. Boggess. *Identification of Roads in Satellite Imagery Using Artificial Neural Networks: A Contextual Approach.* Tech. rep. Mississippi State, MS, USA, 1993. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.897.

[15] H. Bourlard and Y. Kamp. "Auto-association by multilayer perceptrons and singular value decomposition". In: *Biological Cybernetics* 59 (1988), pp. 291–294. URL: https://pdfs.semanticscholar.org/f582/1548720901c89b3b7481f7500d7cd64e99bd.pdf.

[16] T. M. Breuel. "The Effects of Hyperparameters on SGD Training of Neural Networks". In: *CoRR* abs/1508.02788 (2015). URL: http://arxiv.org/abs/1508.02788.

[17] D. S. Broomhead and D. Lowe. "Radial basis functions, multi-variable functional interpolation and adaptive networks". In: *RSRE Memorandum No. 4148* (1988). URL: http://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf.

[18] L. Bruzzone and B. Demir. "A Review of Modern Approaches to Classification of Remote Sensing Data". In: *Land Use and Land Cover Mapping in Europe.* Springer Netherlands, 2014, pp. 127–143. DOI: 10.1007/978-94-007-7969-3_9. URL: https://doi.org/10.1007%2F978-94-007-7969-3_9.

[19] U. S. C. Burea. *TIGER Products.* URL: https://www.census.gov/geo/maps-data/data/tiger.html (visited on 05/15/2017).

[20] B. CHen, J.-A. Ting, B. Marlin, and N. de Freitas. "Deep Learning of Invariant Spatio-Temporal Features from Video". In: *Proceedings of NIPS 2010 Workshop.* 2010. URL: http://www.cs.ubc.ca/~nando/papers/nipsworkshop2010.pdf.

[21] C. Chen, B. Zhang, H. Su, W. Li, and L. Wang. "Land-use scene classification using multi-scale completed local binary patterns". In: *Signal, Image and Video Processing* 10.4 (July 2015), pp. 745–752. DOI: 10.1007/s11760-015-0804-2. URL: https://doi.org/10.1007%2Fs11760-015-0804-2.

[22] G. Cheng, J. Han, and X. Lu. *NWPU-RESISC45 dataset.* 2016. URL: `http://www.escience.cn/people/JunweiHan/NWPU-RESISC45.html` (visited on 05/15/2017).

[23] G. Cheng, J. Han, and X. Lu. "Remote Sensing Image Scene Classification: Benchmark and State of the Art". In: *CoRR* abs/1703.00121 (2017). URL: `http://arxiv.org/abs/1703.00121`.

[24] G. Cheng, J. Han, P. Zhou, and L. Guo. "Multi-class geospatial object detection and geographic image classification based on collection of part detectors". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 98 (Dec. 2014), pp. 119–132. DOI: `10.1016/j.isprsjprs.2014.10.002`. URL: `https://doi.org/10.1016%2Fj.isprsjprs.2014.10.002`.

[25] A. M. Cheriyadat. "Unsupervised Feature Learning for Aerial Scene Classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 52.1 (Jan. 2014), pp. 439–451. DOI: `10.1109/tgrs.2013.2241444`. URL: `https://doi.org/10.1109%2Ftgrs.2013.2241444`.

[26] F. Chollet et al. *Keras.* `https://github.com/fchollet/keras`. 2015.

[27] F. Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *CoRR* abs/1610.02357 (2016). URL: `http://arxiv.org/abs/1610.02357`.

[28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.* 2014. eprint: `arXiv:1412.3555`. URL: `https://arxiv.org/abs/1412.3555v1`.

[29] D. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *CoRR* abs/1511.07289 (2015). URL: `http://arxiv.org/abs/1511.07289`.

[30] M. Cord. *Deep CNN and Weak Supervision Learning for visual recognition.* 2016. URL: `https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/` (visited on 05/09/2017).

[31] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: `10.1007/bf02551274`. URL: `https://doi.org/10.1007%2Fbf02551274`.

[32] D. Dai and W. Yang. "Satellite Image Classification via Two-Layer Sparse Coding With Biased Image Representation". In: *IEEE Geoscience and Remote Sensing Letters* 8.1 (Jan. 2011), pp. 173–176. DOI: `10.1109/lgrs.2010.2055033`. URL: `https://doi.org/10.1109%2Flgrs.2010.2055033`.

[33] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio. "RMSProp and equilibrated adaptive learning rates for non-convex optimization". In: *CoRR* abs/1502.04390 (2015). URL: `http://arxiv.org/abs/1502.04390`.

[34] Decatur. "Application of neural networks to terrain classification". In: *International Joint Conference on Neural Networks.* IEEE, 1989. DOI: `10.1109/ijcnn.1989.118592`. URL: `https://doi.org/10.1109%2Fijcnn.1989.118592`.

[35] DinoTools. *Python Overpass API.* 2017. URL: `https://python-overpy.readthedocs.io/en/latest/` (visited on 05/12/2017).

[36] P. Dollar, Z. Tu, and S. Belongie. "Supervised Learning of Edges and Object Boundaries". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06).* IEEE. DOI: `10.1109/cvpr.2006.298`. URL: `https://doi.org/10.1109%2Fcvpr.2006.298`.

[37] T. Dozat. *Incorporating Nesterov Momentum into Adam.* Tech. rep. 2015. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.897`.

[38] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12 (July 2011), pp. 2121–2159. ISSN: 1532-4435. URL: `http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf`.

[39] V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning". In: *CoRR* abs/1603.07285 (2016). URL: `https://arxiv.org/abs/1603.07285`.

[40] J. L. Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211. URL: `https://crl.ucsd.edu/~elman/Papers/fsit.pdf`.

[41] M. L. Forcada. *Neural Networks: Automata and Formal Models of Computation.* 2000. URL: `http://www.dlsi.ua.es/~mlf/nnafmc/pbook.pdf` (visited on 05/07/2017).

[42] K. Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202. DOI: `10.1007/bf00344251`. URL: `https://doi.org/10.1007%2Fbf00344251`.

[43] GDAL Development Team. *GDAL - Geospatial Data Abstraction Library, Version 2.1.3.* Open Source Geospatial Foundation. 2016. URL: `%E2%80%8Bhttp://www.gdal.org`.

[44] R. Ge, F. Huang, C. Jin, and Y. Yuan. "Escaping From Saddle Points - Online Stochastic Gradient for Tensor Decomposition". In: *CoRR* abs/1503.02101 (2015). URL: `http://arxiv.org/abs/1503.02101`.

[45] D. Geman and B. Jedynak. "An active testing model for tracking roads in satellite images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18.1 (1996), pp. 1–14. DOI: `10.1109/34.476006`. URL: `https://doi.org/10.1109%2F34.476006`.

[46] P. Ghamisi, M. D. Mura, and J. A. Benediktsson. "A Survey on Spectral–Spatial Classification Techniques Based on Attribute Profiles". In: *IEEE Transactions on Geoscience and Remote Sensing* 53.5 (May 2015), pp. 2335–2353. DOI: `10.1109/tgrs.2014.2358934`. URL: `https://doi.org/10.1109%2Ftgrs.2014.2358934`.

[47] S. Gillies et al. *Shapely: manipulation and analysis of geometric objects.* toblerity.org, 2007–. URL: `https://github.com/Toblerity/Shapely`.

[48] X. Glorot, A. Bordes, and Y. Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson, and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323. URL: `http://proceedings.mlr.press/v15/glorot11a.html`.

[49] L. B. Godfrey and M. S. Gashler. "A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks". In: *CoRR* abs/1602.01321 (2016). URL: `http://arxiv.org/abs/1602.01321`.

[50] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks*. 2014. eprint: `arXiv:1406.2661`. URL: `https://arxiv.org/pdf/1406.2661v1.pdf`.

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

[52] Google. *Google Static Maps API*. 2017. URL: `https://developers.google.com/maps/documentation/static-maps/` (visited on 05/09/2017).

[53] Google. *TensorBoard README*. 2017. URL: `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/tensorboard/README.md` (visited on 05/12/2017).

[54] A. Graves, G. Wayne, and I. Danihelka. *Neural Turing Machines*. 2014. eprint: `arXiv:1410.5401`. URL: `https://arxiv.org/abs/1410.5401v2`.

[55] R. M. Haralick. "Automatic remote sensor image processing". In: *Topics in Applied Physics*. Springer Berlin Heidelberg, 1976, pp. 5–63. DOI: `10.1007/3540075798_20`. URL: `http://haralick.org/book_chapters/automatic_remote_sensing.pdf`.

[56] R. M. Haralick, K. Shanmugam, and I. Dinstein. "Textural Features for Image Classification". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3.6 (Nov. 1973), pp. 610–621. DOI: `10.1109/tsmc.1973.4309314`. URL: `https://doi.org/10.1109%2Ftsmc.1973.4309314`.

[57] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Institute of Electrical and Electronics Engineers (IEEE), June 2016. DOI: `10.1109/cvpr.2016.90`. URL: `https://doi.org/10.1109%2Fcvpr.2016.90`.

[58] K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *CoRR* abs/1502.01852 (2015). URL: `http://arxiv.org/abs/1502.01852`.

[59] K. He, X. Zhang, S. Ren, and J. Sun. "Identity Mappings in Deep Residual Networks". In: *CoRR* abs/1603.05027 (2016). URL: `http://arxiv.org/abs/1603.05027`.

[60] D. Hebb. *The Organization of Behavior*. New York: Wiley, 1949.

[61] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1991. ISBN: 0-201-50395-6.

[62] G. E. Hinton and T. J. Sejnowski. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1". In: ed. by D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning and Relearning in Boltzmann Machines, pp. 282–317. ISBN: 0-262-68053-X. URL: `https://www.researchgate.net/publication/242509302_Learning_and_relearning_in_Boltzmann_machines`.

[63] S. Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München.* 1991.

[64] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: `10.1162/neco.1997.9.8.1735`. URL: `https://doi.org/10.1162%2Fneco.1997.9.8.1735`.

[65] S. Honari, J. Yosinski, P. Vincent, and C. Pal. "Recombinator Networks: Learning Coarse-to-Fine Feature Aggregation". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Institute of Electrical and Electronics Engineers (IEEE), June 2016. DOI: `10.1109/cvpr.2016.619`. URL: `https://doi.org/10.1109%2Fcvpr.2016.619`.

[66] J. J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the National Academy of Sciences of the United States of America.* Vol. 79. 8. 1982, pp. 2554–2558. URL: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC346238/`.

[67] K. Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (Jan. 1991), pp. 251–257. DOI: `10.1016/0893-6080(91)90009-t`. URL: `https://doi.org/10.1016%2F0893-6080%2891%2990009-t`.

[68] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. DOI: `10.1016/0893-6080(89)90020-8`. URL: `https://doi.org/10.1016%2F0893-6080%2889%2990020-8`.

[69] Q. Hu, W. Wu, T. Xia, Q. Yu, P. Yang, Z. Li, and Q. Song. "Exploring the Use of Google Earth Imagery and Object-Based Methods in Land Use/Cover Mapping". In: *Remote Sensing* 5.11 (Nov. 2013), pp. 6026–6042. DOI: `10.3390/rs5116026`. URL: `https://doi.org/10.3390%2Frs5116026`.

[70] C. Huang, L. S. Davis, and J. R. G. Townshend. "An assessment of support vector machines for land cover classification". In: *International Journal of Remote Sensing* 23.4 (Jan. 2002), pp. 725–749. DOI: `10.1080/01431160110040323`. URL: `https://doi.org/10.1080%2F01431160110040323`.

[71] J. Huang, S. R. Kumar, M. Mitra, W.-J. Zhu, and R. Zabih. “Image Indexing Using Color Correlograms”. In: *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR ’97)*. CVPR ’97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 762–. ISBN: 0-8186-7822-4. URL: `http://www.cs.cornell.edu/~rdz/Papers/Huang-CVPR97.pdf`.

[72] J. D. Hunter. “Matplotlib: A 2D Graphics Environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: `10.1109/mcse.2007.55`. URL: `http://dx.doi.org/10.1109/MCSE.2007.55`.

[73] J. Idelsohn. “A learning system for terrain recognition”. In: *Pattern Recognition* 2.4 (Dec. 1970), pp. 293–301. DOI: `10.1016/0031-3203(70)90019-1`. URL: `https://doi.org/10.1016%2F0031-3203%2870%2990019-1`.

[74] I. W. III/4. *ISPRS 2D Semantic Labeling Contest*. 2016. URL: `http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html` (visited on 05/07/2017).

[75] *Image Classification Transfer Learning with Inception v3*. URL: `https://codelabs.developers.google.com/codelabs/cpb102-txf-learning` (visited on 05/11/2017).

[76] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *CoRR* abs/1611.07004 (2016). URL: `http://arxiv.org/abs/1611.07004`.

[77] A. G. Ivakhnenko. “Polynomial Theory of Complex Systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-1.4 (Oct. 1971), pp. 364–378. DOI: `10.1109/tsmc.1971.4308320`. URL: `https://doi.org/10.1109%2Ftsmc.1971.4308320`.

[78] A. Ivakhnenko. “Heuristic self-organization in problems of engineering cybernetics”. In: *Automatica* 6.2 (Mar. 1970), pp. 207–219. DOI: `10.1016/0005-1098(70)90092-0`. URL: `https://doi.org/10.1016%2F0005-1098%2870%2990092-0`.

[79] H. Jaeger. “Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication”. In: *Science* 304.5667 (Apr. 2004), pp. 78–80. DOI: `10.1126/science.1091277`. URL: `https://doi.org/10.1126%2Fscience.1091277`.

[80] K. Janocha and W. M. Czarnecki. “On Loss Functions for Deep Neural Networks in Classification”. In: *CoRR* abs/1702.05659 (2017). URL: `http://arxiv.org/abs/1702.05659`.

[81] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *CoRR* abs/1408.5093 (2014). URL: `http://arxiv.org/abs/1408.5093`.

[82] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. “Deep Learning with S-shaped Rectified Linear Activation Units”. In: *CoRR* abs/1512.07030 (2015). URL: `http://arxiv.org/abs/1512.07030`.

[83] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 2015-05-12]. 2001–. URL: `http://www.scipy.org/`.

[84] A. Jung. *imgaug – Image augmentation library for machine learning.* 2017. URL: https://github.com/aleju/imgaug (visited on 05/15/2017).

[85] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. "A Convolutional Neural Network for Modelling Sentences". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Association for Computational Linguistics, 2014. DOI: 10.3115/v1/p14-1062. URL: https://doi.org/10.3115%2Fv1%2Fp14-1062.

[86] A. Karpathy. *CS231n: Convolutional Neural Networks for Visual Recognition – Module 2: Convolutional Neural Networks.* 2017. URL: http://cs231n.github.io/neural-networks-2/ (visited on 05/09/2017).

[87] R. Kettig and D. Landgrebe. "Classification of Multispectral Image Data by Extraction and Classification of Homogeneous Objects". In: *IEEE Transactions on Geoscience Electronics* 14.1 (Jan. 1976), pp. 19–26. DOI: 10.1109/tge.1976.294460. URL: https://doi.org/10.1109%2Ftge.1976.294460.

[88] Y. Kim. "Convolutional Neural Networks for Sentence Classification". In: *CoRR* abs/1408.5882 (2014). URL: http://arxiv.org/abs/1408.5882.

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

[90] D. P. Kingma and M. Welling. *Auto-association by multilayer perceptrons and singular value decomposition.* arXiv preprint. 2013. URL: https://arxiv.org/abs/1312.6114.

[91] S. Kluckner and H. Bischof. "Semantic classification by covariance descriptors within a randomized forest". In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops.* IEEE, Sept. 2009. DOI: 10.1109/iccvw.2009.5457638. URL: https://doi.org/10.1109%2Ficcvw.2009.5457638.

[92] S. Kluckner, T. Mauthner, P. M. Roth, and H. Bischof. "Semantic Classification in Aerial Imagery by Integrating Appearance and Height Information". In: *Computer Vision – ACCV 2009.* Springer Berlin Heidelberg, 2010, pp. 477–488. DOI: 10.1007/978-3-642-12304-7_45. URL: https://doi.org/10.1007%2F978-3-642-12304-7_45.

[93] T. Kohonen. "Self-organized formation of topologically correct feature maps". In: *Biological Cybernetics* 43.1 (1982), pp. 59–69. DOI: 10.1007/bf00337288. URL: https://doi.org/10.1007%2Fbf00337288.

[94] R. Kotikalapudi. *keras-vis: Keras Visualization Toolkit.* 2015. URL: https://github.com/raghakot/keras-vis.

[95] A. Krizhevsky. *Learning multiple layers of features from tiny images.* Tech. rep. 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[96] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[97] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. "Deep Convolutional Inverse Graphics Network". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 2539–2547. URL: http://papers.nips.cc/paper/5851-deep-convolutional-inverse-graphics-network.pdf.

[98] G. Larsson, M. Maire, and G. Shakhnarovich. "FractalNet: Ultra-Deep Neural Networks without Residuals". In: *CoRR* abs/1605.07648 (2016). URL: http://arxiv.org/abs/1605.07648.

[99] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: http://dx.doi.org/10.1162/neco.1989.1.4.541.

[100] Y. Lecun. "A theoretical framework for back-propagation". In: *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*. Ed. by D. Touretzky, G. Hinton, and T. Sejnowski. Morgan Kaufmann, 1988, pp. 21–28. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-88.pdf.

[101] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE 86.11*. 1998, pp. 2278–2324. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf.

[102] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. "Deeply-Supervised Nets". In: *AISTATS*. Ed. by G. Lebanon and S. V. N. Vishwanathan. Vol. 38. JMLR Proceedings. JMLR.org, 2015. URL: http://arxiv.org/pdf/1409.5185.pdf.

[103] J. Lee, R. Weger, S. Sengupta, and R. Welch. "A neural network approach to cloud classification". In: *IEEE Transactions on Geoscience and Remote Sensing* 28.5 (1990), pp. 846–855. DOI: 10.1109/36.58972. URL: https://doi.org/10.1109%2F36.58972.

[104] H. Lewis. "A neural network approach to cloud classification from multi-temporal satellite imagery". In: *4th International Conference on Artificial Neural Networks*. IEE, 1995. DOI: 10.1049/cp:19950539. URL: https://doi.org/10.1049%2Fcp%3A19950539.

[105] L.-L. Li, T.-S. Jin, H. Shi, and C.-H. Li. "An approach to the interpretation of aerial image". In: *2010 International Conference on Machine Learning and Cybernetics*. IEEE, July 2010. DOI: 10.1109/icmlc.2010.5580559. URL: https://doi.org/10.1109%2Ficmlc.2010.5580559.

[106] S. Li, J. Jiao, Y. Han, and T. Weissman. “Demystifying ResNet”. In: *CoRR* abs/1611.01186 (2016). URL: `http://arxiv.org/abs/1611.01186`.

[107] Z. Lian, H. Huang, Y. Tan, Y. Cui, X. Jing, and X. Wang. “DropConnect Regularization Method with Sparsity Constraint for Neural Networks”. In: *Chinese Journal of Electronics* 25.1 (Jan. 2016), pp. 152–158. DOI: `10.1049/cje.2016.01.023`. URL: `https://doi.org/10.1049%2Fcje.2016.01.023`.

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

[109] S. Linnainmaa. “Taylor expansion of the accumulated rounding error”. In: *BIT* 16.2 (June 1976), pp. 146–160. DOI: `10.1007/bf01931367`. URL: `https://doi.org/10.1007%2Fbf01931367`.

[110] W. Liu and V. Prinet. “Building detection from high-resolution satellite image using probability model”. In: *Proceedings. 2005 IEEE International Geoscience and Remote Sensing Symposium IGARSS ’05.* IEEE, 2005. DOI: `10.1109/igarss.2005.1525759`. URL: `https://doi.org/10.1109%2Figarss.2005.1525759`.

[111] B. Luo, S. Jiang, and L. Zhang. “Indexing of Remote Sensing Images With Different Resolutions by Multiple Features”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 6.4 (Aug. 2013), pp. 1899–1912. DOI: `10.1109/jstars.2012.2228254`. URL: `https://doi.org/10.1109%2Fjstars.2012.2228254`.

[112] W. Maass, T. Natschläger, and H. Markram. “Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations”. In: *Neural Computation* 14.11 (Nov. 2002), pp. 2531–2560. DOI: `10.1162/089976602760407955`. URL: `https://doi.org/10.1162%2F089976602760407955`.

[113] Mapbox. *Mapbox Studio.* 2017. URL: `https://www.mapbox.com/mapbox-studio/` (visited on 05/09/2017).

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

[115] H. Mayer. “Automatic Object Extraction from Aerial Imagery—A Survey Focusing on Buildings”. In: *Computer Vision and Image Understanding* 74.2 (May 1999), pp. 138–149. DOI: `10.1006/cviu.1999.0750`. URL: `https://doi.org/10.1006%2Fcviu.1999.0750`.

[116] W. S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: `10.1007/bf02478259`. URL: `https://doi.org/10.1007%2Fbf02478259`.

[117] D. M. McKeown, W. A. Harvey, and J. McDermott. “Rule-Based Interpretation of Aerial Imagery”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-7.5 (Sept. 1985), pp. 570–585. DOI: `10.1109/tpami.1985.4767704`. URL: `https://doi.org/10.1109%2Ftpami.1985.4767704`.

[118] W. McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference.* Ed. by S. van der Walt and J. Millman. 2010, pp. 51–56.

[119] R. Mehra. "Group method of data handling (GMDH): Review and experience". In: *1977 IEEE Conference on Decision and Control including the 16th Symposium on Adaptive Processes and A Special Symposium on Fuzzy Set Theory and Applications.* IEEE, Dec. 1977. DOI: `10.1109/cdc.1977.271540`. URL: `https://doi.org/10.1109%2Fcdc.1977.271540`.

[120] D. Mishkin and J. Matas. "All you need is a good init". In: *CoRR* abs/1511.06422 (2015). URL: `http://arxiv.org/abs/1511.06422`.

[121] D. Mishkin, N. Sergievskiy, and J. Matas. "Systematic evaluation of CNN advances on the ImageNet". In: *CoRR* abs/1606.02228 (2016). URL: `http://arxiv.org/abs/1606.02228`.

[122] V. Mnih. "Machine Learning for Aerial Image Labeling". PhD thesis. University of Toronto, 2013. URL: `https://www.cs.toronto.edu/~vmnih/docs/Mnih_Volodymyr_PhD_Thesis.pdf`.

[123] V. Mnih and G. Hinton. "Learning to Detect Roads in High-Resolution Aerial Images". In: *Proceedings of the 11th European Conference on Computer Vision (ECCV).* Sept. 2010. URL: `https://www.cs.toronto.edu/~vmnih/docs/road_detection.pdf`.

[124] V. Mnih and G. Hinton. "Learning to Label Aerial Images from Noisy Data". In: *Proceedings of the 29th Annual International Conference on Machine Learning (ICML 2012).* Edinburgh, Scotland, June 2012. URL: `https://www.cs.toronto.edu/~vmnih/docs/noisy_maps.pdf`.

[125] S. Muruganandham. "Semantic Segmentation of Satellite Images using Deep Learning". MA thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics, Aug. 2016.

[126] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10).* Ed. by J. Fürnkranz and T. Joachims. Omnipress, 2010, pp. 807–814. URL: `http://www.cs.toronto.edu/~fritz/absps/reluICML.pdf`.

[127] NASA. *History.* 2017. URL: `https://landsat.gsfc.nasa.gov/about/history/` (visited on 05/15/2017).

[128] T. T. Nguyen, S. Kluckner, H. Bischof, and F. Leberl. "Aerial Photo Building Classification by Stacking Appearance and Elevation Measurements". In: *In: Proceedings ISPRS, 100 Years ISPRS - Advancing Remote Sensing Science, on CD-ROM.* 2010. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.190.3127`.

[129] K. Nogueira, W. O. Miranda, and J. A. D. Santos. “Improving Spatial Feature Representation from Aerial Scenes by Using Convolutional Networks”. In: *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*. IEEE, Aug. 2015. DOI: 10.1109/sibgrapi.2015.39. URL: https://doi.org/10.1109%2Fsibgrapi.2015.39.

[130] K. Nogueira, O. A. B. Penatti, and J. A. dos Santos. “Towards Better Exploiting Convolutional Neural Networks for Remote Sensing Scene Classification”. In: *CoRR* abs/1602.01517 (2016). URL: http://arxiv.org/abs/1602.01517.

[131] K. Nogueira, J. A. D. Santos, T. Fornazari, T. S. F. Silva, L. P. Morellato, and R. D. S. Torres. “Towards vegetation species discrimination by using data-driven descriptors”. In: *2016 9th IAPR Workshop on Pattern Recogniton in Remote Sensing (PRRS)*. IEEE, Dec. 2016. DOI: 10.1109/prrs.2016.7867024. URL: https://doi.org/10.1109%2Fprrs.2016.7867024.

[132] H. Noh, S. Hong, and B. Han. “Learning Deconvolution Network for Semantic Segmentation”. In: *CoRR* abs/1505.04366 (2015). URL: http://arxiv.org/abs/1505.04366.

[133] S. J. Nowlan and G. E. Hinton. “Simplifying Neural Networks by Soft Weight-Sharing”. In: *Neural Computation* 4.4 (July 1992), pp. 473–493. DOI: 10.1162/neco.1992.4.4.473. URL: https://doi.org/10.1162%2Fneco.1992.4.4.473.

[134] *OpenAerialMap*. URL: https://openaerialmap.org/ (visited on 05/15/2017).

[135] OpenStreetMap. *Forest*. 2017. URL: http://wiki.openstreetmap.org/wiki/Forest (visited on 05/15/2017).

[136] OpenStreetMap. *Harbour*. 2017. URL: http://wiki.openstreetmap.org/wiki/Harbour (visited on 05/15/2017).

[137] OpenStreetMap. *Key:water*. 2017. URL: https://wiki.openstreetmap.org/wiki/Key:water (visited on 05/15/2017).

[138] OpenStreetMap. *Map Features*. 2017. URL: http://wiki.openstreetmap.org/wiki/Map_Features (visited on 05/15/2017).

[139] OpenStreetMap. *Overpass API*. 2017. URL: http://wiki.openstreetmap.org/wiki/Overpass_API (visited on 05/15/2017).

[140] OpenStreetMap. *Overpass turbo*. 2017. URL: http://overpass-turbo.eu/ (visited on 05/15/2017).

[141] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org . 2017.

[142] Ö. Özak. *GeoRasters: Fast and flexible tool to work with GIS raster files*. [Online; accessed 2017-02-27]. 2015–. URL: https://github.com/ozak/georasters.

[143] J. Paola and R. Schowengerdt. “A detailed comparison of backpropagation neural network and maximum-likelihood classifiers for urban land use classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 33.4 (July 1995), pp. 981–996. DOI: 10.1109/36.406684. URL: https://doi.org/10.1109%2F36.406684.

[144] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: `http://arxiv.org/pdf/1201.0490v2.pdf`.

[145] O. A. B. Penatti, K. Nogueira, and J. A. dos Santos. “Do deep features generalize from everyday objects to remote sensing and aerial scenes domains?” In: *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2015. DOI: `10.1109/cvprw.2015.7301382`. URL: `https://doi.org/10.1109%2Fcvprw.2015.7301382`.

[146] O. A. B. Penatti, K. Nogueira, and J. A. dos Santos. *PatreoDatasets*. 2016. URL: `http://www.patreo.dcc.ufmg.br/category/downloads/datasets/`.

[147] F. Perez and B. E. Granger. “IPython: A System for Interactive Scientific Computing”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 21–29. DOI: `10.1109/mcse.2007.53`. URL: `http://dx.doi.org/10.1109/MCSE.2007.53`.

[148] B. Polyak. “Some methods of speeding up the convergence of iteration methods”. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (Jan. 1964), pp. 1–17. DOI: `10.1016/0041-5553(64)90137-5`. URL: `https://doi.org/10.1016%2F0041-5553%2864%2990137-5`.

[149] J. Porway, K. Wang, B. Yao, and S. C. Zhu. “A hierarchical and contextual model for aerial image understanding”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2008. DOI: `10.1109/cvpr.2008.4587359`. URL: `https://doi.org/10.1109%2Fcvpr.2008.4587359`.

[150] N. T. Quang, N. T. Thuy, D. V. Sang, and H. T. T. Binh. “An Efficient Framework for Pixel-wise Building Segmentation from Aerial Images”. In: *Proceedings of the Sixth International Symposium on Information and Communication Technology - SoICT 2015*. Association for Computing Machinery (ACM), 2015. DOI: `10.1145/2833258.2833272`. URL: `https://doi.org/10.1145%2F2833258.2833272`.

[151] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. “Efficient learning of sparse representations with an energy-based model”. In: *Proceedings of 20th International Conference on Neural Information Processing Systems*. 2007. URL: `https://papers.nips.cc/paper/3112-efficient-learning-of-sparse-representations-with-an-energy-based-model.pdf`.

[152] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press, 1996. ISBN: 0-521-46086-7.

[153] V. Risojevic. “Analysis of learned features for remote sensing image classification”. In: *2016 13th Symposium on Neural Networks and Applications (NEUREL)*. IEEE, Nov. 2016. DOI: `10.1109/neurel.2016.7800145`. URL: `https://doi.org/10.1109%2Fneurel.2016.7800145`.

[154] V. Risojevic and Z. Babic. "Unsupervised Quaternion Feature Learning for Remote Sensing Image Classification". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 9.4 (Apr. 2016), pp. 1521–1531. DOI: 10.1109/jstars.2015.2513898. URL: https://doi.org/10.1109%2Fjstars.2015.2513898.

[155] V. Risojević, S. Momić, and Z. Babić. "Gabor Descriptors for Aerial Image Classification". In: *Adaptive and Natural Computing Algorithms*. Springer Berlin Heidelberg, 2011, pp. 51–60. DOI: 10.1007/978-3-642-20267-4_6. URL: https://doi.org/10.1007%2F978-3-642-20267-4_6.

[156] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. URL: http://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf.

[157] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1". In: ed. by D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning Internal Representations by Error Propagation, pp. 318–362. ISBN: 0-262-68053-X. URL: http://www.cnbc.cmu.edu/~plaut/IntroPDP/papers/RumelhartETAL86.backprop.pdf.

[158] O. Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Apr. 2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007%2Fs11263-015-0816-y.

[159] O. Russakovsky et al. *Large Scale Visual Recognition Challenge 2016 (ILSVRC2016)*. 2016. URL: http://image-net.org/challenges/LSVRC/2016/results (visited on 05/15/2017).

[160] J. Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003. URL: https://doi.org/10.1016%2Fj.neunet.2014.09.003.

[161] M. Schuster and K. K. Paliwal. "Bidirectional recurrent neural networks". In: *Neural Computation* 45.11 (1997), pp. 2673–2681. URL: https://maxwell.ict.griffith.edu.au/spl/publications/papers/ieeesp97_schuster.pdf.

[162] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". In: *CoRR* abs/1610.02391 (2016). URL: http://arxiv.org/abs/1610.02391.

[163] T. Sercu, C. Puhrsch, B. Kingsbury, and Y. LeCun. "Very Deep Multilingual Convolutional Neural Networks for LVCSR". In: *CoRR* abs/1509.08967 (2015). URL: http://arxiv.org/abs/1509.08967.

[164] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *CoRR* abs/1312.6229 (2013). URL: http://arxiv.org/abs/1312.6229.

[165] E. Shelhamer, J. Long, and T. Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *CoRR* abs/1605.06211 (2016). URL: `http://arxiv.org/abs/1605.06211`.

[166] G. Sheng, W. Yang, T. Xu, and H. Sun. “High-resolution satellite scene classification using a sparse coding based multiple feature combination”. In: *International Journal of Remote Sensing* 33.8 (Apr. 2012), pp. 2395–2412. DOI: `10.1080/01431161.2011.608740`. URL: `https://doi.org/10.1080%2F01431161.2011.608740`.

[167] J. Sherrah. “Fully Convolutional Networks for Dense Semantic Labelling of High-Resolution Aerial Imagery”. In: *CoRR* abs/1606.02585 (2016). URL: `http://arxiv.org/abs/1606.02585`.

[168] K. Simonyan, A. Vedaldi, and A. Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *CoRR* abs/1312.6034 (2013). URL: `http://arxiv.org/abs/1312.6034`.

[169] K. Simonyan and A. Zisserman. “Two-Stream Convolutional Networks for Action Recognition in Videos”. In: *CoRR* abs/1406.2199 (2014). URL: `http://arxiv.org/abs/1406.2199`.

[170] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). URL: `http://arxiv.org/abs/1409.1556`.

[171] P. Smolensky. “Information processing in dynamical systems: Foundations of harmony theory”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Vol. 1. 1986, pp. 194–281. URL: `http://smolensky.johnshopkins.edu/docs/Smolensky86PDPch6`.

[172] X. Song, G. Fan, and M. Rao. “Automatic CRP mapping using nonparametric machine learning approaches”. In: *IEEE Transactions on Geoscience and Remote Sensing* 43.4 (Apr. 2005), pp. 888–897. DOI: `10.1109/tgrs.2005.844031`. URL: `https://doi.org/10.1109%2Ftgrs.2005.844031`.

[173] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. “Striving for Simplicity: The All Convolutional Net”. In: *CoRR* abs/1412.6806 (2014). URL: `http://arxiv.org/abs/1412.6806`.

[174] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435. URL: `https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf`.

[175] R. O. Stehling, M. A. Nascimento, and A. X. Falcão. “A compact and efficient image retrieval approach based on border/interior pixel classification”. In: *Proceedings of the eleventh international conference on Information and knowledge management - CIKM '02*. ACM Press, 2002. DOI: `10.1145/584792.584812`. URL: `https://doi.org/10.1145%2F584792.584812`.

[176] N. G. Survey. *Airport Aeference Point Computation.* URL: https://www.ngs.noaa.gov/AERO/arpcomp/arpframe.html (visited on 05/15/2017).

[177] U. G. Survey. *The National Map.* 2017. URL: https://nationalmap.gov/ (visited on 05/15/2017).

[178] C. Szegedy, S. Ioffe, and V. Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *CoRR* abs/1602.07261 (2016). URL: http://arxiv.org/abs/1602.07261.

[179] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298594. URL: https://doi.org/10.1109%2Fcvpr.2015.7298594.

[180] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). URL: http://arxiv.org/abs/1512.00567.

[181] S. Targ, D. Almeida, and K. Lyman. "Resnet in Resnet: Generalizing Residual Architectures". In: *CoRR* abs/1603.08029 (2016). URL: http://arxiv.org/abs/1603.08029.

[182] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: http://arxiv.org/abs/1605.02688.

[183] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude.* COURSERA: Neural Networks for Machine Learning. 2012. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[184] F. van Veen. *The neural network zoo.* [Online; accessed 2017-02-28]. 2016. URL: http://www.asimovinstitute.org/neural-network-zoo/.

[185] V. Vijayaraj, A. Cheriyadat, P. Sallee, B. Colder, R. Vatsavai, E. Bright, and B. Bhaduri. "Overhead image statistics". In: *2008 37th IEEE Applied Imagery Pattern Recognition Workshop.* IEEE, 2008. DOI: 10.1109/aipr.2008.4906471. URL: https://doi.org/10.1109%2Faipr.2008.4906471.

[186] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. "Extracting and Composing Robust Features with Denoising Autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning.* ICML '08. Helsinki, Finland: ACM, 2008, pp. 1096–1103. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390294. URL: http://doi.acm.org/10.1145/1390156.1390294.

[187] S. van der Walt, S. C. Colbert, and G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science & Engineering* 13.2 (Mar. 2011), pp. 22–30. DOI: 10.1109/mcse.2011.37. URL: http://dx.doi.org/10.1109/MCSE.2011.37.

[188] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. “Regularization of Neural Networks Using Dropconnect”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA: JMLR.org, 2013, pp. III-1058–III-1066. URL: `http://cs.nyu.edu/~wanli/dropc/dropc.pdf`.

[189] M. Waskom et al. *seaborn: v0.7.1 (June 2016)*. June 2016. DOI: `10.5281/zenodo.54844`. URL: `https://doi.org/10.5281/zenodo.54844`.

[190] P. J. Werbos. “Applications of advances in nonlinear sensitivity analysis”. In: *System Modeling and Optimization*. Springer-Verlag, 1981, pp. 762–770. DOI: `10.1007/bfb0006203`. URL: `https://doi.org/10.1007%2Fbfb0006203`.

[191] Wikipedia contributors. *Wikipedia*. `https://en.wikipedia.org` . 2017.

[192] N. Willis. *OpenStreetMap project completes import of United States TIGER data*. 2008. URL: `https://www.linux.com/news/openstreetmap-project-completes-import-united-states-tiger-data` (visited on 05/15/2017).

[193] G. Xia, J. Hu, F. Hu, B. Shi, X. Bai, Y. Zhong, and L. Zhang. *AID: A Benchmark Dataset for Performance Evaluation of Aerial Scene Classification*. 2016. URL: `http://www.lmars.whu.edu.cn/xia/AID-project.html` (visited on 05/15/2017).

[194] G. Xia, J. Hu, F. Hu, B. Shi, X. Bai, Y. Zhong, and L. Zhang. “AID: A Benchmark Dataset for Performance Evaluation of Aerial Scene Classification”. In: *CoRR* abs/1608.05167 (2016). URL: `http://arxiv.org/abs/1608.05167`.

[195] G.-S. Xia, W. Yang, J. Delon, Y. Gousseau, H. Sun, and H. Maître. “Structural High-resolution Satellite Image Indexing”. In: *ISPRS TC VII Symposium - 100 Years ISPRS*. Ed. by B. Wagner W. Székely. Vol. XXXVIII. Vienna, Austria, July 2010, pp. 298–303. URL: `https://hal.archives-ouvertes.fr/hal-00458685`.

[196] B. Xu, N. Wang, T. Chen, and M. Li. “Empirical Evaluation of Rectified Activations in Convolutional Network”. In: *CoRR* abs/1505.00853 (2015). URL: `http://arxiv.org/abs/1505.00853`.

[197] Y. Yang and S. Newsam. “Bag-of-visual-words and spatial extensions for land-use classification”. In: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS ’10*. ACM Press, 2010. DOI: `10.1145/1869790.1869829`. URL: `https://doi.org/10.1145%2F1869790.1869829`.

[198] Y. Yang and S. Newsam. “Comparing SIFT descriptors and gabor texture features for classification of remote sensed imagery”. In: *2008 15th IEEE International Conference on Image Processing*. IEEE, 2008. DOI: `10.1109/icip.2008.4712139`. URL: `https://doi.org/10.1109%2Ficip.2008.4712139`.

[199] Y. Yang and S. Newsam. *UC Merced Land Use Dataset*. 2010. URL: `http://vision.ucmerced.edu/datasets/landuse.html` (visited on 02/28/2017).

[200] S. Zagoruyko and N. Komodakis. “Wide Residual Networks”. In: *CoRR* abs/1605.07146 (2016). URL: `http://arxiv.org/abs/1605.07146`.

[201] W. Zaremba and I. Sutskever. *Reinforcement Learning Neural Turing Machines - Revised*. 2015. eprint: `arXiv:1505.00521`. URL: `https://arxiv.org/abs/1505.00521`.

[202] M. D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701 (2012). URL: `http://arxiv.org/abs/1212.5701`.

[203] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *CoRR* abs/1311.2901 (2013). URL: `http://arxiv.org/abs/1311.2901`.

[204] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. "Deconvolutional networks". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Institute of Electrical and Electronics Engineers (IEEE), June 2010. DOI: `10.1109/cvpr.2010.5539957`. URL: `https://doi.org/10.1109%2Fcvpr.2010.5539957`.

[205] B. Zhao, Y. Zhong, G.-S. Xia, and L. Zhang. "Dirichlet-Derived Multiple Topic Scene Classification Model for High Spatial Resolution Remote Sensing Imagery". In: *IEEE Transactions on Geoscience and Remote Sensing* 54.4 (Apr. 2016), pp. 2108–2123. DOI: `10.1109/tgrs.2015.2496185`. URL: `https://doi.org/10.1109%2Ftgrs.2015.2496185`.

[206] L. Zhao, P. Tang, and L. Huo. "Feature significance-based multibag-of-visual-words model for remote sensing image scene classification". In: *Journal of Applied Remote Sensing* 10.3 (July 2016), p. 035004. DOI: `10.1117/1.jrs.10.035004`. URL: `https://doi.org/10.1117%2F1.jrs.10.035004`.

[207] Y. Zhong, F. Fei, Y. Liu, B. Zhao, H. Jiao, and L. Zhang. "SatCNN: satellite image dataset classification using agile convolutional neural networks". In: *Remote Sensing Letters* 8.2 (Oct. 2016), pp. 136–145. DOI: `10.1080/2150704x.2016.1235299`. URL: `https://doi.org/10.1080%2F2150704x.2016.1235299`.

[208] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba. "Learning Deep Features for Discriminative Localization". In: *CoRR* abs/1512.04150 (2015). URL: `http://arxiv.org/abs/1512.04150`.

[209] Zhou and Chellappa. "Computation of optical flow using a neural network". In: *IEEE International Conference on Neural Networks*. IEEE, 1988. DOI: `10.1109/icnn.1988.23914`. URL: `https://doi.org/10.1109%2Ficnn.1988.23914`.

Appendices

Appendix A

Imagery examples

A.1 Examples for class *airport*

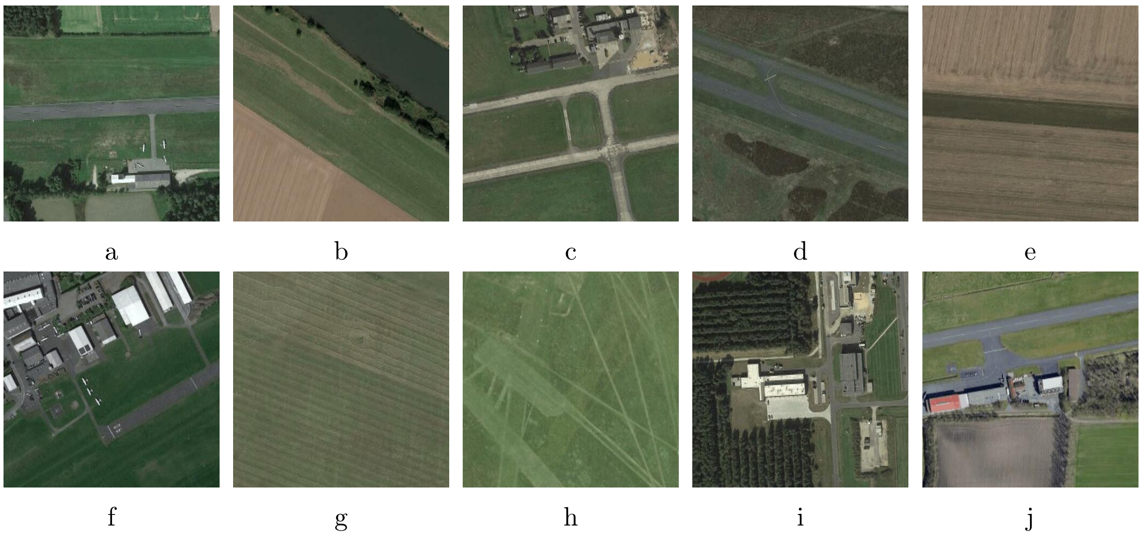


Figure A.1: Examples for class *airport*

A.2 Examples for class *baseball*

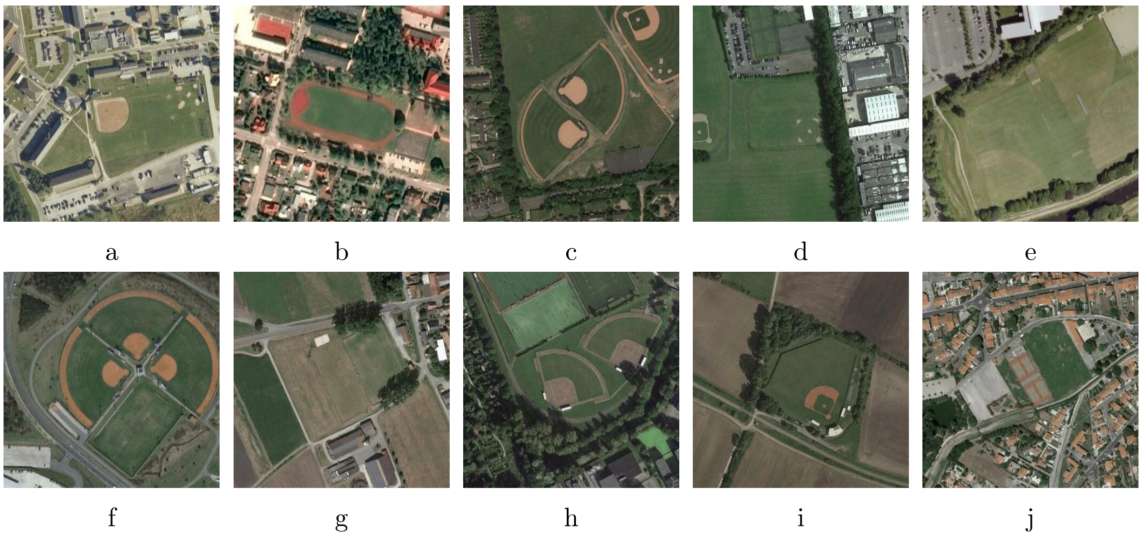


Figure A.2: Examples for class *baseball*

A.3 Examples for class *bridge*

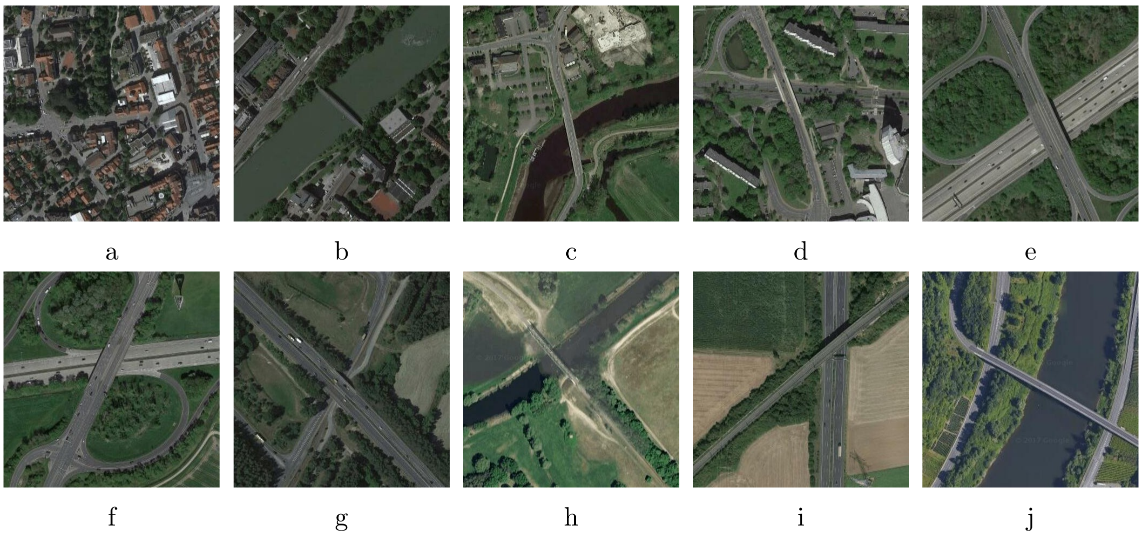


Figure A.3: Examples for class *bridge*

A.4 Examples for class *building*

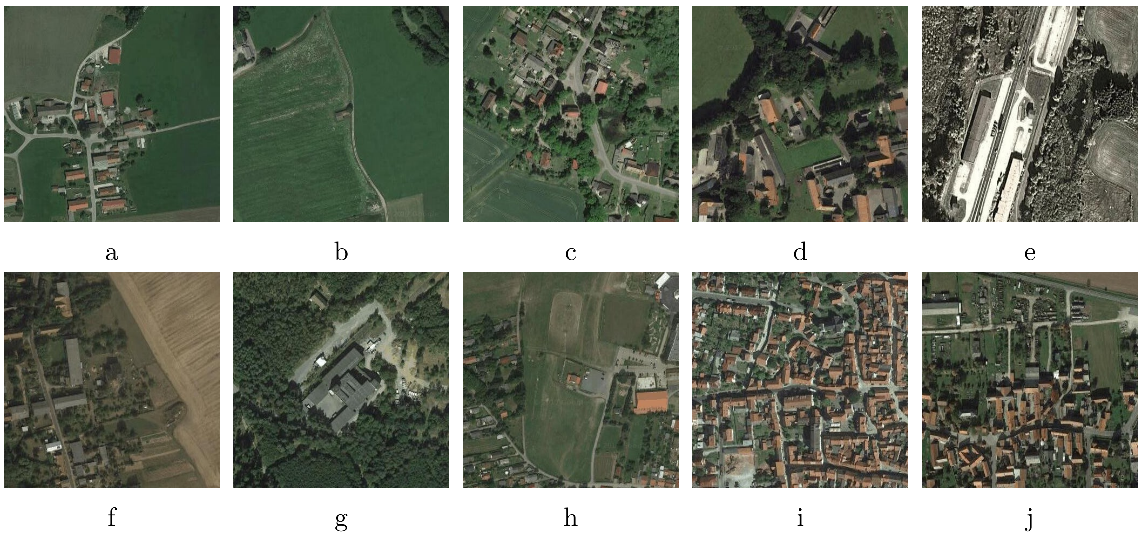


Figure A.4: Examples for class *building*

A.5 Examples for class *castle*

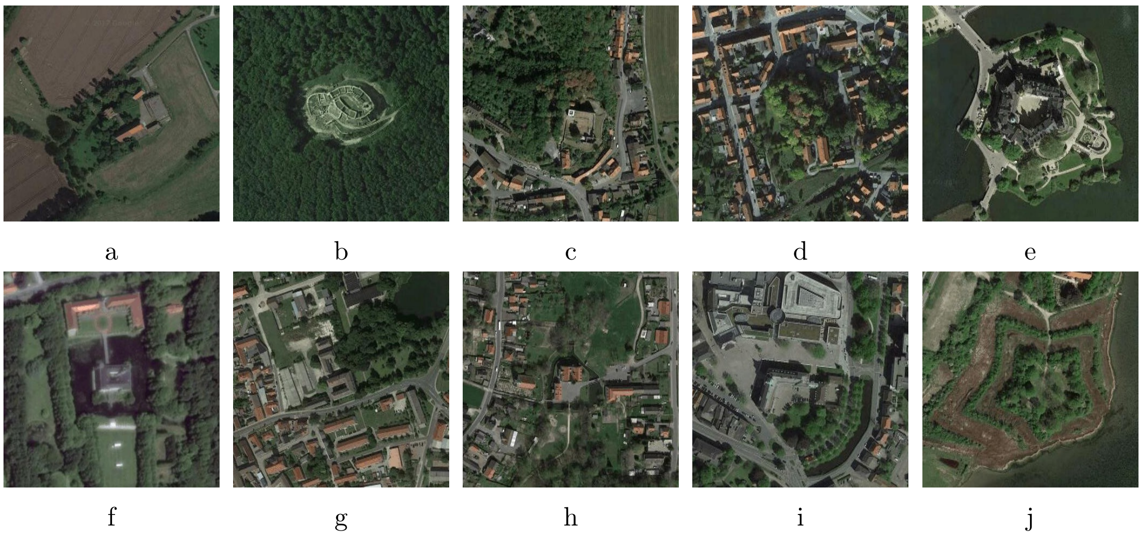


Figure A.5: Examples for class *castle*

A.6 Examples for class *cemetery*

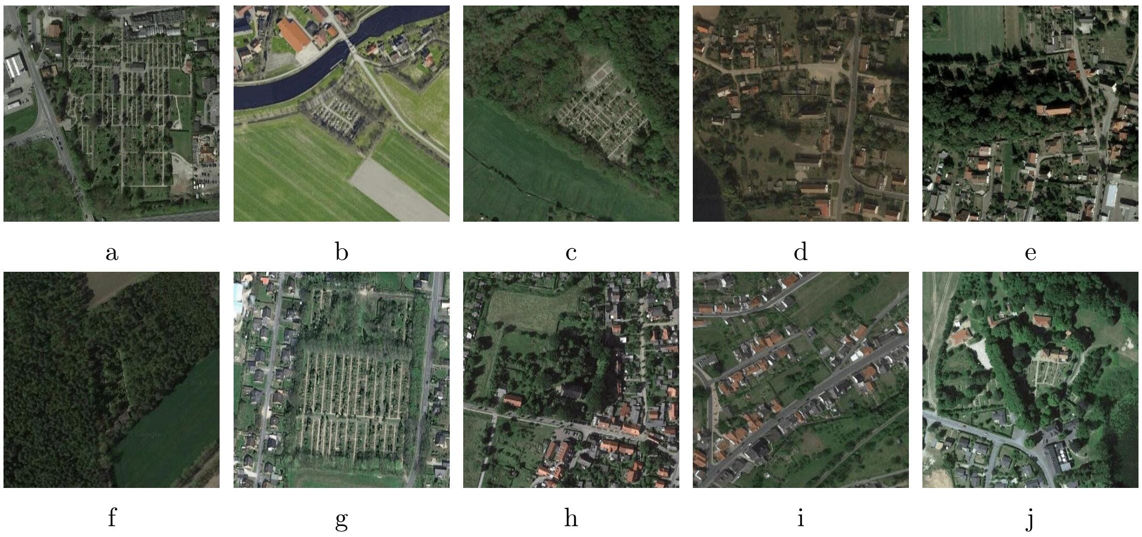


Figure A.6: Examples for class *cemetery*

A.7 Examples for class *chimney*

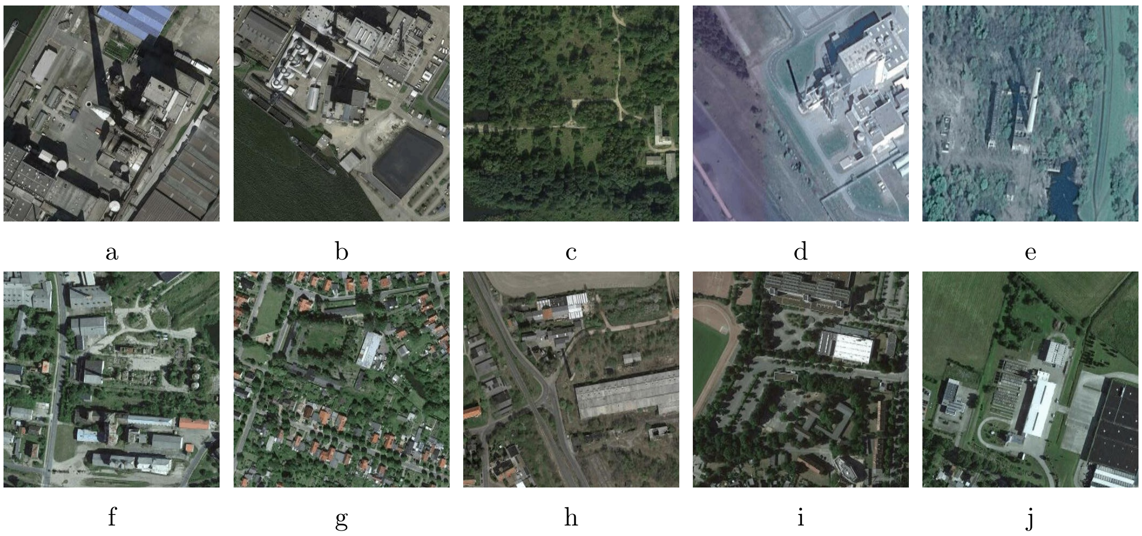


Figure A.7: Examples for class *chimney*

A.8 Examples for class *church*

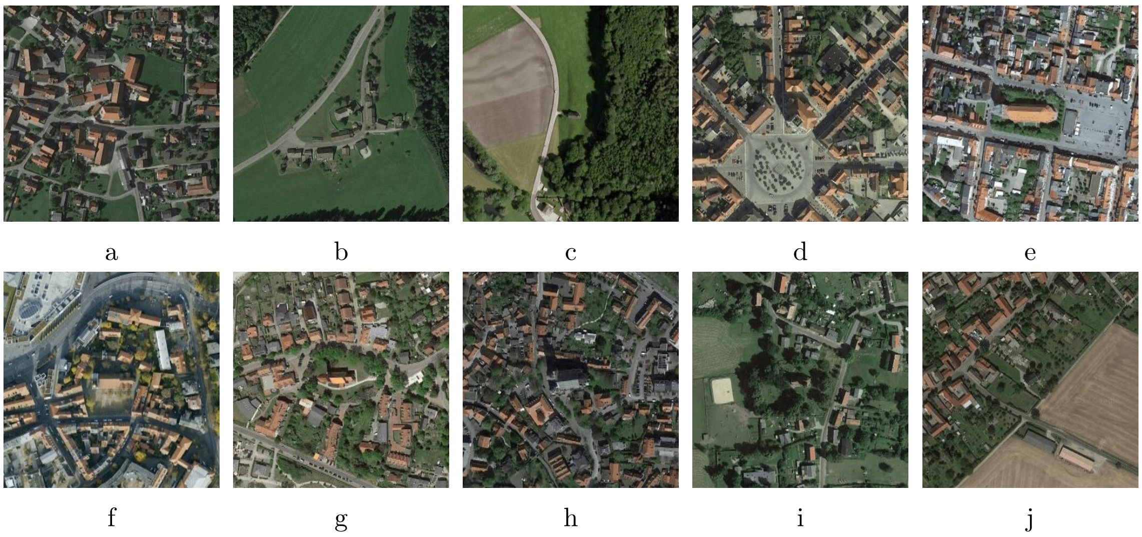


Figure A.8: Examples for class *church*

A.9 Examples for class *coastline*

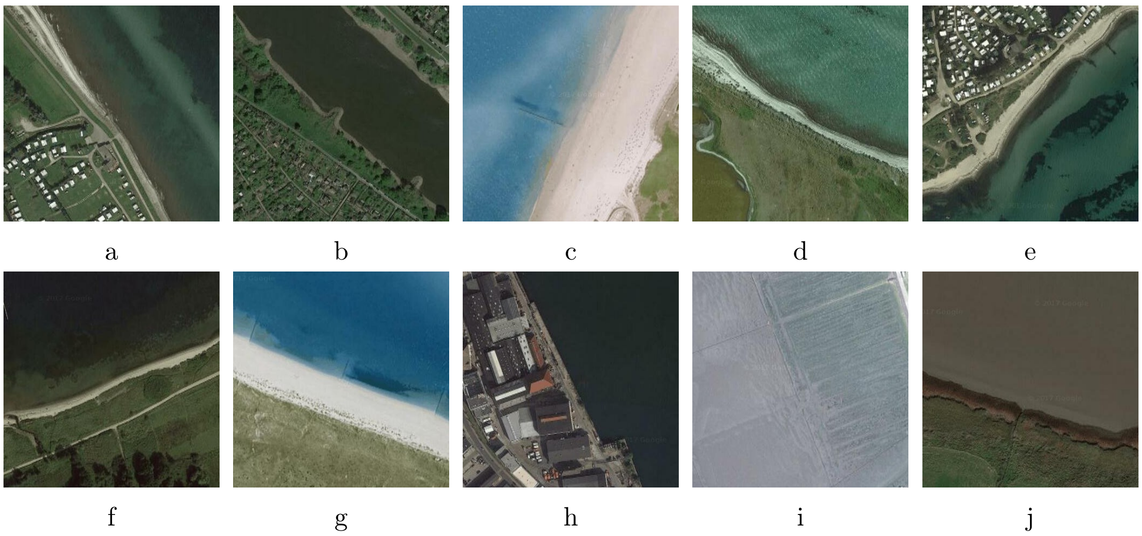


Figure A.9: Examples for class *coastline*

A.10 Examples for class *cooling_ tower*

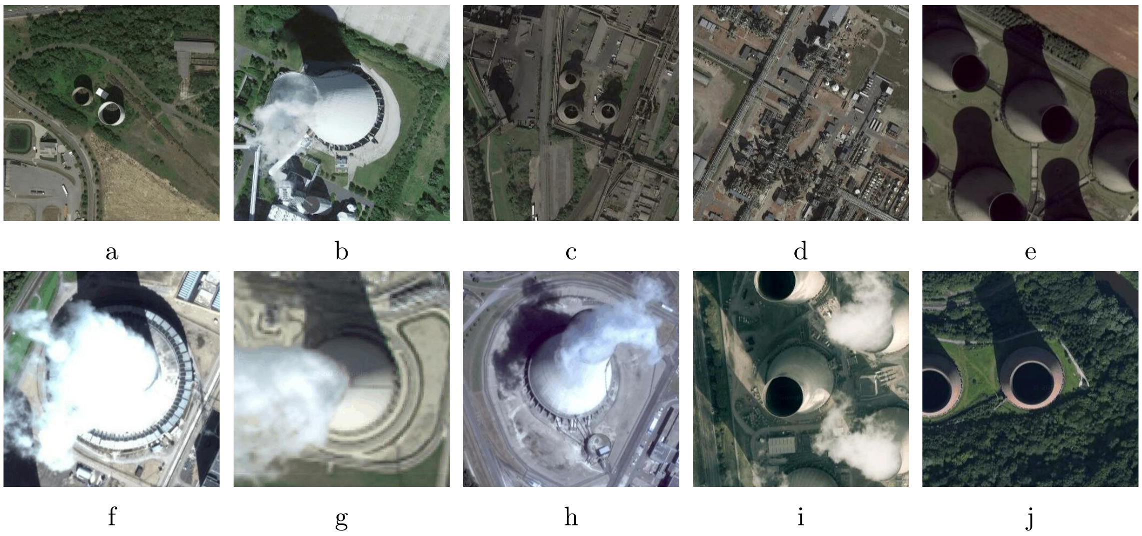


Figure A.10: Examples for class *cooling_ tower*

A.11 Examples for class *dam*

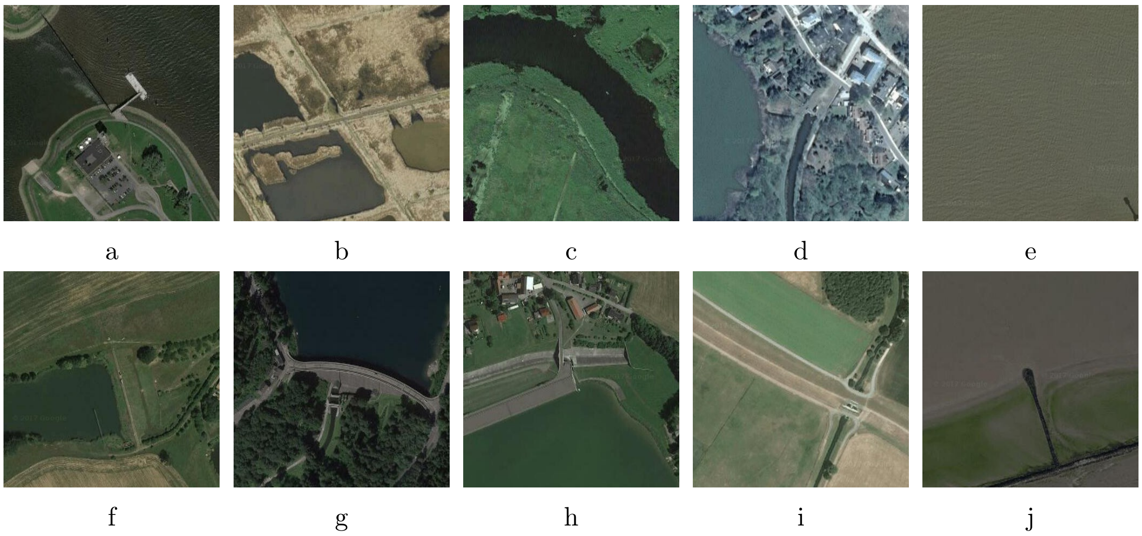


Figure A.11: Examples for class *dam*

A.12 Examples for class *farmland*

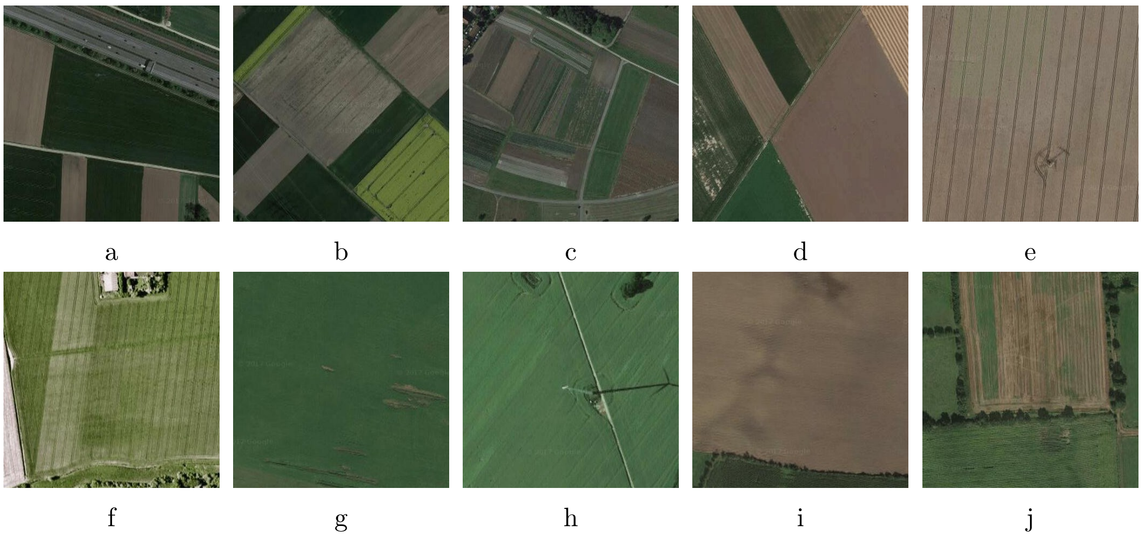


Figure A.12: Examples for class *farmland*

A.13 Examples for class *forest*

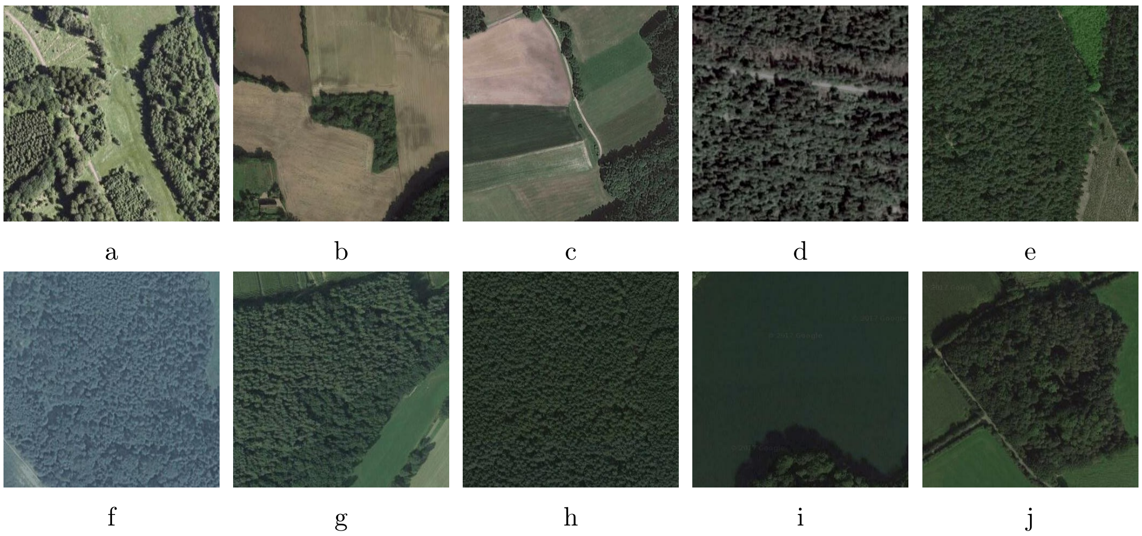


Figure A.13: Examples for class *forest*

A.14 Examples for class *fort*

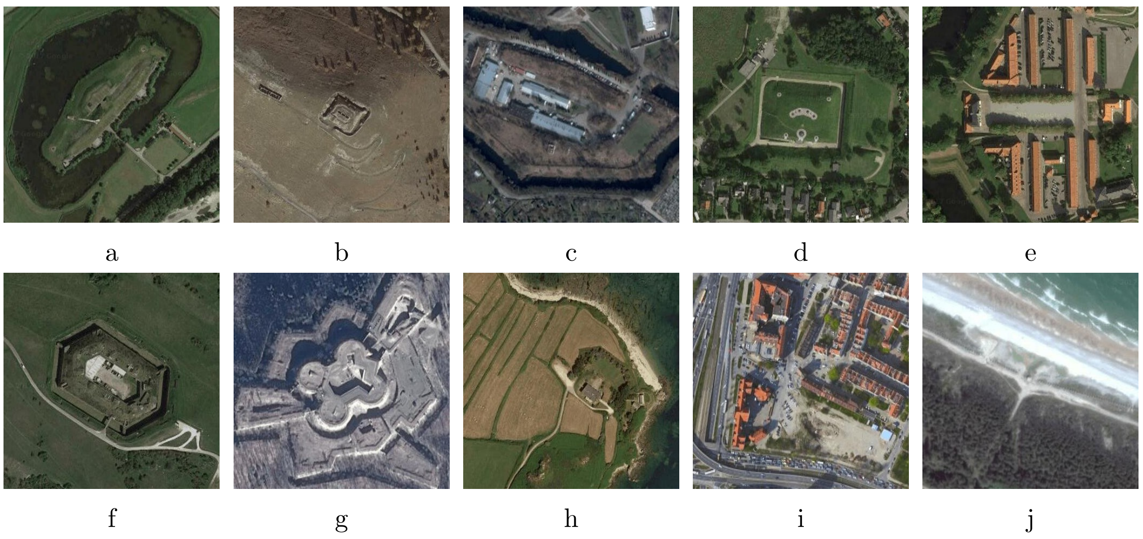


Figure A.14: Examples for class *fort*

A.15 Examples for class *fuel_ station*

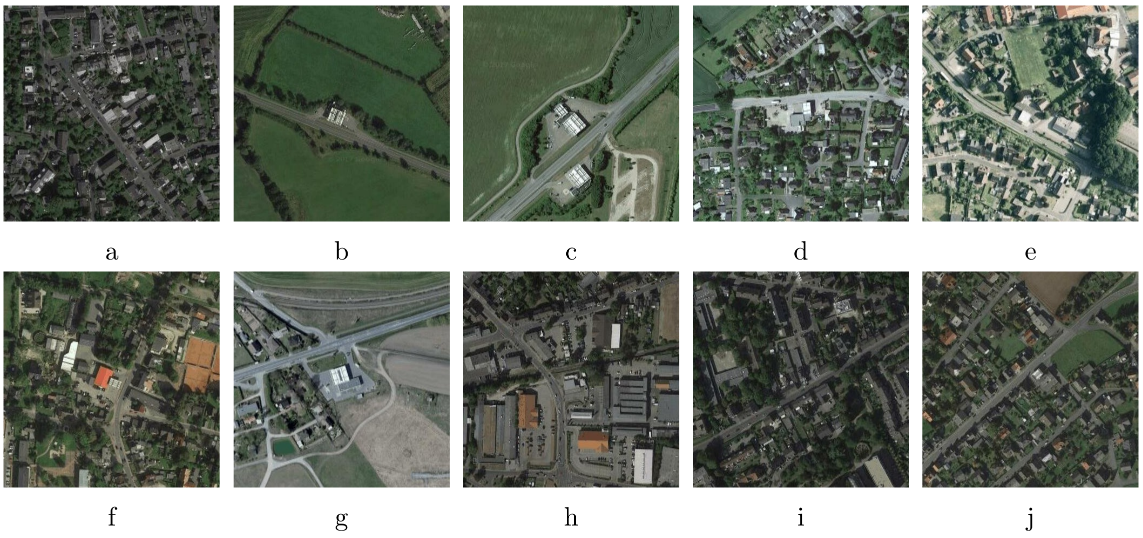


Figure A.15: Examples for class *fuel_ station*

A.16 Examples for class *golf*

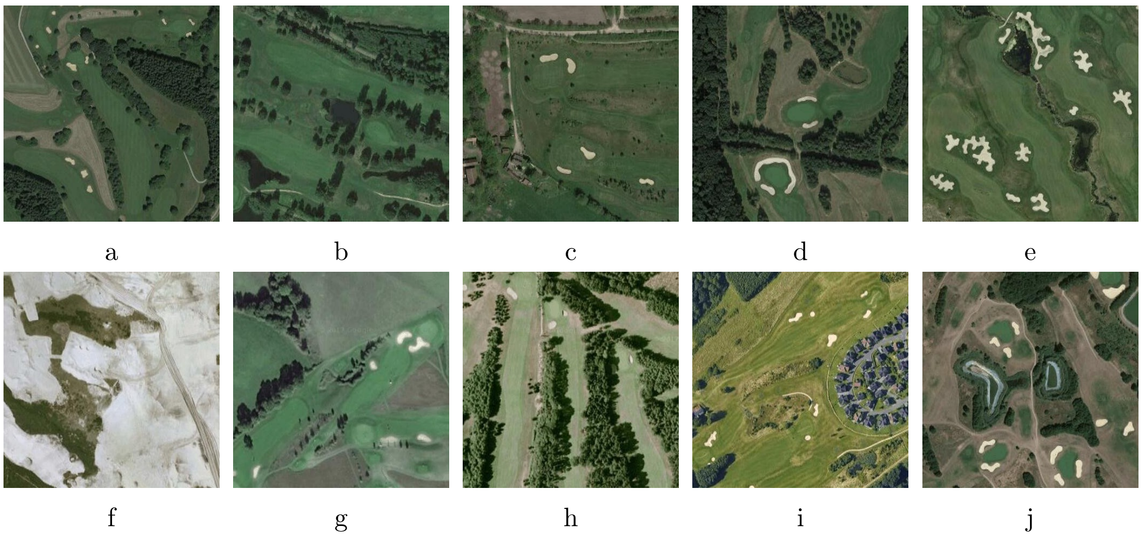


Figure A.16: Examples for class *golf*

A.17 Examples for class *greenhouse*

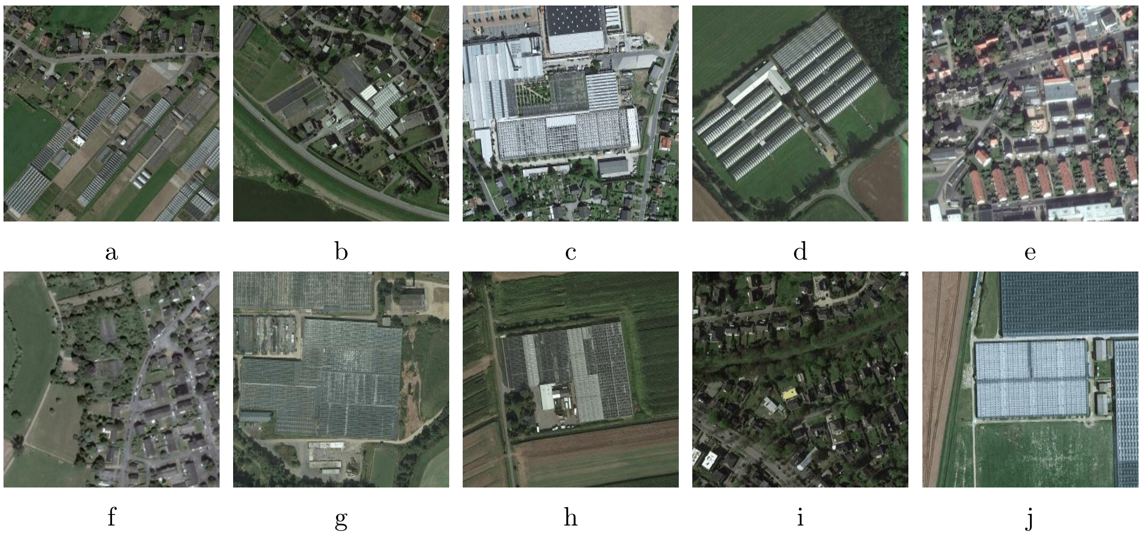


Figure A.17: Examples for class *greenhouse*

A.18 Examples for class *harbour*

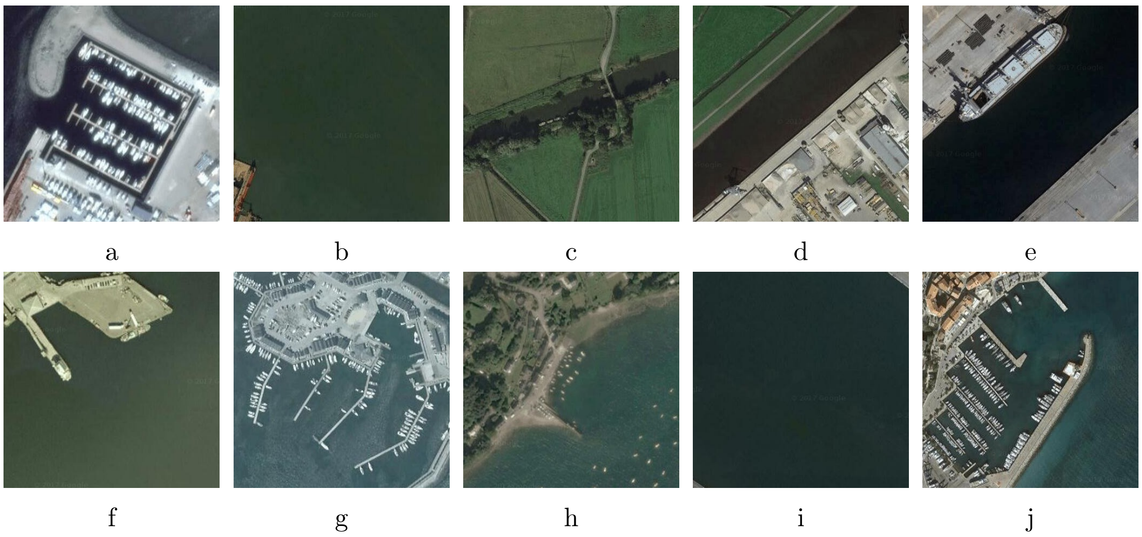


Figure A.18: Examples for class *harbour*

A.19 Examples for class *helipad*

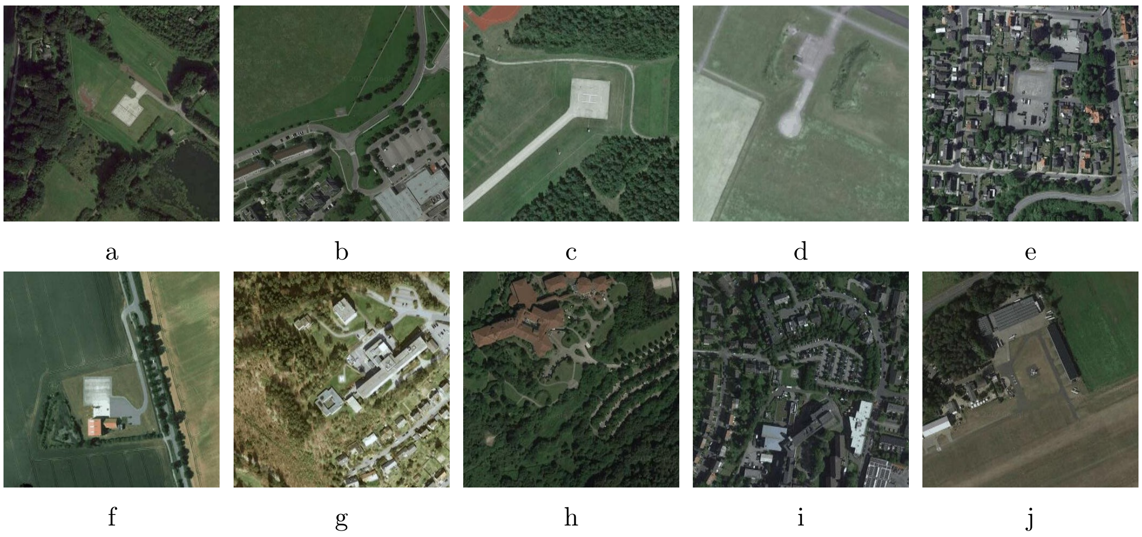


Figure A.19: Examples for class *helipad*

A.20 Examples for class *highway*

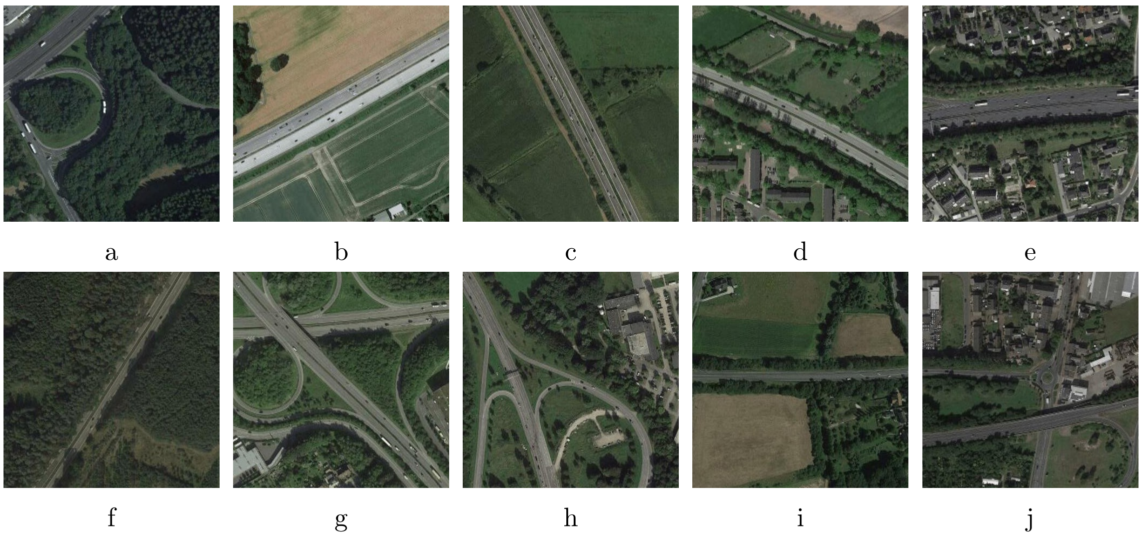


Figure A.20: Examples for class *highway*

A.21 Examples for class *lake*

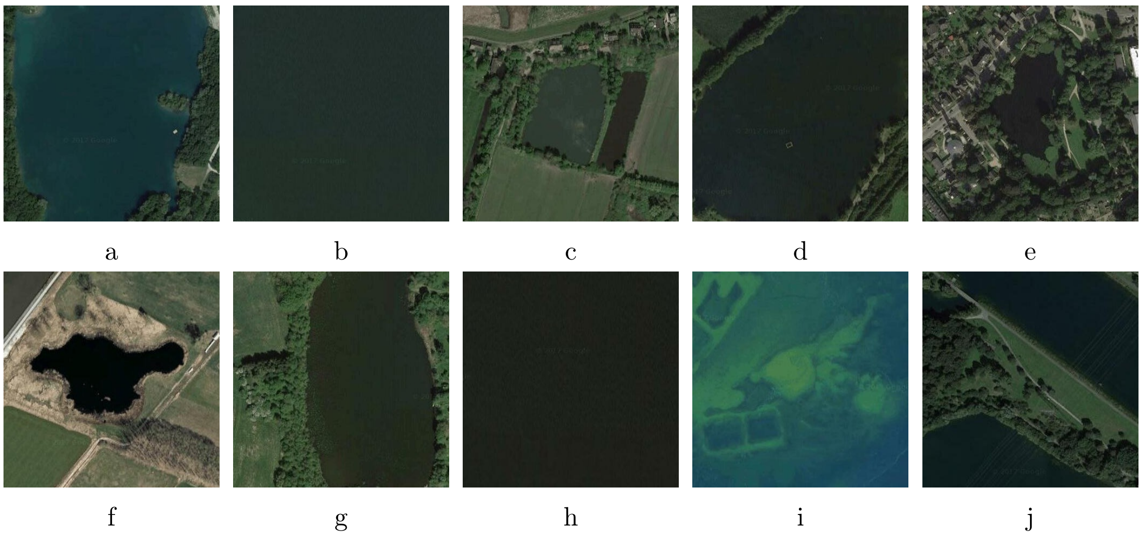


Figure A.21: Examples for class *lake*

A.22 Examples for class *landfill*

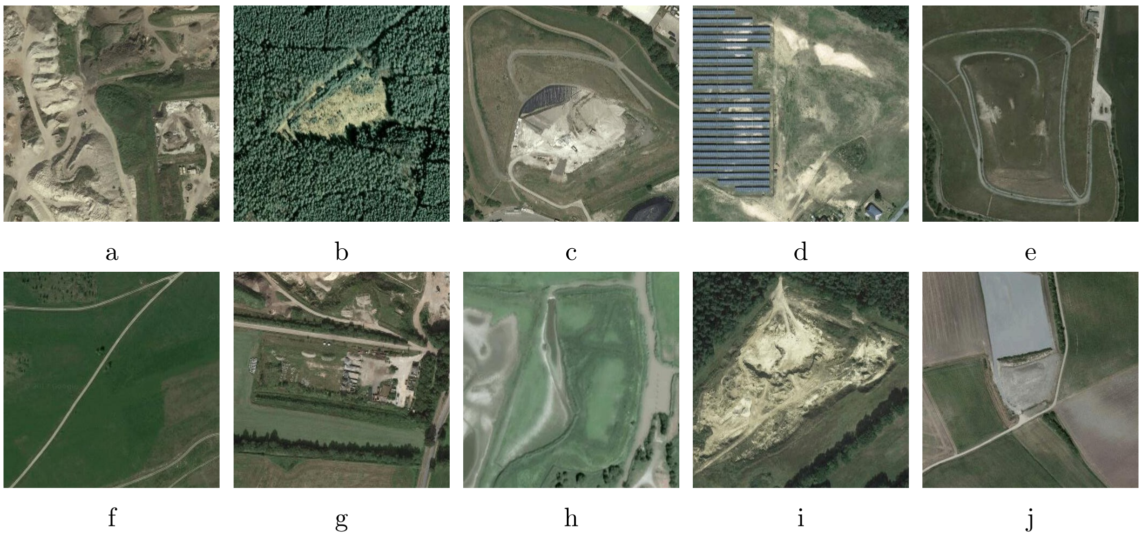


Figure A.22: Examples for class *landfill*

A.23 Examples for class *marina*

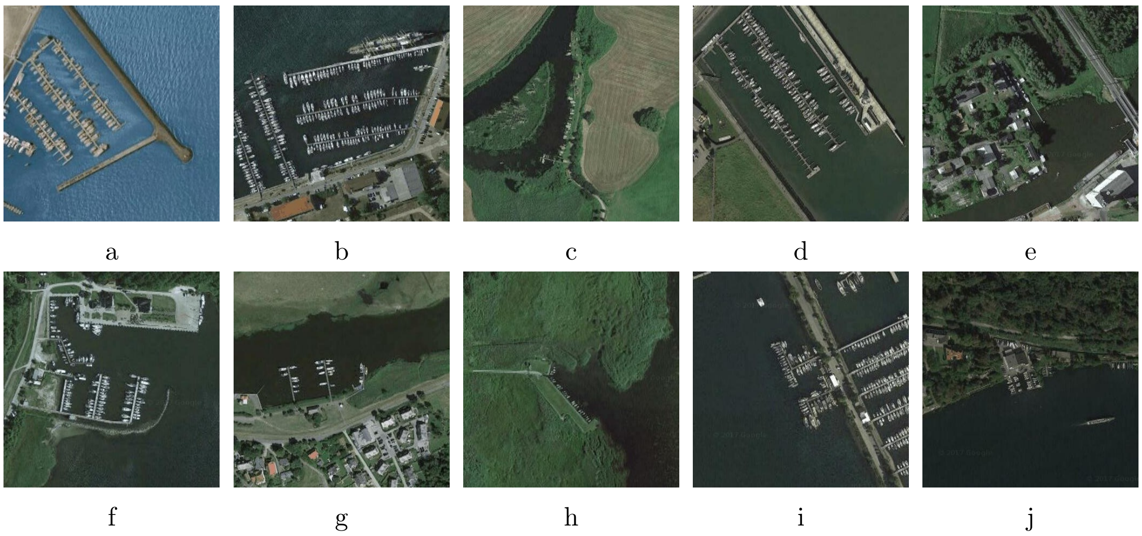


Figure A.23: Examples for class *marina*

A.24 Examples for class *meadow*

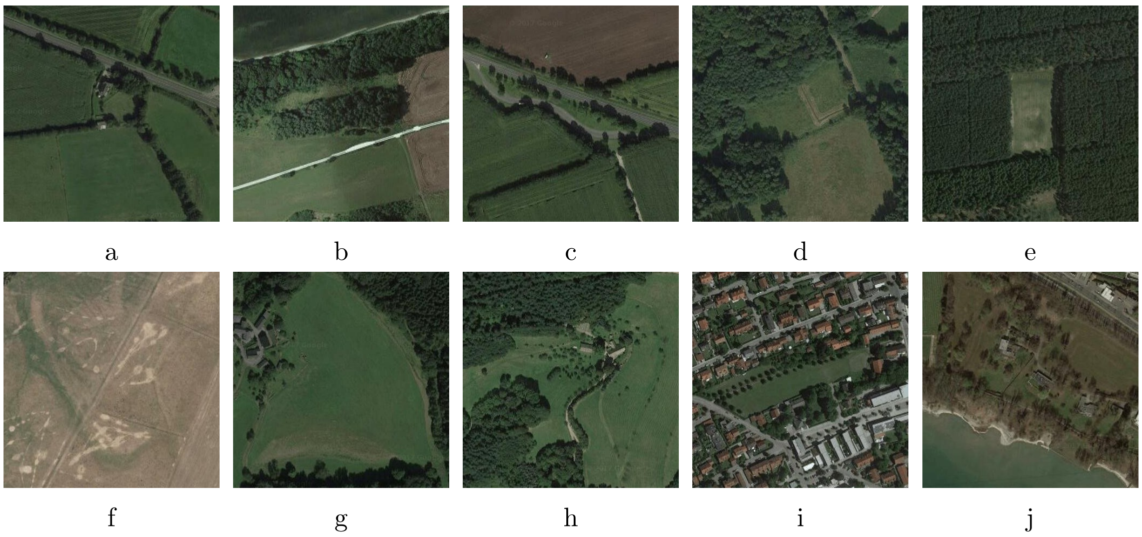


Figure A.24: Examples for class *meadow*

A.25 Examples for class *orchard*

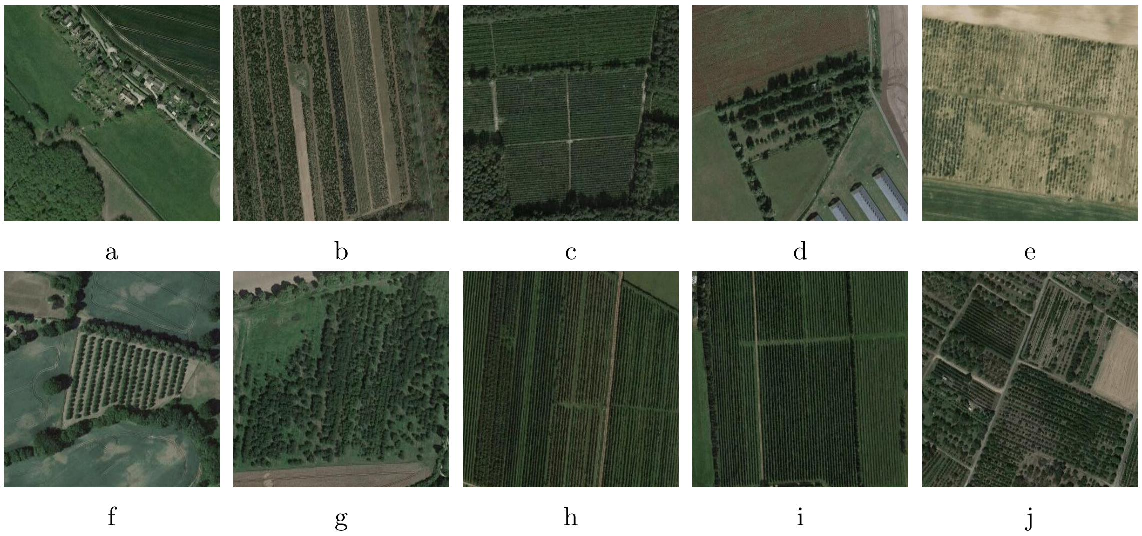


Figure A.25: Examples for class *orchard*

A.26 Examples for class *parking*

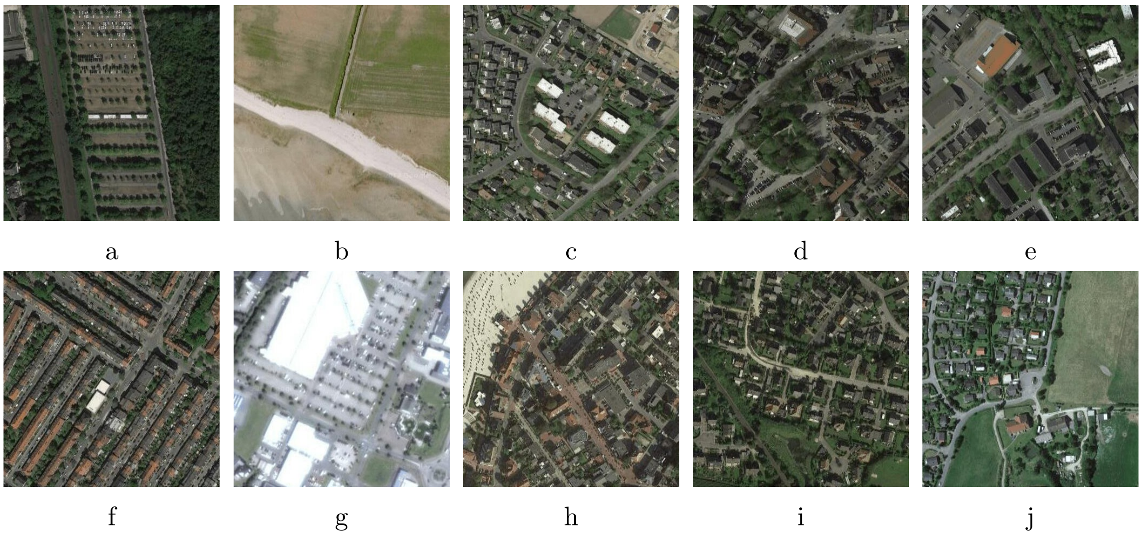


Figure A.26: Examples for class *parking*

A.27 Examples for class *pipeline*

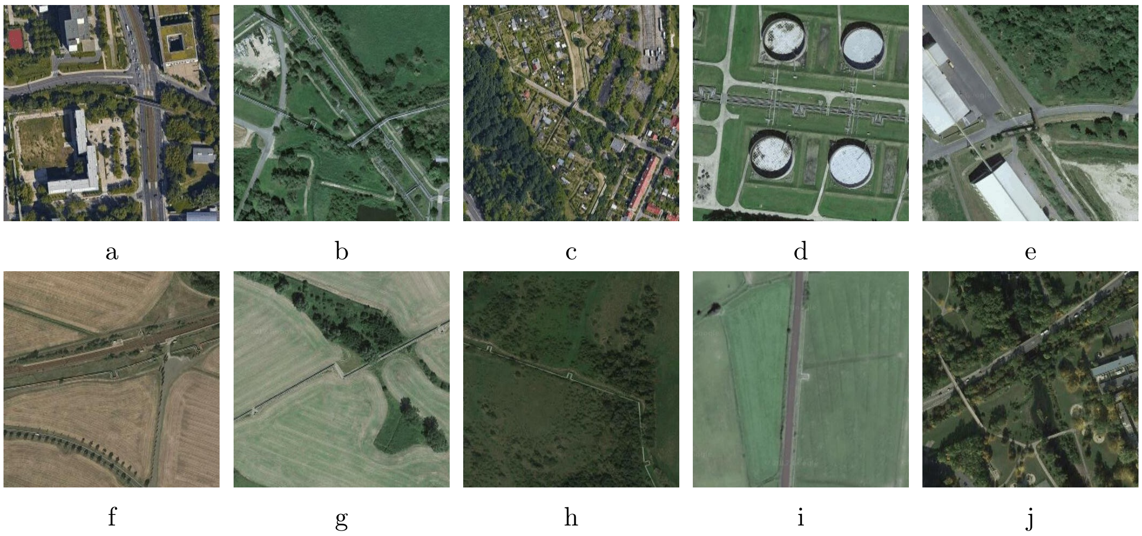


Figure A.27: Examples for class *pipeline*

A.28 Examples for class *power_ coal*

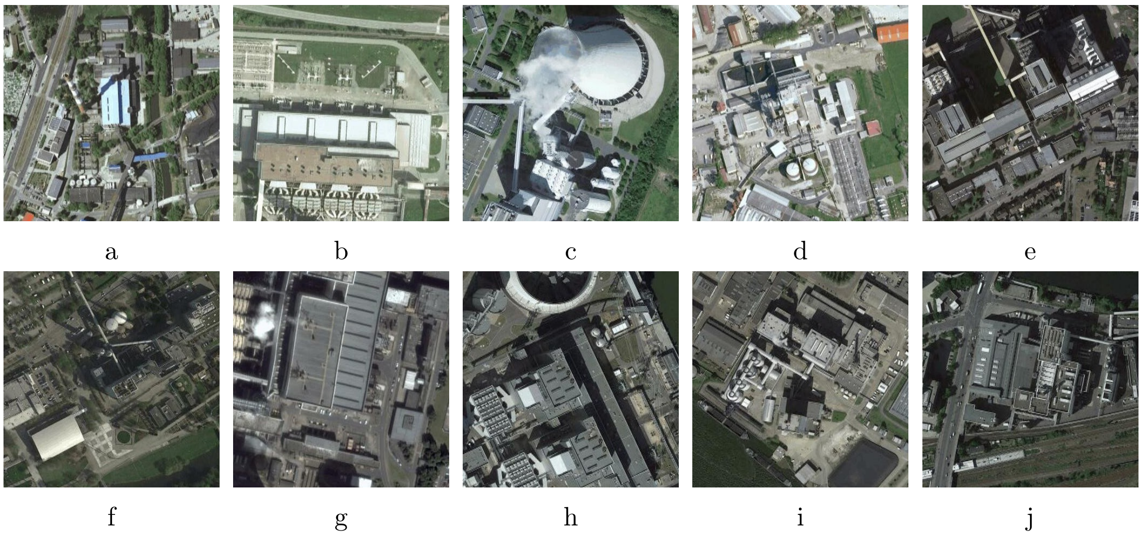


Figure A.28: Examples for class *power_ coal*

A.29 Examples for class *power_ hydro*

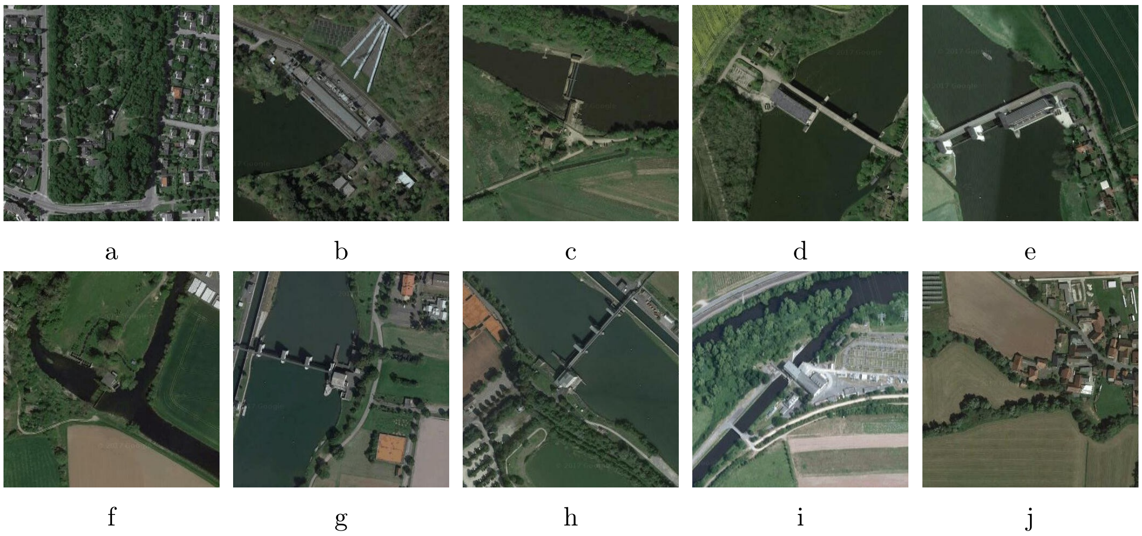


Figure A.29: Examples for class *power_ hydro*

A.30 Examples for class *power_ nuclear*

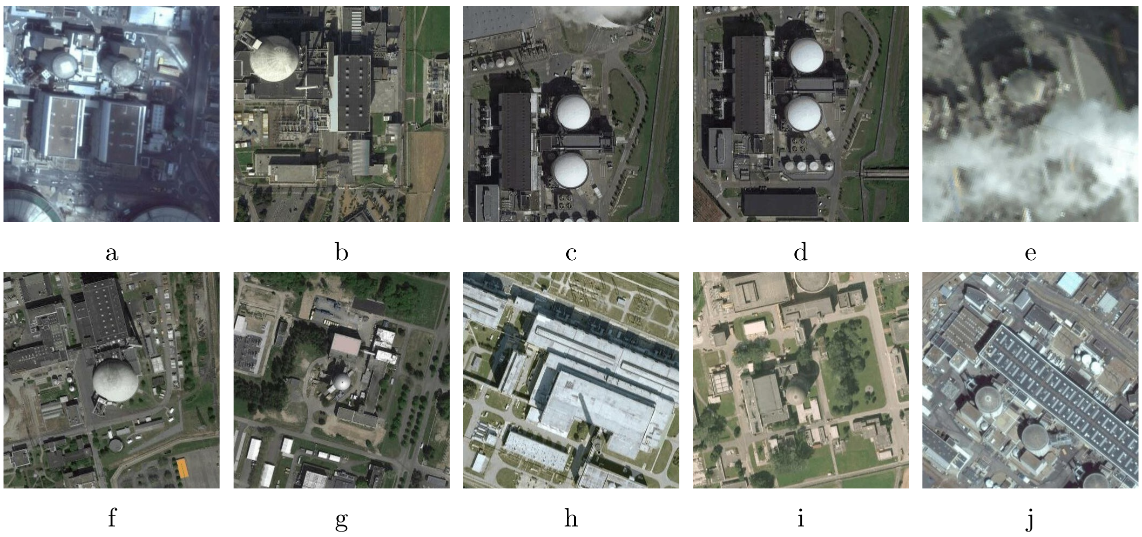


Figure A.30: Examples for class *power_ nuclear*

A.31 Examples for class *power_ oil*

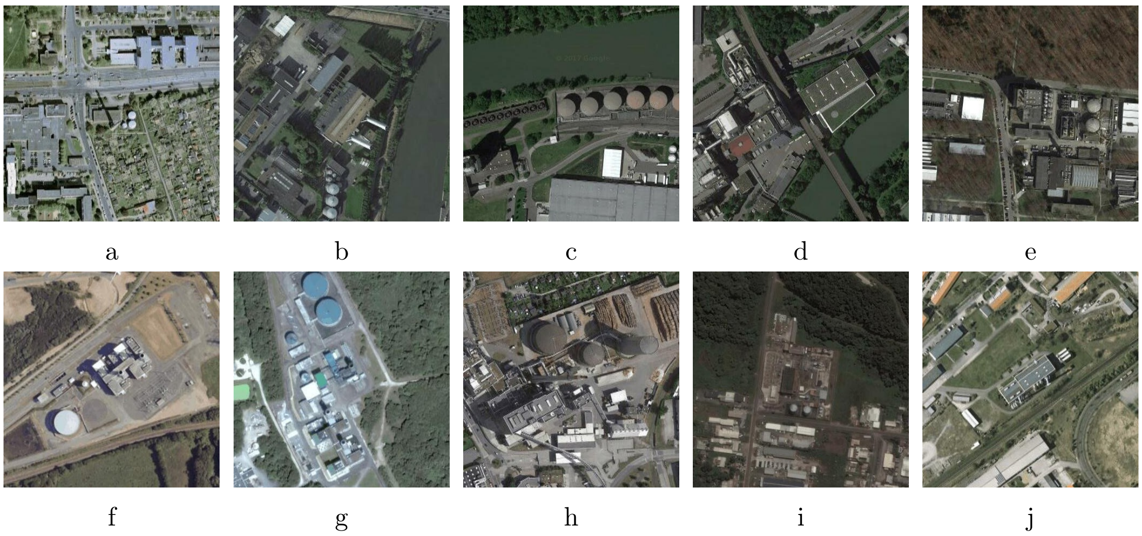


Figure A.31: Examples for class *power_ oil*

A.32 Examples for class *power_ solar*

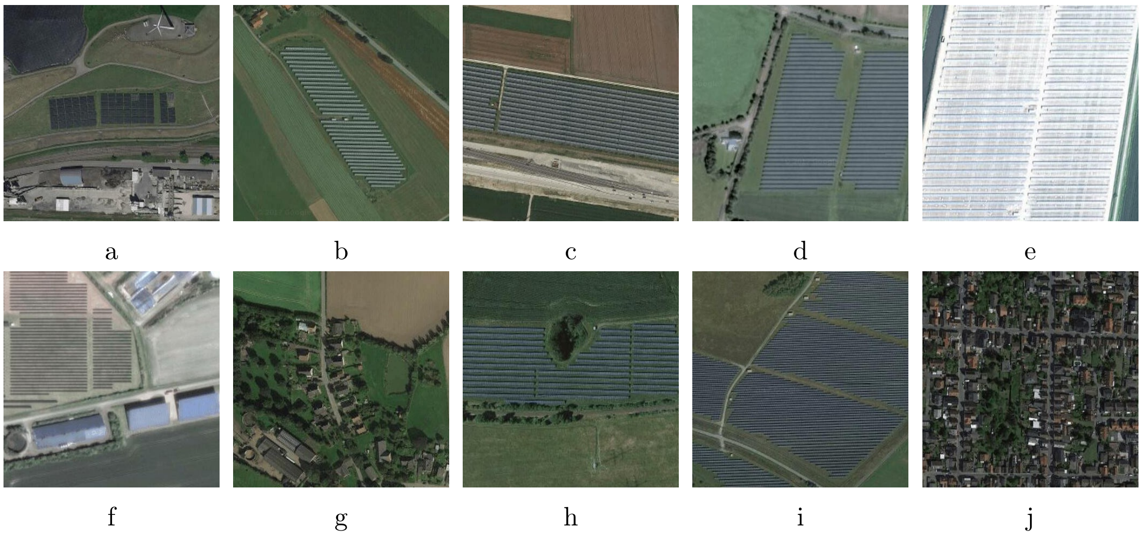


Figure A.32: Examples for class *power_ solar*

A.33 Examples for class *power_ wind*

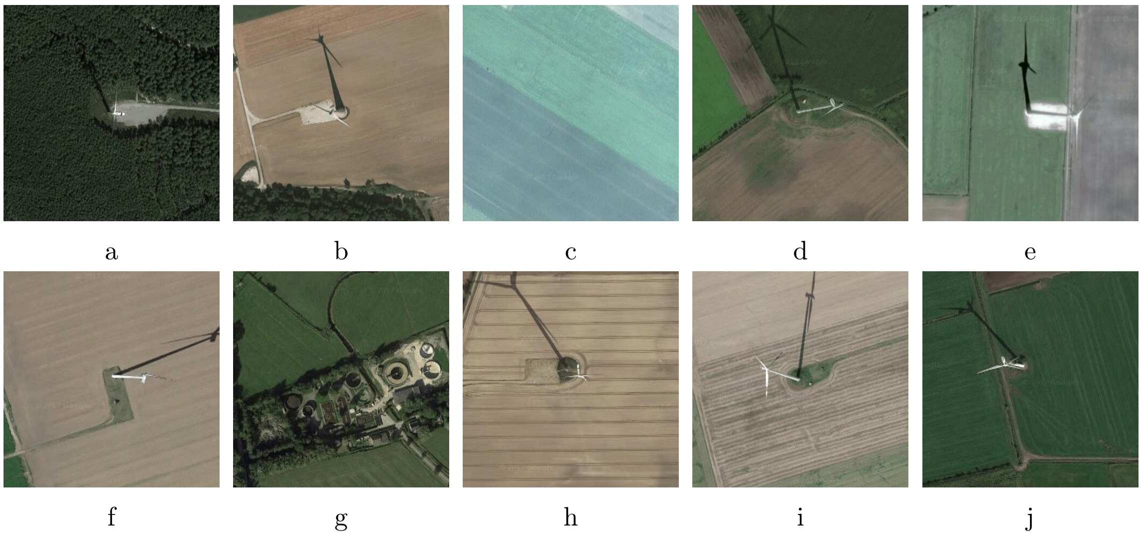


Figure A.33: Examples for class *power_ wind*

A.34 Examples for class *quarry*

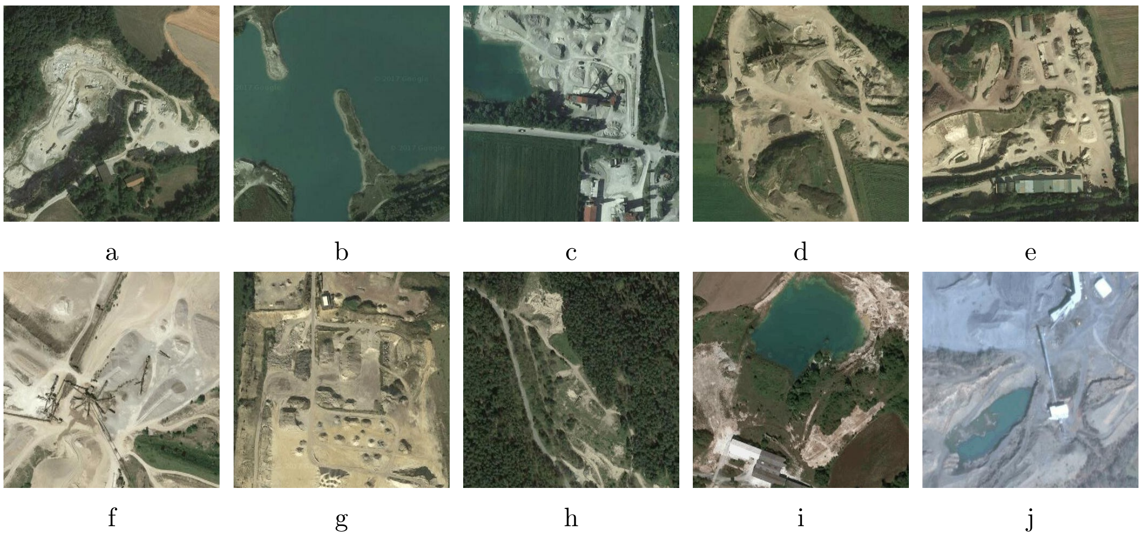


Figure A.34: Examples for class *quarry*

A.35 Examples for class *raceway*

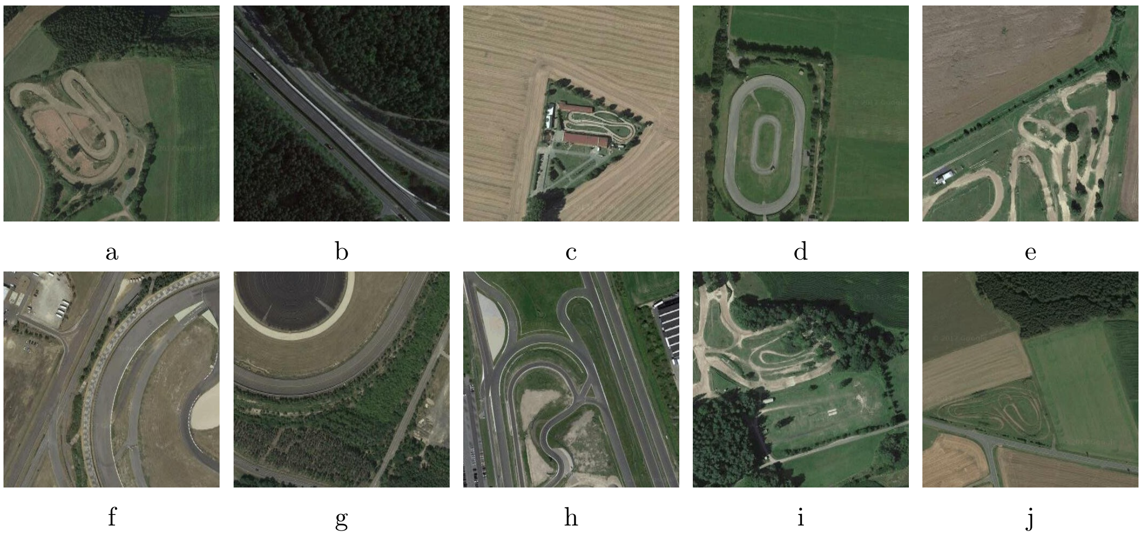


Figure A.35: Examples for class *raceway*

A.36 Examples for class *rail*

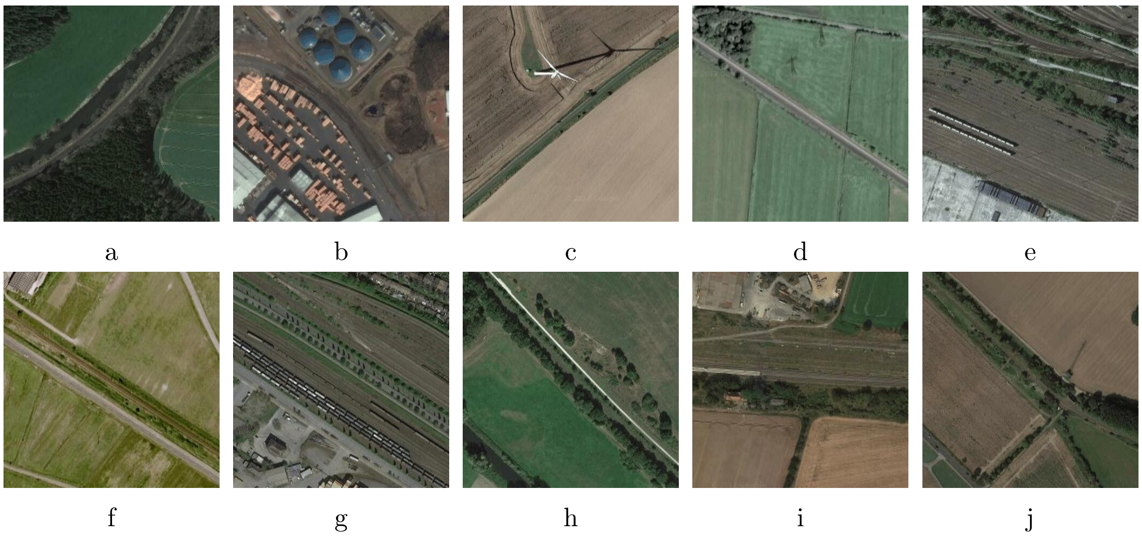


Figure A.36: Examples for class *rail*

A.37 Examples for class *river*

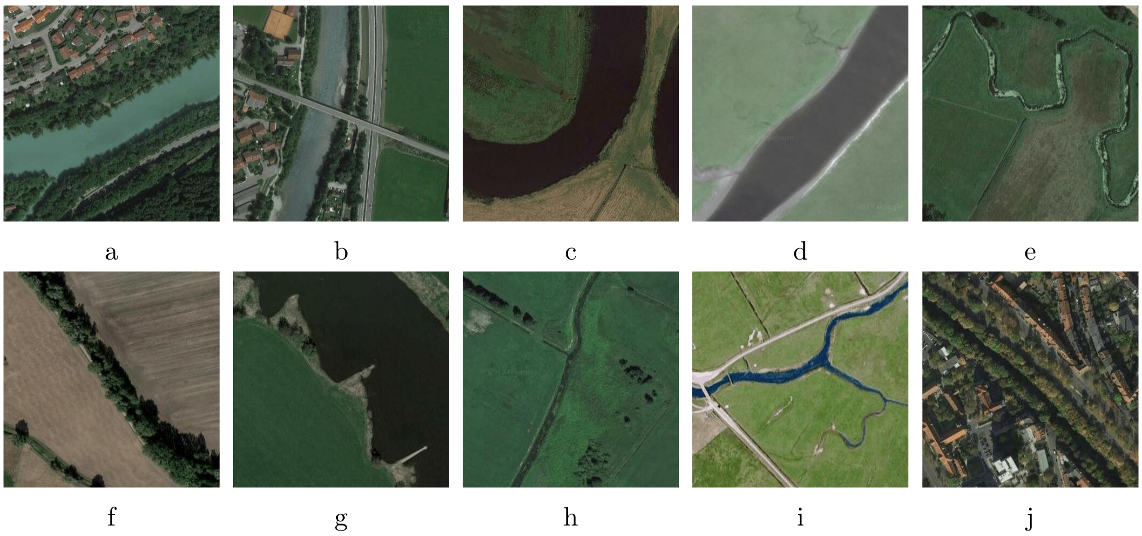


Figure A.37: Examples for class *river*

A.38 Examples for class *road*

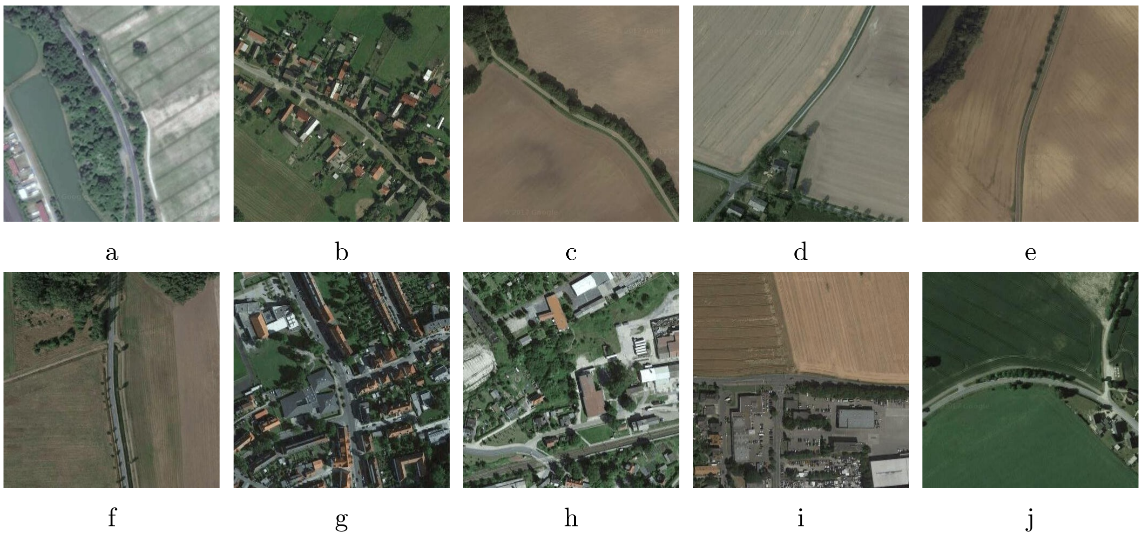


Figure A.38: Examples for class *road*

A.39 Examples for class *runway*

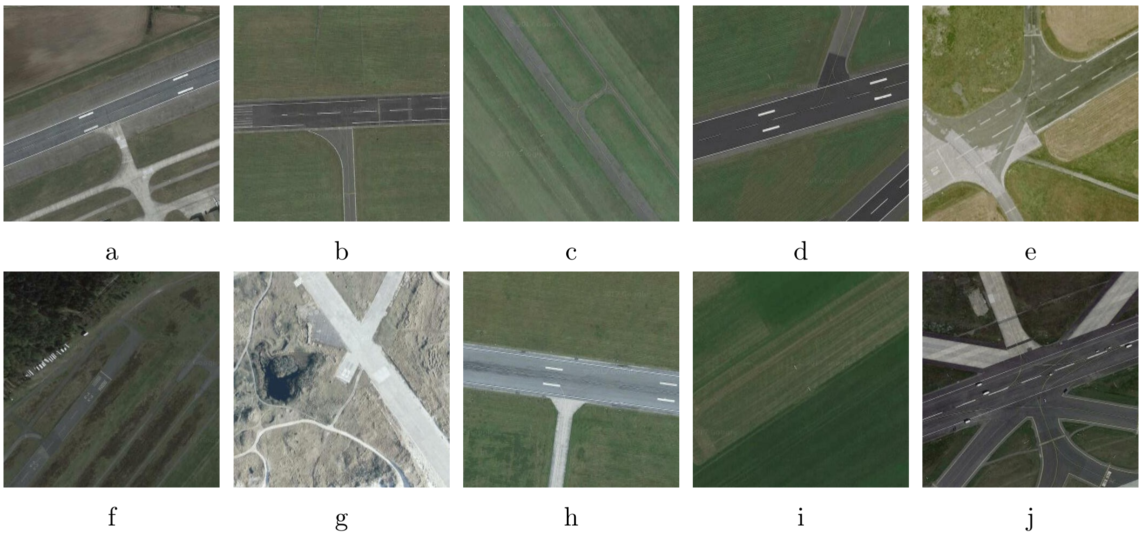


Figure A.39: Examples for class *runway*

A.40 Examples for class *snow_fence*

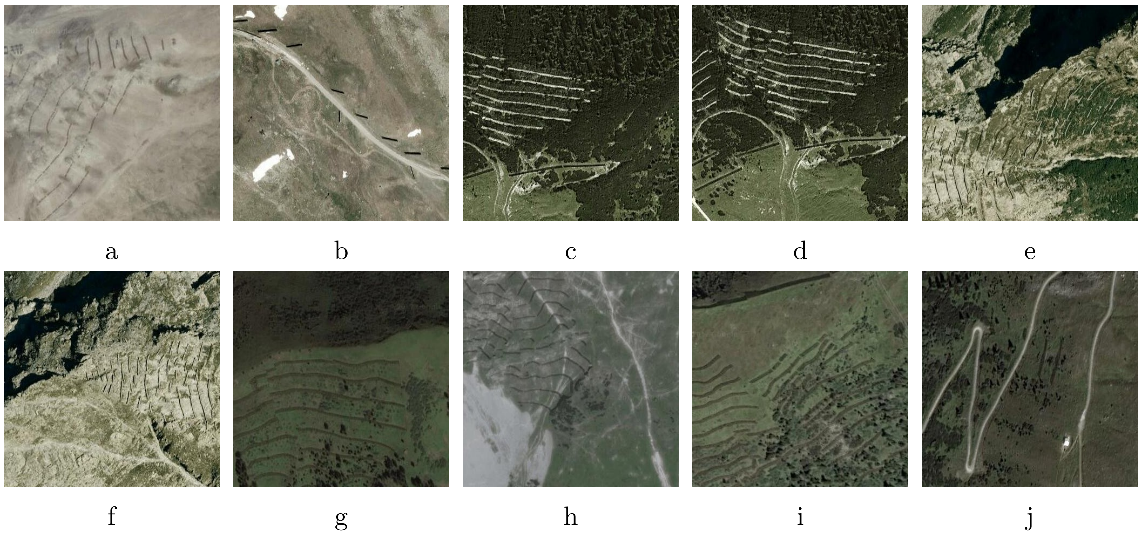


Figure A.40: Examples for class *snow_fence*

A.41 Examples for class *stadium*

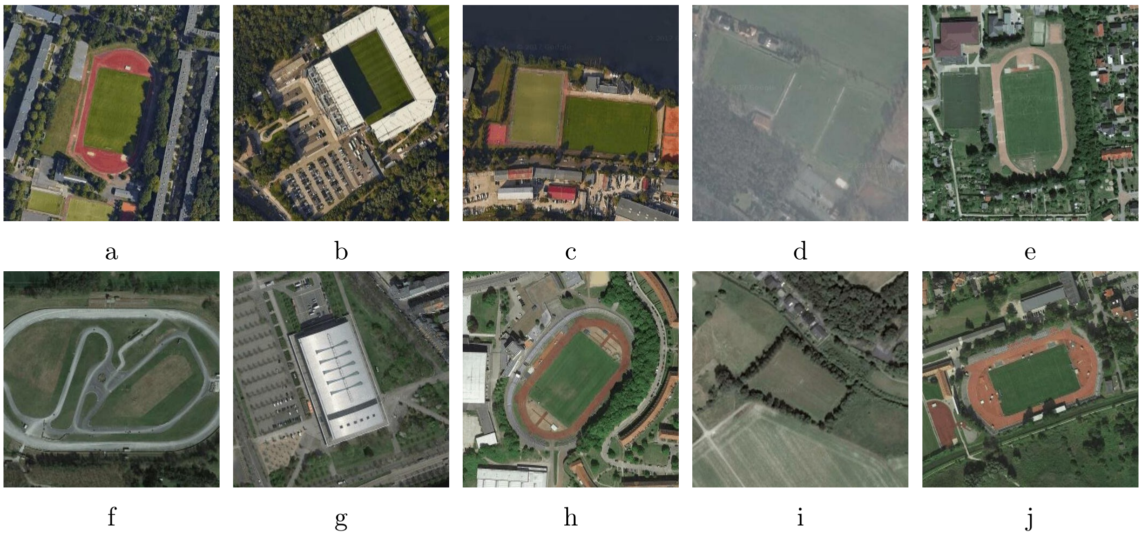


Figure A.41: Examples for class *stadium*

A.42 Examples for class *taxiway*

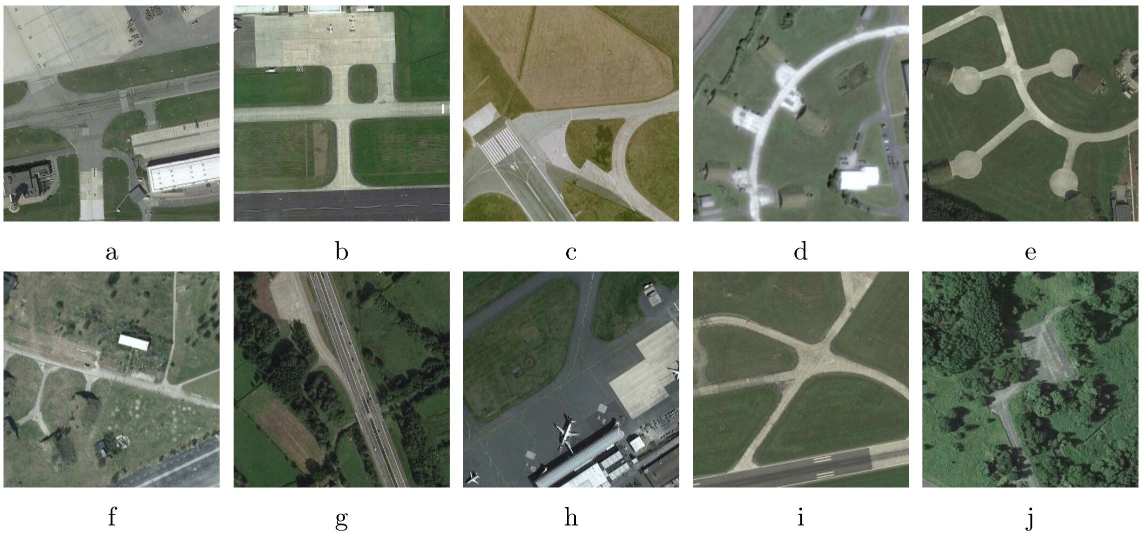


Figure A.42: Examples for class *taxiway*

A.43 Examples for class *train_ station*

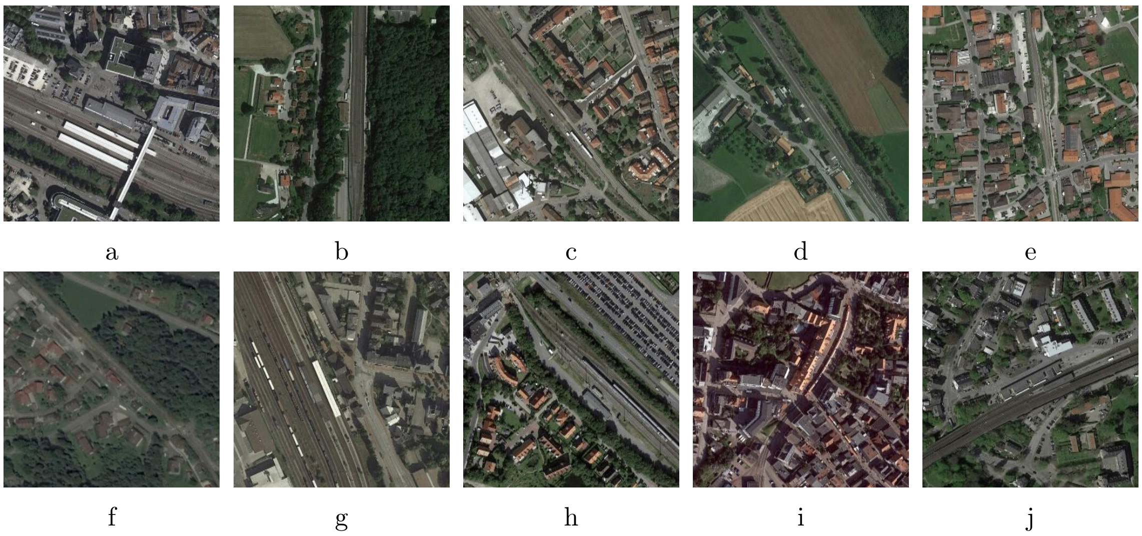


Figure A.43: Examples for class *train_ station*

A.44 Examples for class *vineyard*

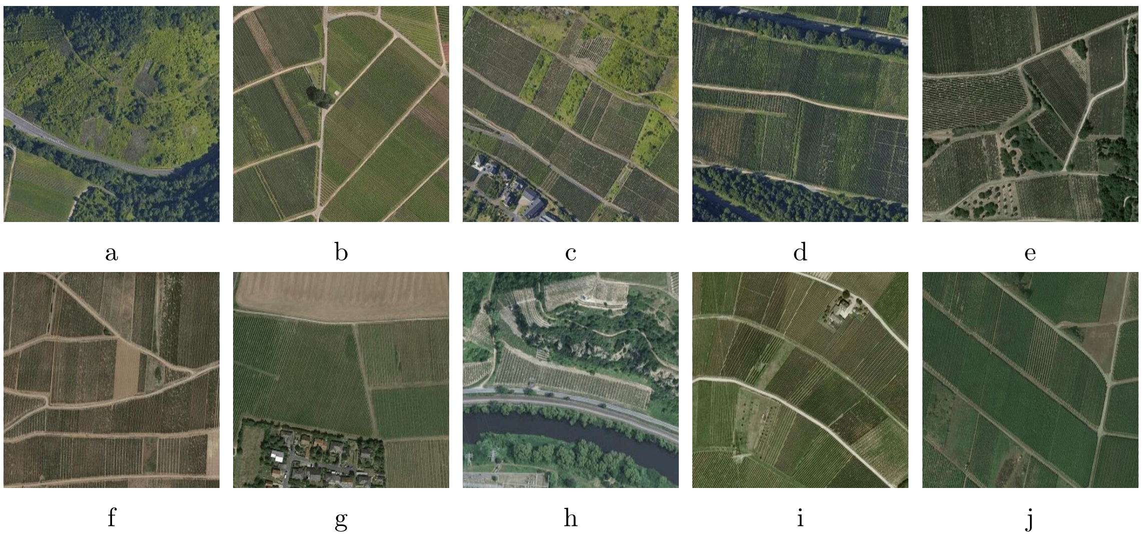


Figure A.44: Examples for class *vineyard*

A.45 Blured images

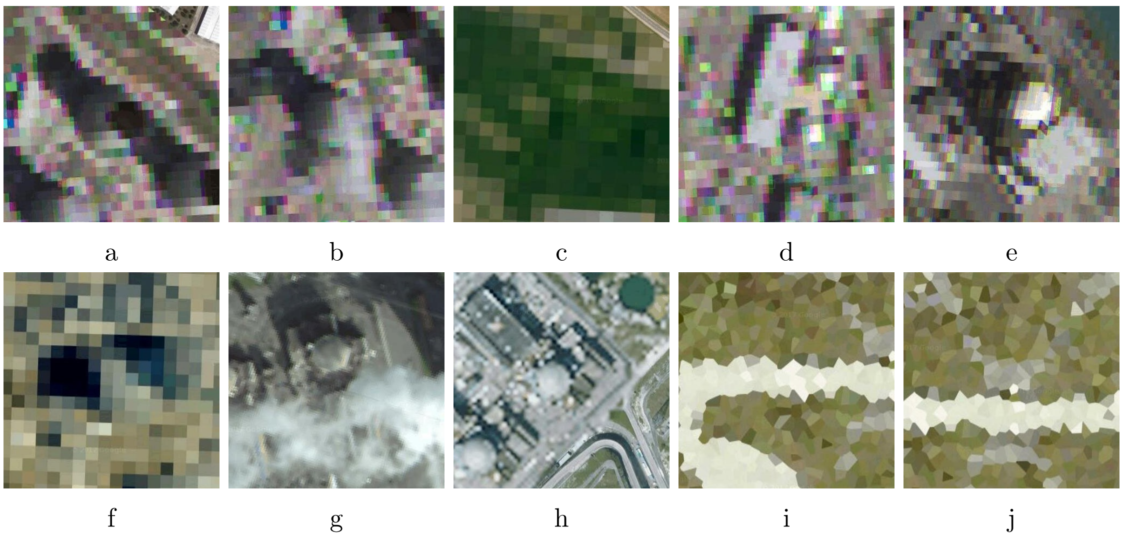
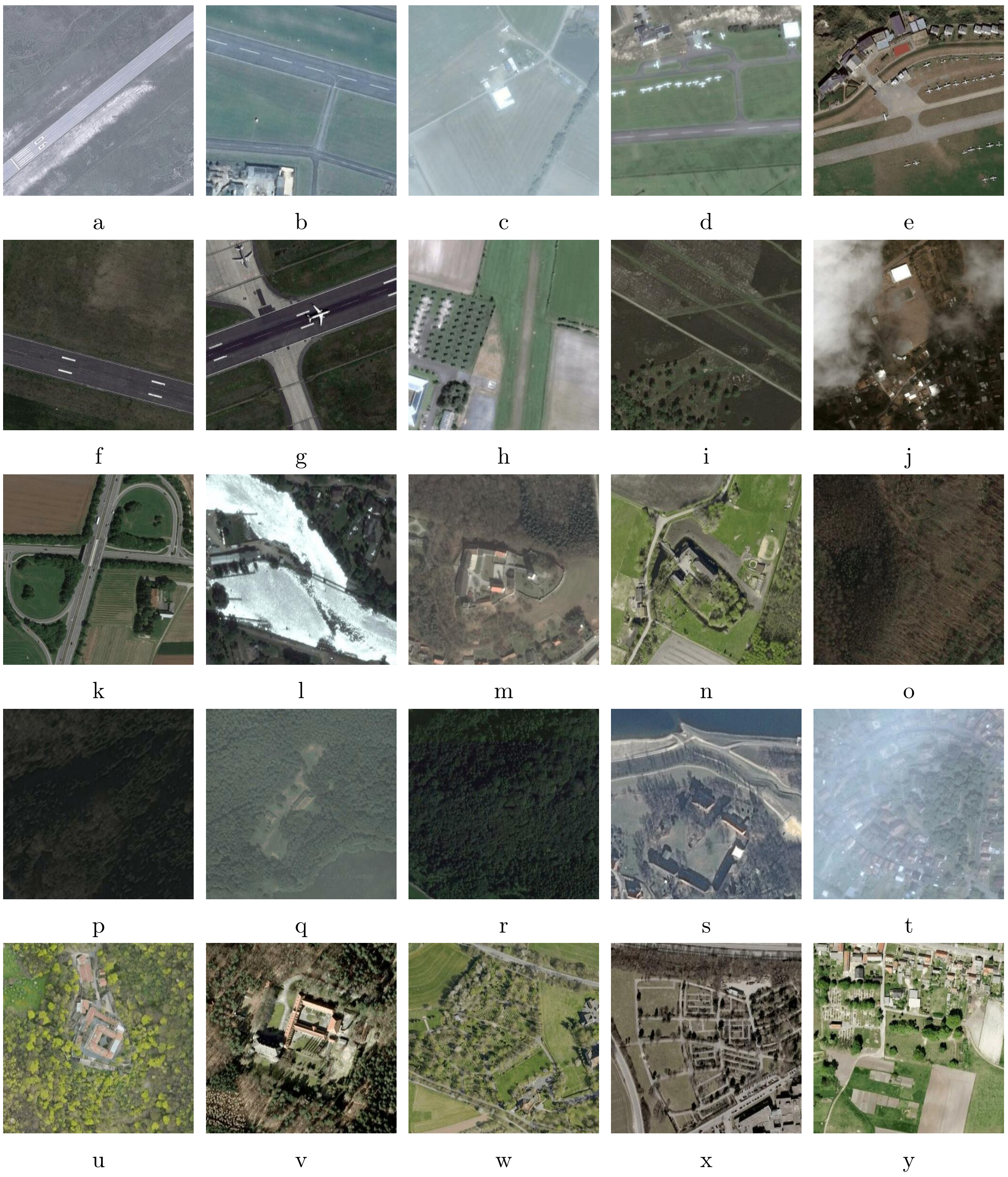


Figure A.45: The dataset contains several images that were blurred by the provider for security reasons. The number of blurred images in the dataset is insignificant and it concerns only several classes — *power_nuclear*, *airport*, *taxiway*, and *runway*.

A.46 Brightness and contrast variability of the dataset



z

aa

ab

ac

ad

ae

af

ag

ah

ai

aj

ak

al

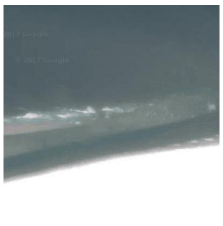
am

an

ao

ap

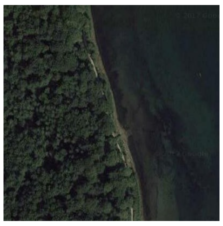
aq


ar

as

at

au


av

aw

ax

ay

az

ba

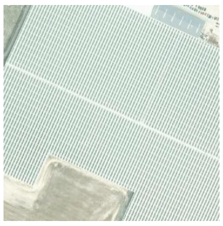
bb

bc

bd

be

bf

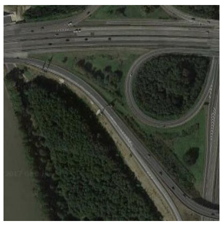
bg

bh

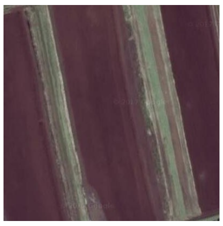
bi

bj

bk

bl

bm

bn

bo

bp

bq

br

bs

bt

bu

bv

bw

Figure A.46: Several handpicked examples to show the great contrast and brightness variaton in the proposed dataset.

Appendix B

Transfer learning

B.1 Performance comparison

			Top layers					
			GAP sig	**GMP sig**	**GMP 256d sig**	**GAP 256d sig**	**GMP 256d 0.5drop sig**	**GAP 256d 0.5drop sig**
		Fig.	B.1a	B.2a	B.3a	B.4a	B.5a	B.6a
Architecture	Fig. B.7a	**ResNet50**	88.23	88.23	92.10	**92.19**	88.98	89.04
	Fig. B.8a	**VGG 16**	87.25	84.76	87.92	89.95	86.58	88.07
	Fig. B.9a	**VGG 19**	87.23	84.82	87.96	89.96	86.68	88.05
	Fig. B.10a	**Xception**	88.07	86.65	88.12	89.81	86.11	88.19
	Fig. B.11a	**Inception v3**	88.06	87.51	89.52	91.25	87.76	88.72

Table B.1: The **training** accuracy of the labeling task on *LAB20* dataset. The accuracy is reported for the last epoch. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**. The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP sig	**GMP sig**	**GMP 256d sig**	**GAP 256d sig**	**GMP 256d 0.5drop sig**	**GAP 256d 0.5drop sig**
		Fig.	B.1a	B.2a	B.3a	B.4a	B.5a	B.6a
Architecture	Fig. B.7a	**ResNet50**	87.73	87.73	86.16	86.09	**88.32**	88.23
	Fig. B.8a	**VGG 16**	87.39	85.02	87.22	86.74	87.06	87.88
	Fig. B.9a	**VGG 19**	87.32	84.73	87.27	86.54	87.18	87.94
	Fig. B.10a	**Xception**	87.62	86.15	87.47	87.16	86.60	88.09
	Fig. B.11a	**Inception v3**	87.72	87.15	86.74	85.82	87.78	87.98

Table B.2: The **validation** accuracy of the labeling task on *LAB20* dataset. The reported accuracy is the average over last 10 epochs. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**.The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP sig	**GMP sig**	**GMP 256d sig**	**GAP 256d sig**	**GMP 256d 0.5drop sig**	**GAP 256d 0.5drop sig**
		Fig.	B.1b	B.2b	B.3b	B.4b	B.5b	B.6b
Architecture	Fig. B.7b	**ResNet50**	93.05	93.05	94.03	**94.04**	93.16	93.18
	Fig. B.8b	**VGG 16**	92.64	91.65	92.74	93.43	92.38	92.84
	Fig. B.9b	**VGG 19**	92.63	91.69	92.72	93.40	92.34	92.83
	Fig. B.10b	**Xception**	92.99	92.16	92.97	93.50	92.43	92.91
	Fig. B.11b	**Inception v3**	93.00	92.75	93.37	93.73	92.74	93.01

Table B.3: The **training** accuracy of the labeling task on *LAB44* dataset. The accuracy is reported for the last epoch. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**. The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP sig	**GMP sig**	**GMP 256d sig**	**GAP 256d sig**	**GMP 256d 0.5drop sig**	**GAP 256d 0.5drop sig**
		Fig.	B.1b	B.2b	B.3b	B.4b	B.5b	B.6b
Architecture	Fig. B.7b	**ResNet50**	92.99	93.02	92.70	92.69	**93.15**	93.13
	Fig. B.8b	**VGG 16**	92.60	92.01	92.75	92.77	92.57	92.99
	Fig. B.9b	**VGG 19**	92.70	91.57	92.72	92.79	92.52	92.98
	Fig. B.10b	**Xception**	92.94	91.85	92.82	92.96	92.61	93.03
	Fig. B.11b	**Inception v3**	92.92	92.73	92.70	92.62	92.88	93.00

Table B.4: The **validation** accuracy of the labeling task on *LAB44* dataset. The reported accuracy is the average over last 10 epochs. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**.The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP sig	**GMP sig**	**GMP 256d sig**	**GAP 256d sig**	**GMP 256d 0.5drop sig**	**GAP 256d 0.5drop sig**
		Fig.	B.1c	B.2c	B.3c	B.4c	B.5c	B.6c
Architecture	Fig. B.7c	**ResNet50**	95.00	95.00	95.80	**95.85**	95.18	95.18
	Fig. B.8c	**VGG 16**	94.76	93.95	94.87	95.39	94.62	94.94
	Fig. B.9c	**VGG 19**	94.75	93.98	94.85	95.39	94.61	94.93
	Fig. B.10c	**Xception**	94.97	94.37	95.01	95.47	94.69	95.02
	Fig. B.11c	**Inception v3**	94.97	94.80	95.36	95.61	94.89	95.09

Table B.5: The **training** accuracy of the labeling task on *LAB20S44* dataset. The accuracy is reported for the last epoch. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**. The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP sig	**GMP sig**	**GMP 256d sig**	**GAP 256d sig**	**GMP 256d 0.5drop sig**	**GAP 256d 0.5drop sig**
		Fig.	B.1c	B.2c	B.3c	B.4c	B.5c	B.6c
Architecture	Fig. B.7c	**ResNet50**	95.01	95.02	94.72	94.72	**95.09**	**95.09**
	Fig. B.8c	**VGG 16**	94.84	93.85	94.86	94.77	94.77	95.01
	Fig. B.9c	**VGG 19**	94.77	93.36	94.84	94.76	94.74	94.97
	Fig. B.10c	**Xception**	94.71	94.58	94.89	94.86	94.87	95.06
	Fig. B.11c	**Inception v3**	94.97	94.78	94.78	94.72	94.98	95.04

Table B.6: The **validation** accuracy of the labeling task on *LAB20S44* dataset. The reported accuracy is the average over last 10 epochs. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**.The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP soft	**GMP soft**	**GMP 256d soft**	**GAP 256d soft**	**GMP 256d 0.5drop soft**	**GAP 256d 0.5drop soft**
		Fig.	B.1d	B.2d	B.3d	B.4d	B.5d	B.6d
Architecture	Fig. B.7d	**ResNet50**	65.61	65.61	96.13	96.15	67.79	67.76
	Fig. B.8d	**VGG 16**	54.50	34.43	5.18	71.83	42.17	50.41
	Fig. B.9d	**VGG 19**	54.11	32.90	56.96	87.15	41.36	60.12
	Fig. B.10d	**Xception**	64.20	38.19	5.13	85.71	5.13	60.21
	Fig. B.11d	**Inception v3**	63.68	58.42	77.23	**98.66**	50.86	75.63

Table B.7: The **training** accuracy of the classification task on *CLS2O* dataset. The accuracy is reported for the last epoch. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**. The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP soft	**GMP soft**	**GMP 256d soft**	**GAP 256d soft**	**GMP 256d 0.5drop soft**	**GAP 256d 0.5drop soft**
		Fig.	B.1d	B.2d	B.3d	B.4d	B.5d	B.6d
Architecture	Fig. B.7d	**ResNet50**	56.47	56.47	54.39	54.77	**59.75**	58.76
	Fig. B.8d	**VGG 16**	52.97	33.70	5.27	48.11	46.95	51.87
	Fig. B.9d	**VGG 19**	52.20	30.67	46.21	48.12	48.09	53.31
	Fig. B.10d	**Xception**	53.97	34.57	5.27	50.65	5.27	56.88
	Fig. B.11d	**Inception v3**	53.30	49.92	47.05	50.80	51.58	54.24

Table B.8: The **validation** accuracy of the classification task on *CLS2O* dataset. The reported accuracy is the average over last 10 epochs. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**.The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP soft	**GMP soft**	**GMP 256d soft**	**GAP 256d soft**	**GMP 256d 0.5drop soft**	**GAP 256d 0.5drop soft**
		Fig.	B.1e	B.2e	B.3e	B.4e	B.5e	B.6e
Architecture	Fig. B.7e	**ResNet50**	52.38	52.39	69.37	69.44	45.22	45.27
	Fig. B.8e	**VGG 16**	42.16	16.06	2.57	46.87	24.07	30.95
	Fig. B.9e	**VGG 19**	41.88	18.04	37.11	N/A	22.73	41.35
	Fig. B.10e	**Xception**	52.01	21.81	2.51	60.15	2.54	42.33
	Fig. B.11e	**Inception v3**	51.99	45.89	50.52	**82.91**	32.08	52.41

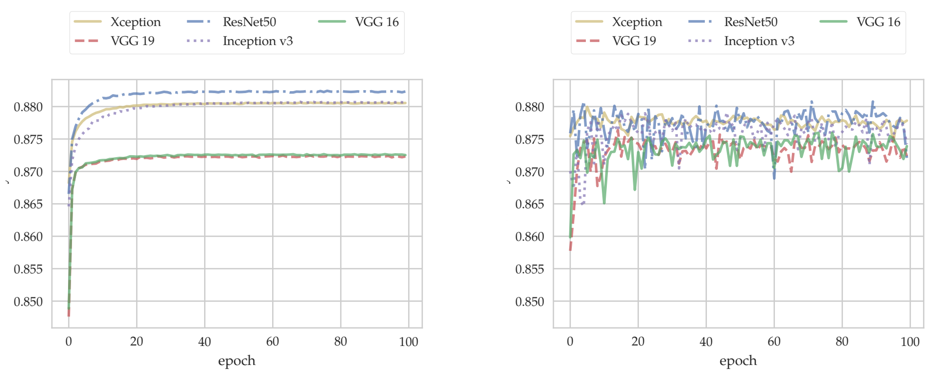
Table B.9: The **training** accuracy of the classification task on *CLS44* dataset. The accuracy is reported for the last epoch. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**. The references to figures contain graphs of whole training for the given architecture or top layer configuration.

			Top layers					
			GAP soft	**GMP soft**	**GMP 256d soft**	**GAP 256d soft**	**GMP 256d 0.5drop soft**	**GAP 256d 0.5drop soft**
		Fig.	B.1e	B.2e	B.3e	B.4e	B.5e	B.6e
Architecture	Fig. B.7e	**ResNet50**	45.20	45.14	43.81	43.29	46.21	**46.46**
	Fig. B.8e	**VGG 16**	40.34	14.75	2.57	39.69	6.10	36.41
	Fig. B.9e	**VGG 19**	40.97	18.22	35.63	N/A	31.40	42.32
	Fig. B.10e	**Xception**	43.22	19.09	2.57	42.64	2.57	44.25
	Fig. B.11e	**Inception v3**	42.04	38.24	38.91	36.76	38.26	43.48

Table B.10: The **validation** accuracy of the classification task on *CLS44* dataset. The reported accuracy is the average over last 10 epochs. The highest value for given architecture is in green, for the particular top layer in blue, and the overall highest value is in **bold**.The references to figures contain graphs of whole training for the given architecture or top layer configuration.

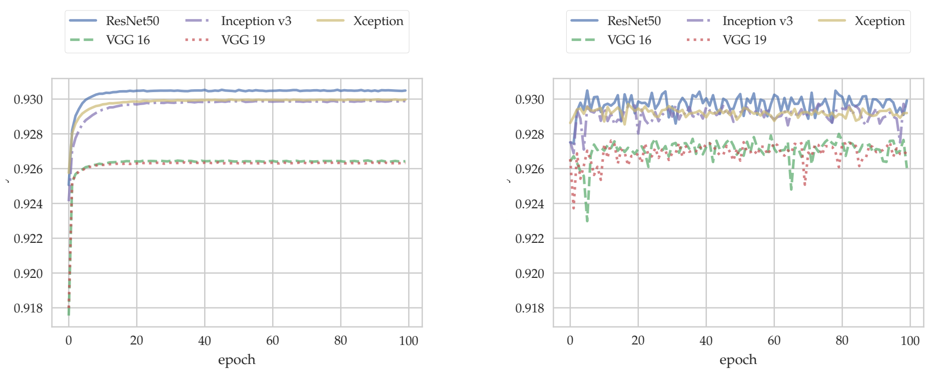
B.2 Training process of all NNs

B.2.1 Different architectures with GAP layer for transfer learning



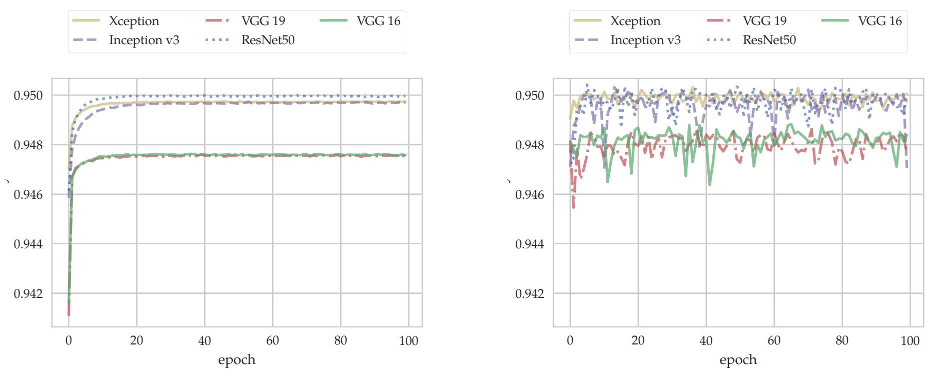
training validation

a Dataset *LAB20*



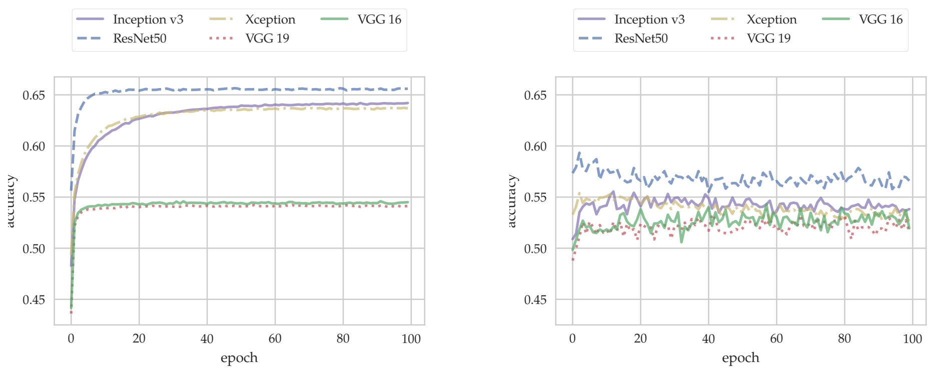
training validation

b Dataset *LAB44*



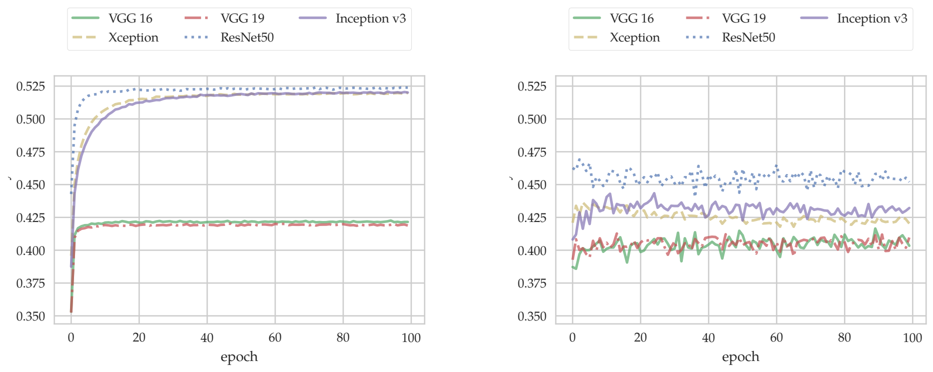
training validation

c Dataset *LAB20S44*



training validation

d Dataset *CLS20*

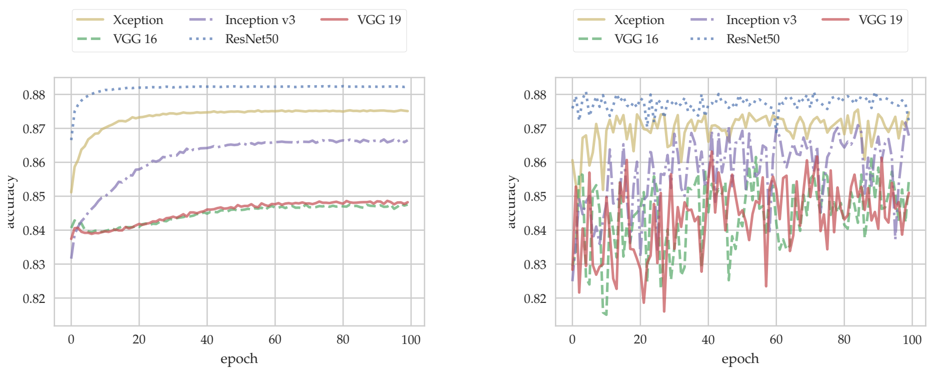


training validation

e Dataset *CLS44*

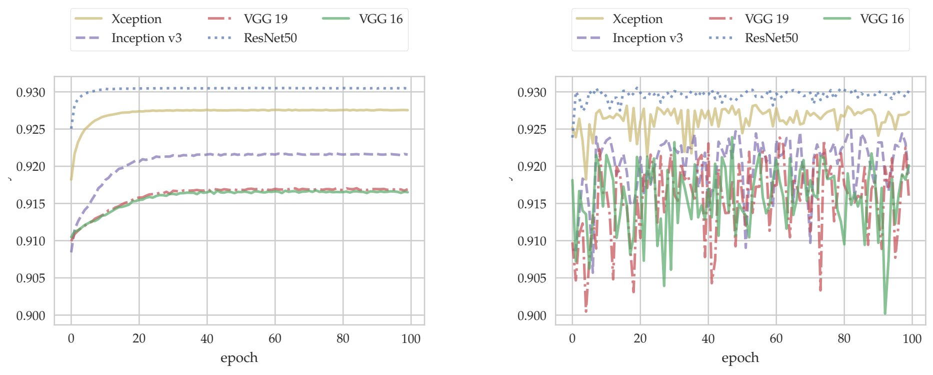
Figure B.1

B.2.2 Different architectures with GMP layers for transfer learning



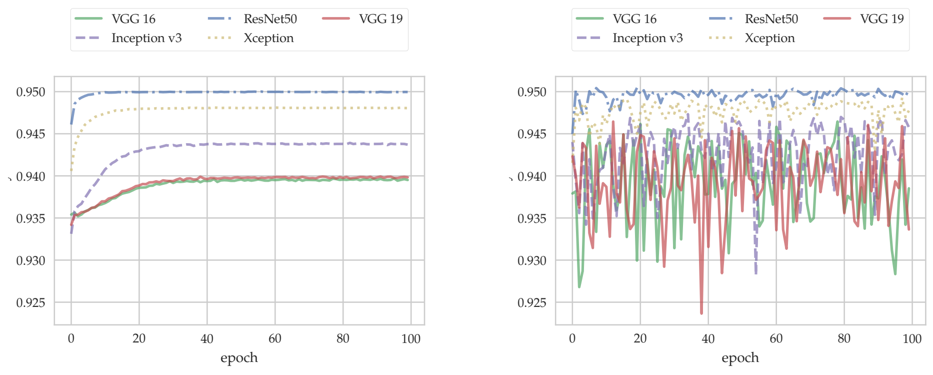
training validation

a Dataset *LAB20*



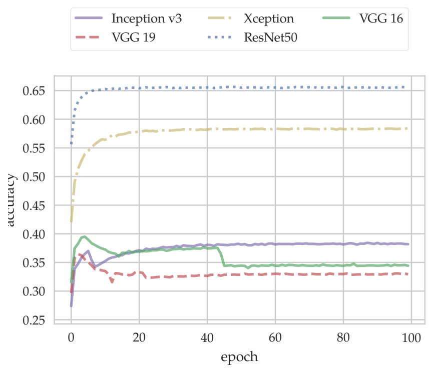
training validation

b Dataset *LAB44*

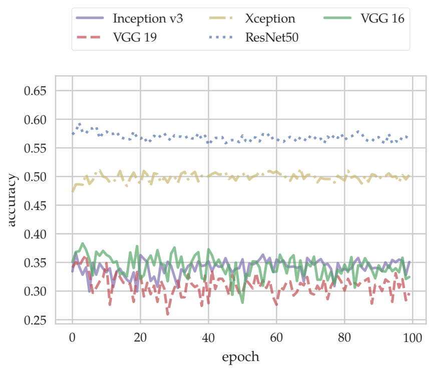


training validation

c Dataset *LAB20S44*

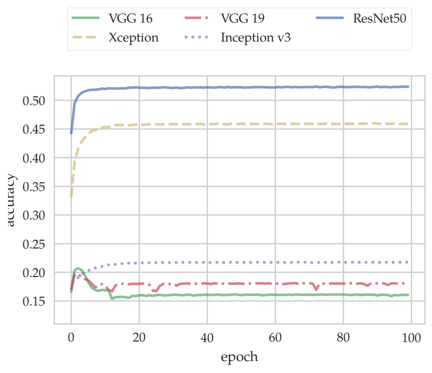


training

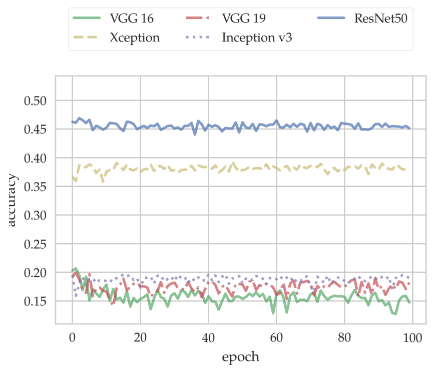


validation

d Dataset *CLS20*



training

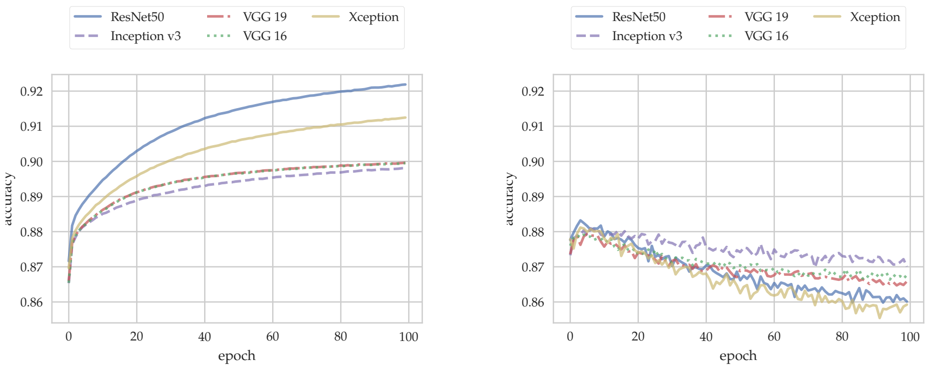


validation

e Dataset *CLS44*

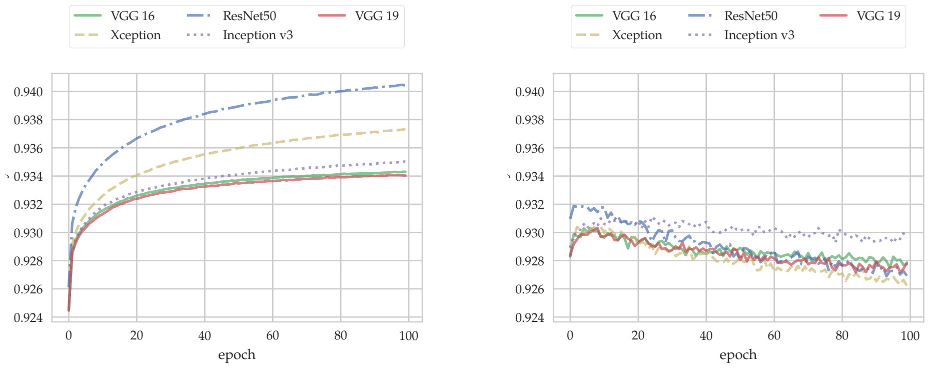
Figure B.2

B.2.3 Different architectures with GAP-256dense layers for transfer learning



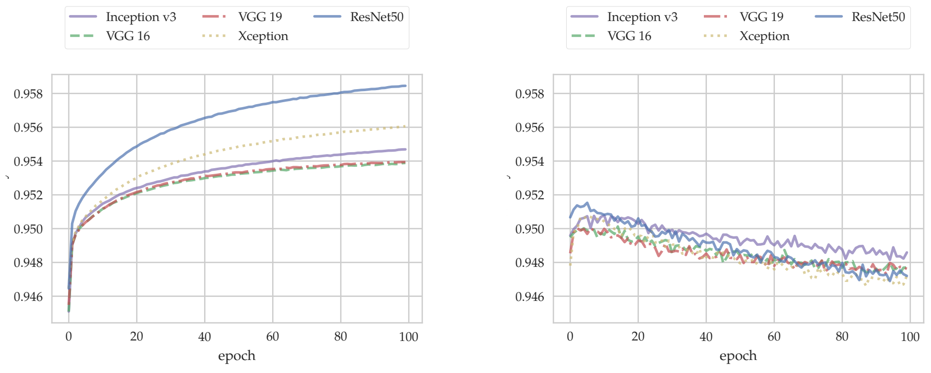
training validation

a Dataset *LAB20*



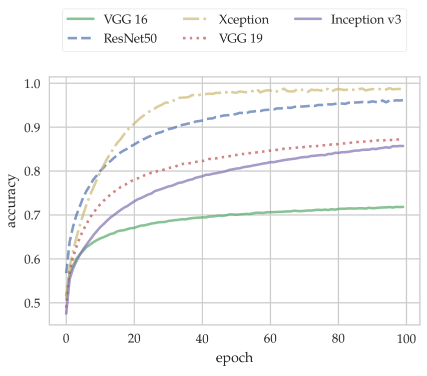
training validation

b Dataset *LAB44*

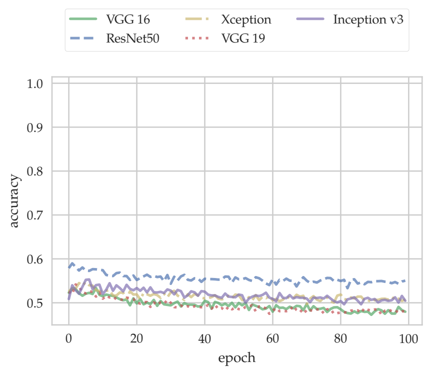


training validation

c Dataset *LAB20S44*

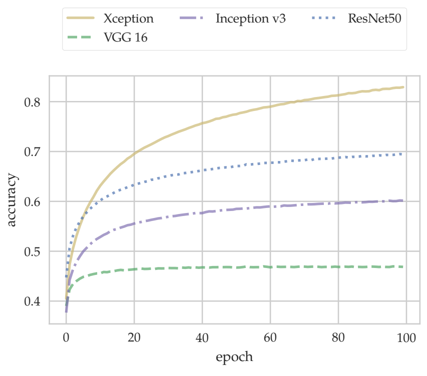


training

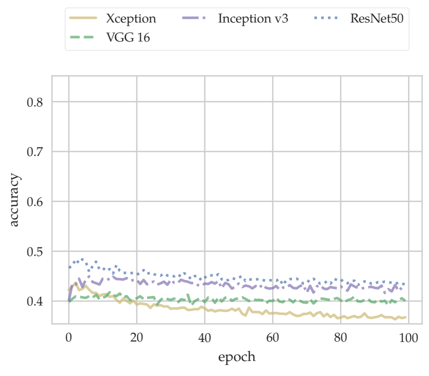


validation

d Dataset *CLS20*



training

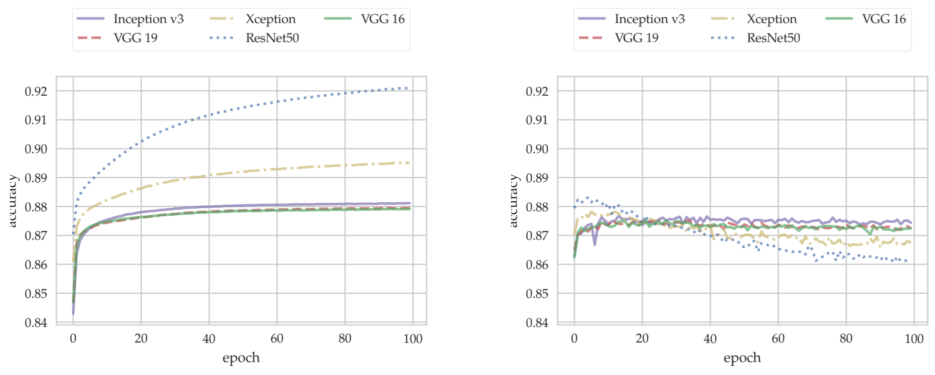


validation

e Dataset *CLS44*

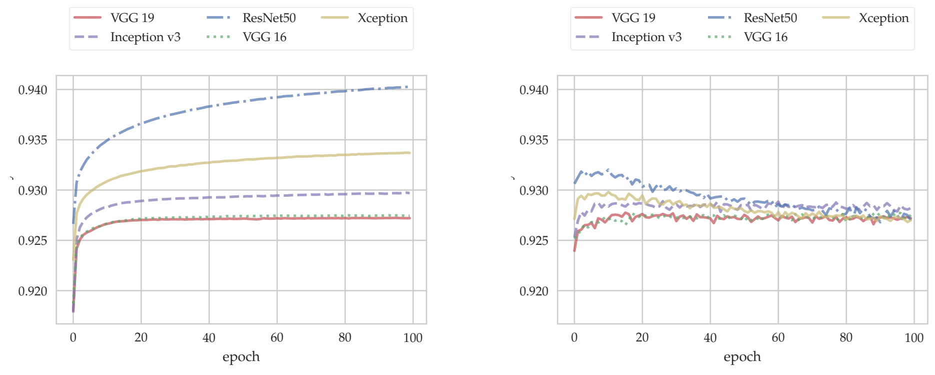
Figure B.3

B.2.4 Different architectures with GMP-256dense layers for transfer learning



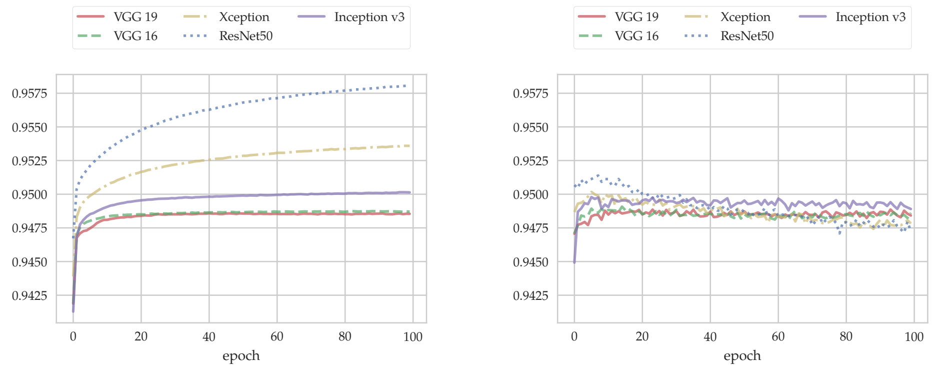
training validation

a Dataset *LAB20*



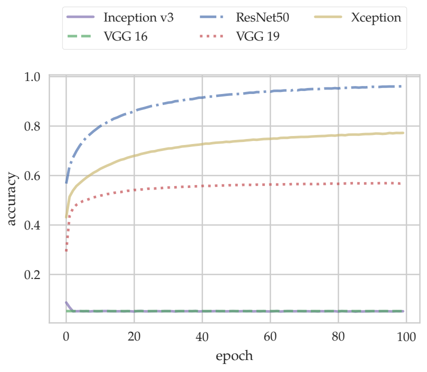
training validation

b Dataset *LAB44*

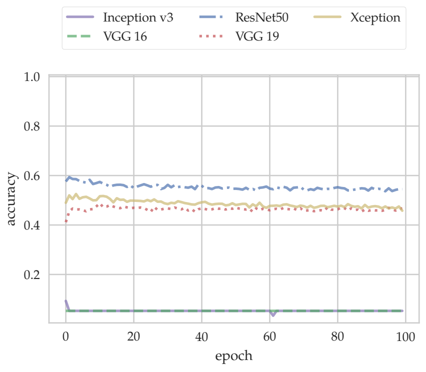


training validation

c Dataset *LAB20S44*

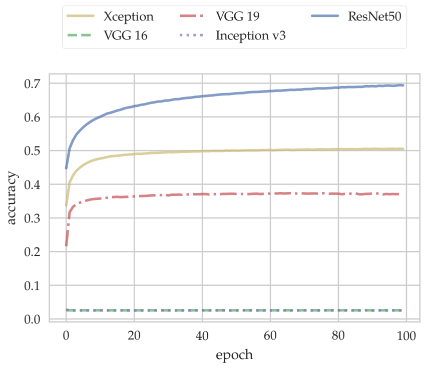


training

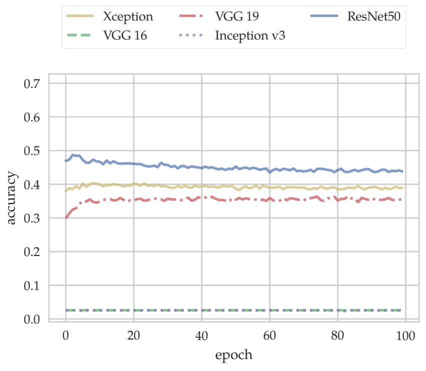


validation

d Dataset *CLS20*



training

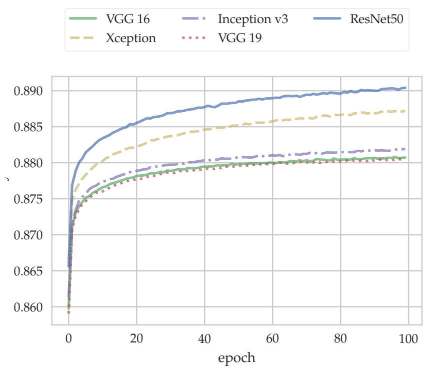


validation

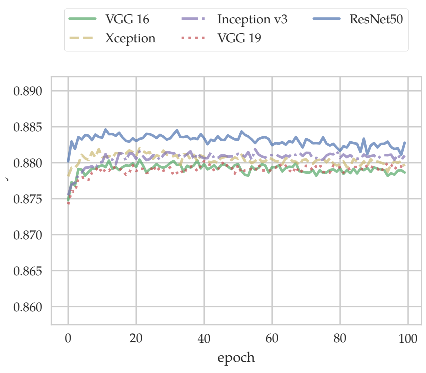
e Dataset *CLS44*

Figure B.4

B.2.5 Different architectures with GAP-256dense-0.5drop layers for transfer learning

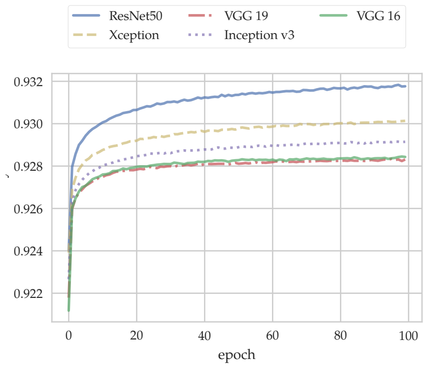


training

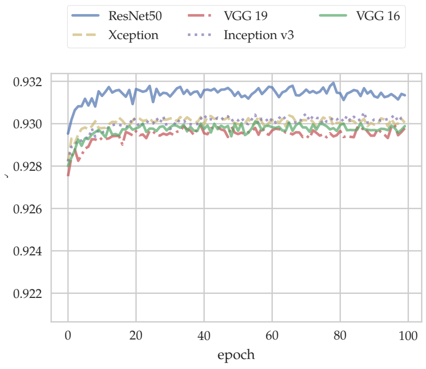


validation

a Dataset *LAB20*

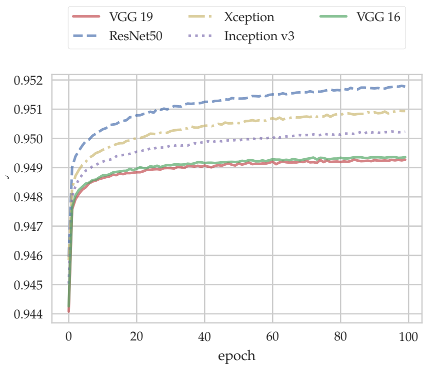


training

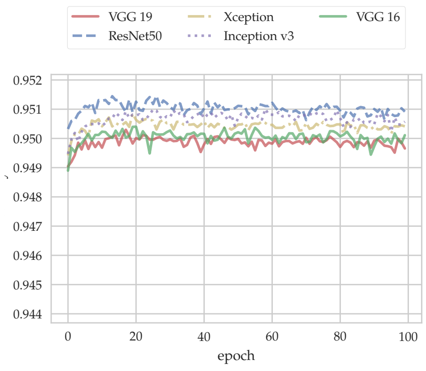


validation

b Dataset *LAB44*

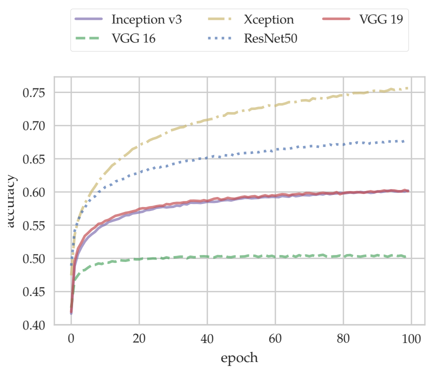


training

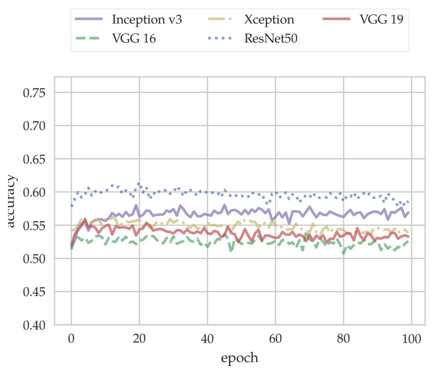


validation

c Dataset *LAB20S44*

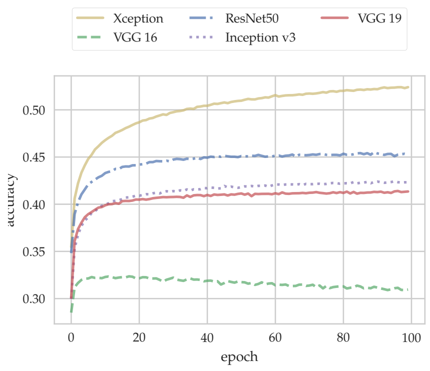


training

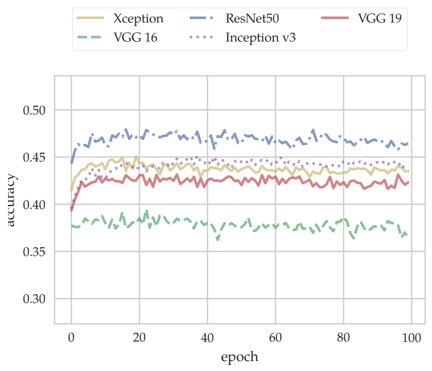


validation

d Dataset *CLS20*



training

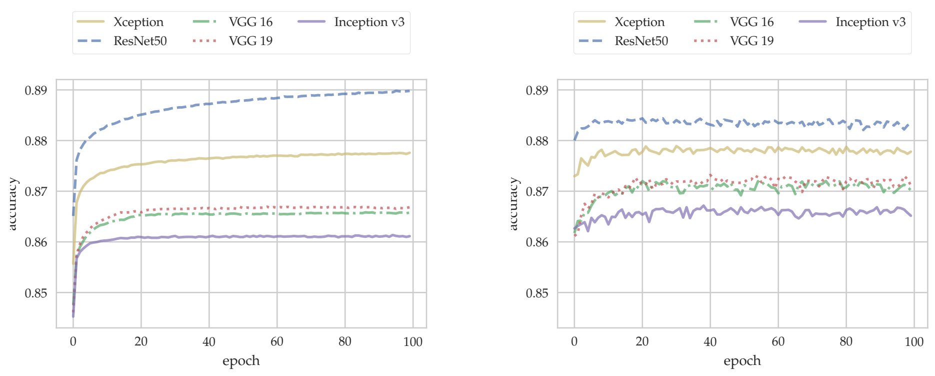


validation

e Dataset *CLS44*

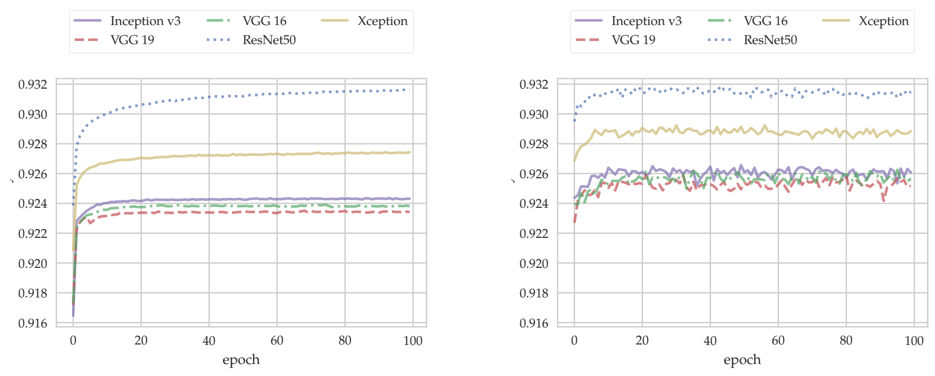
Figure B.5

B.2.6 Different architectures with GMP-256dense-0.5drop layers for transfer learning



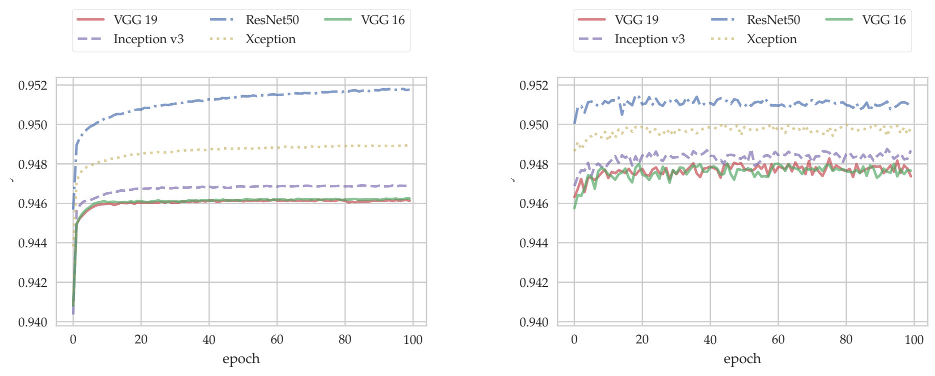
training validation

a Dataset *LAB20*



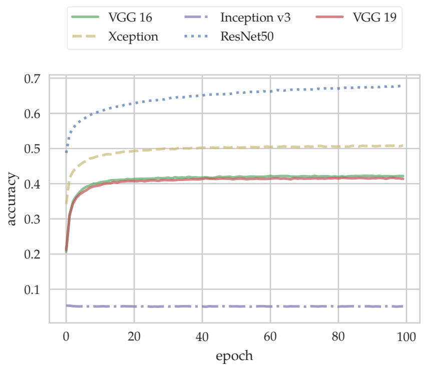
training validation

b Dataset *LAB44*

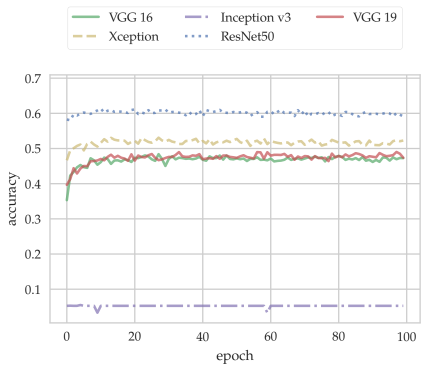


training validation

c Dataset *LAB20S44*

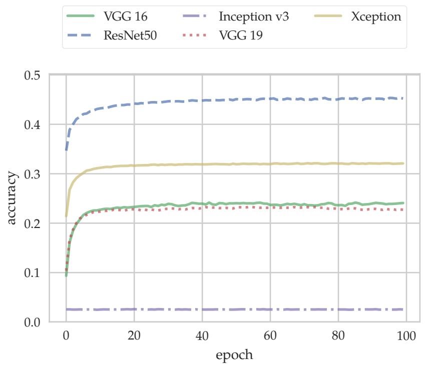


training

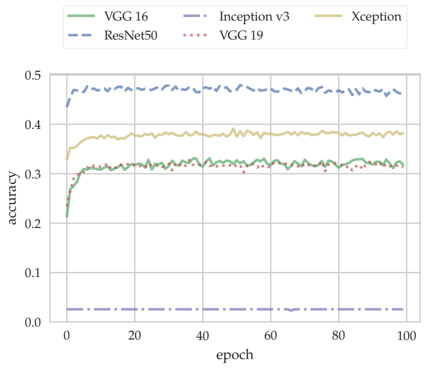


validation

d Dataset *CLS20*



training



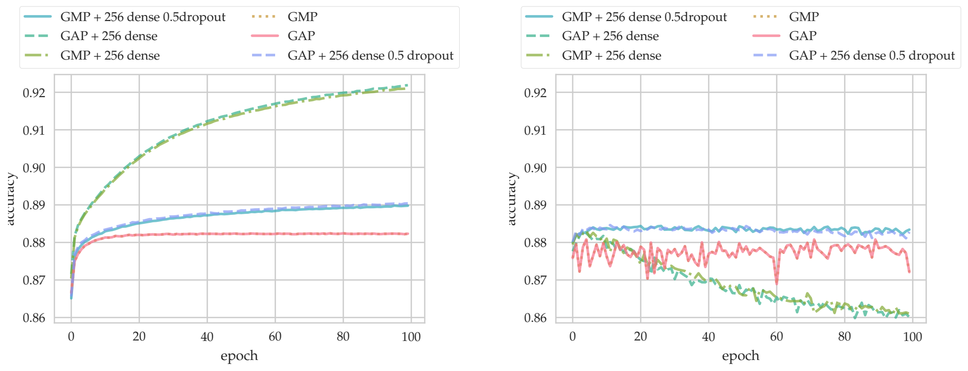
validation

e Dataset *CLS44*

Figure B.6

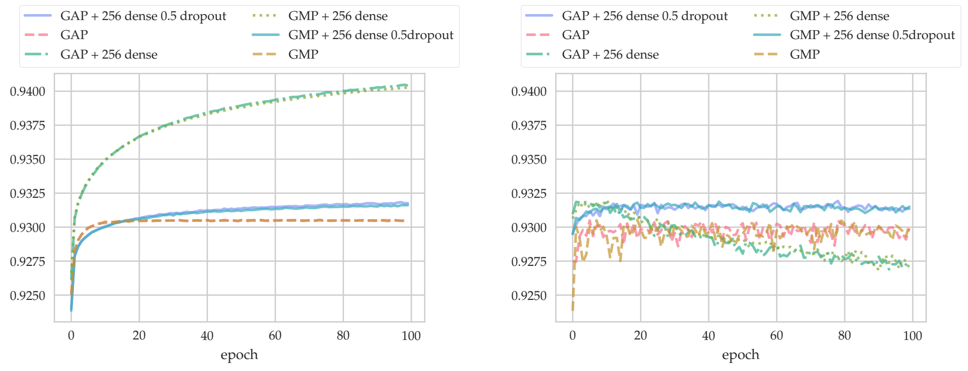
B.3 Different top layers for transfer learning by architecture

B.3.1 Different top layers of ResNet50 for transfer learning



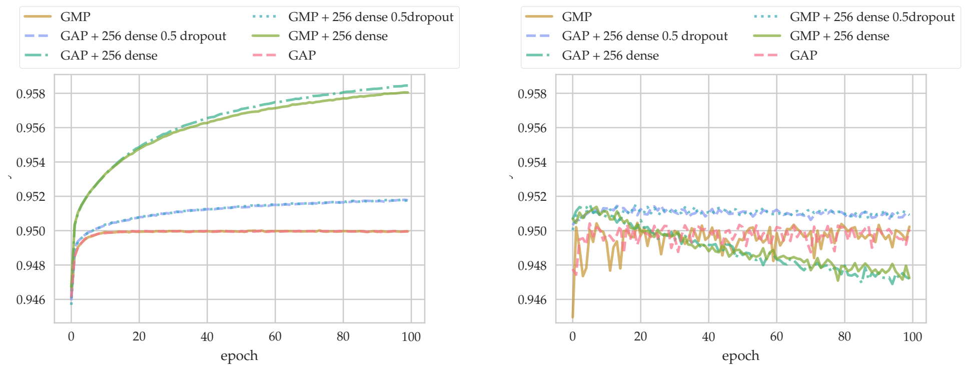
training validation

a Dataset *LAB20*



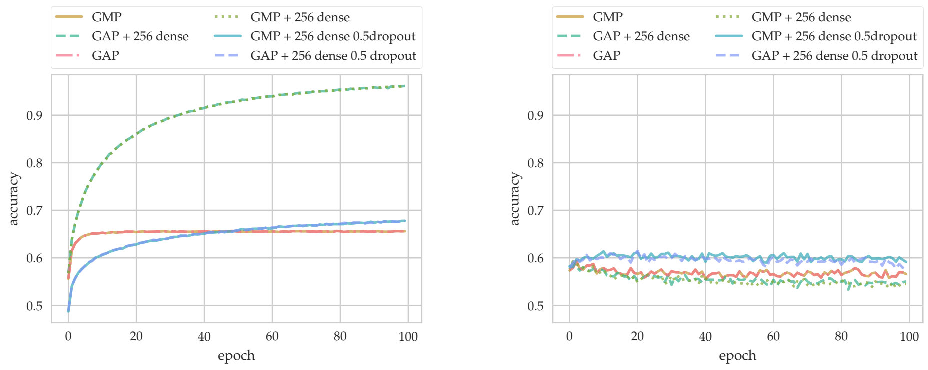
training validation

b Dataset *LAB44*



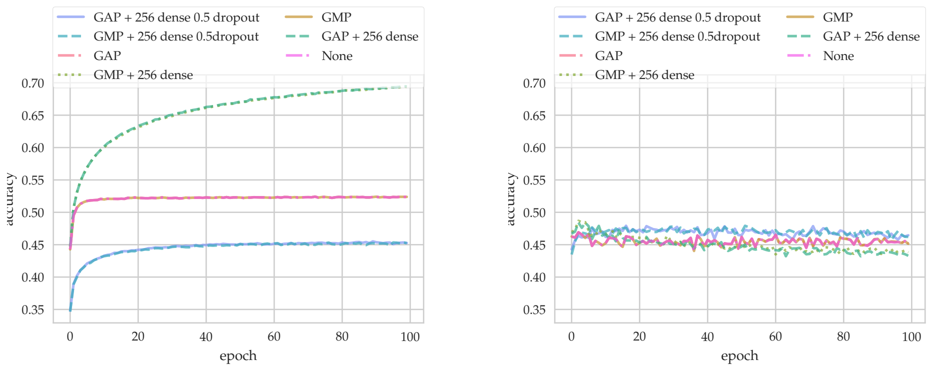
training validation

c Dataset *LAB20S44*



training validation

d Dataset *CLS20*

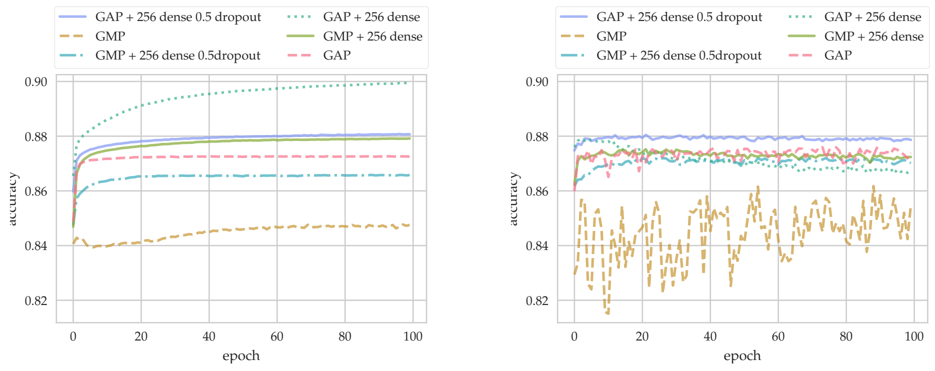


training validation

e Dataset *CLS44*

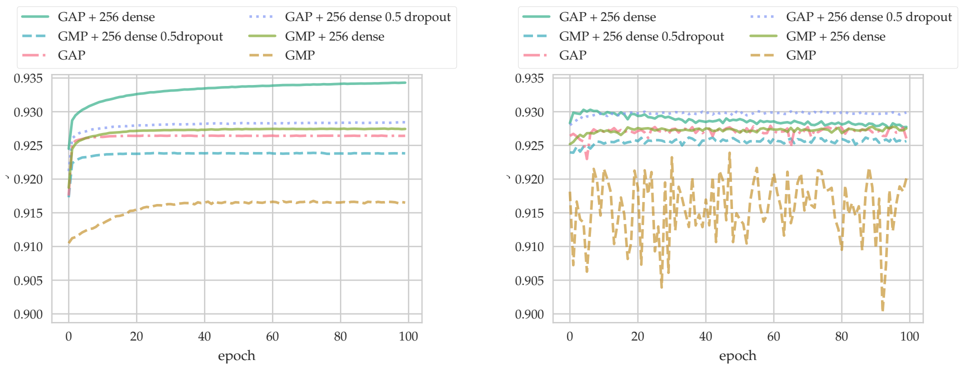
Figure B.7

B.3.2 Different top layers of VGG 16 for transfer learning



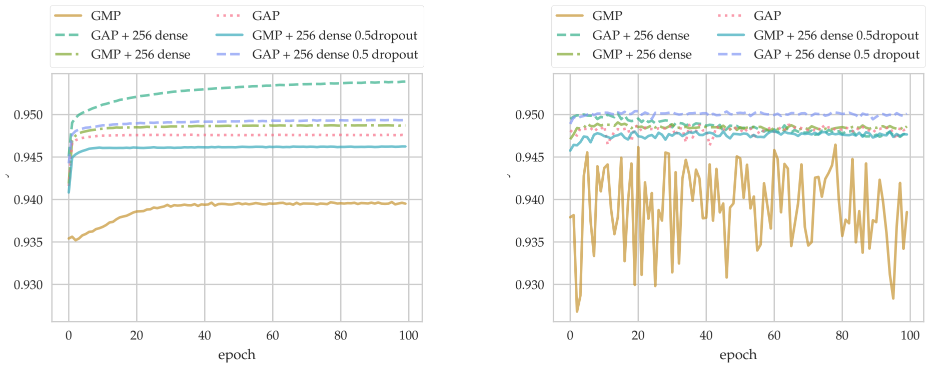
training validation

a Dataset *LAB20*



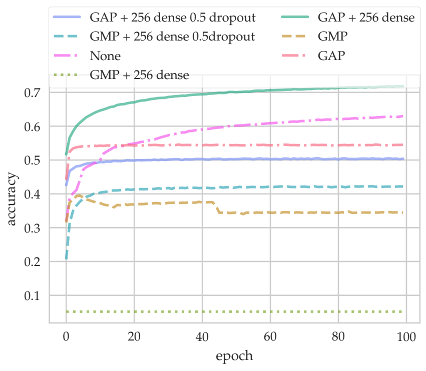
training validation

b Dataset *LAB44*

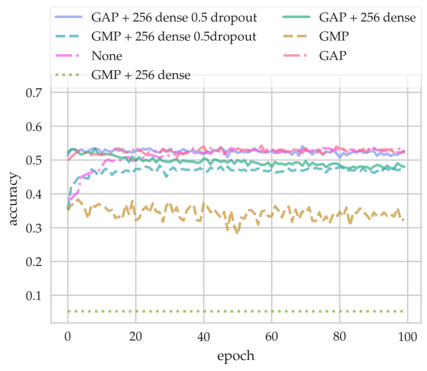


training validation

c Dataset *LAB20S44*

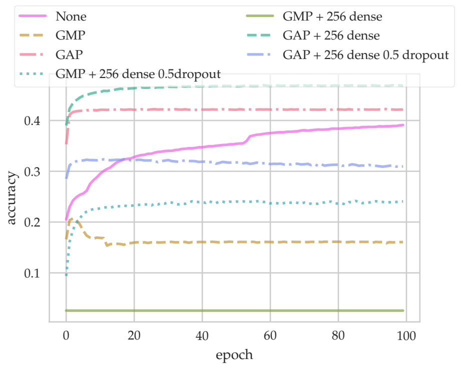


training

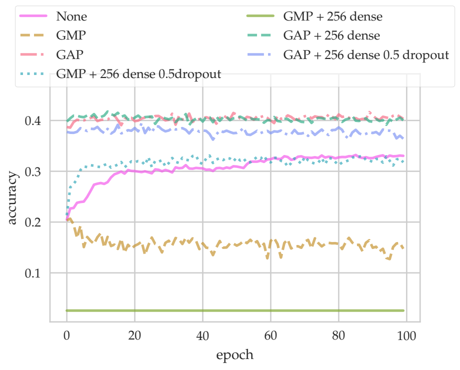


validation

d Dataset *CLS20*



training

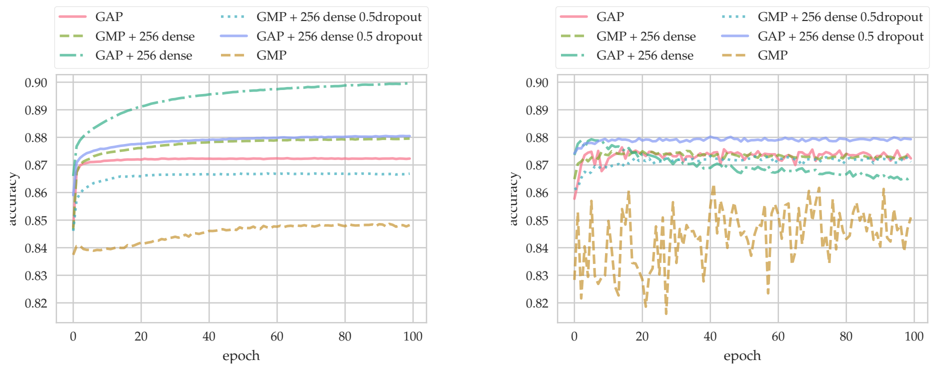


validation

e Dataset *CLS44*

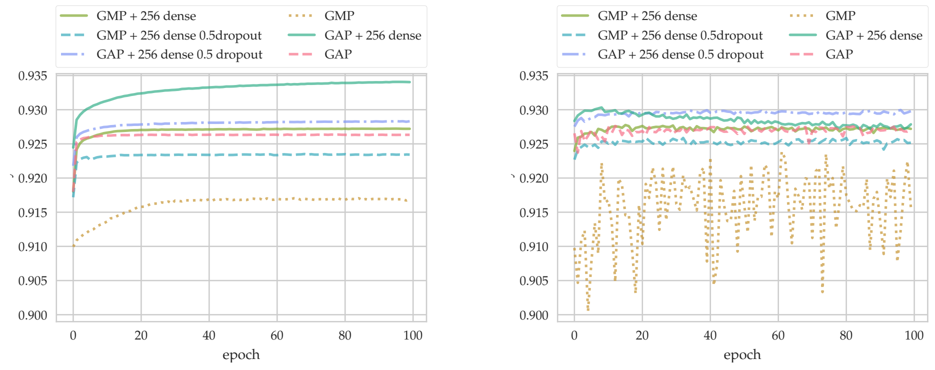
Figure B.8

B.3.3 Different top layers of VGG 19 for transfer learning



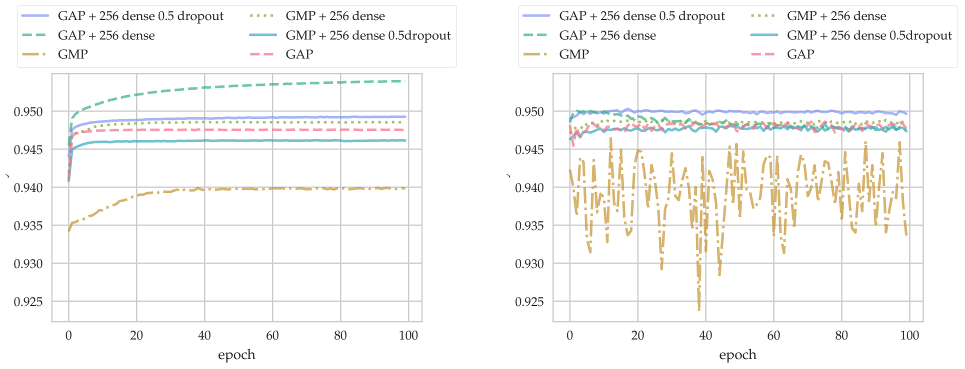
training validation

a Dataset *LAB20*



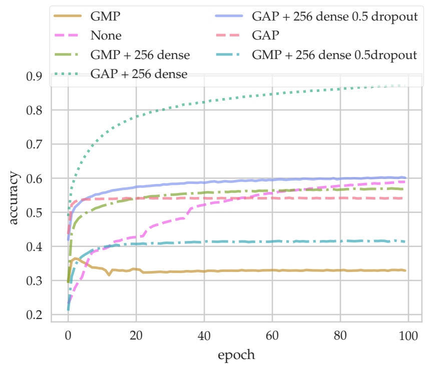
training validation

b Dataset *LAB44*

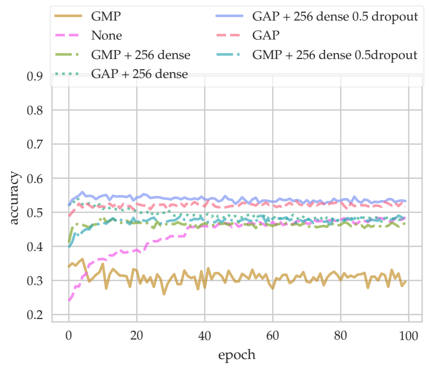


training validation

c Dataset *LAB20S44*

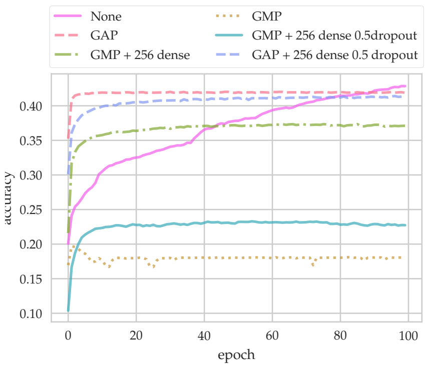


training

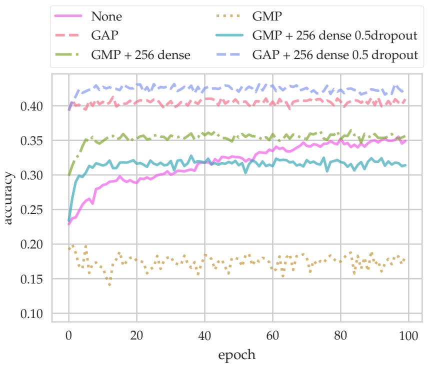


validation

d Dataset *CLS20*



training

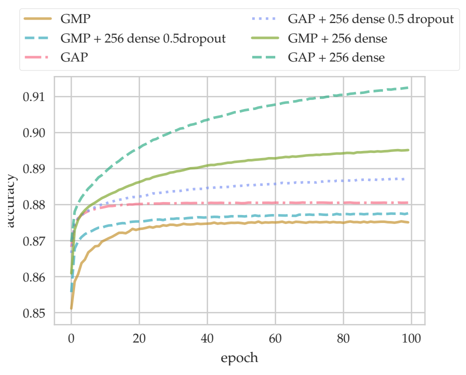


validation

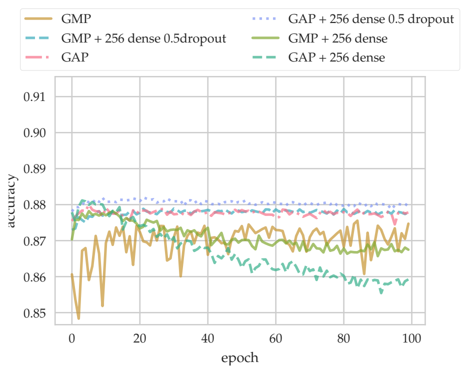
e Dataset *CLS44*

Figure B.9

B.3.4 Different top layers of Xception for transfer learning

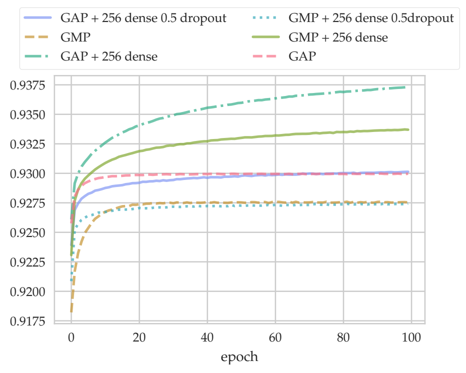


training

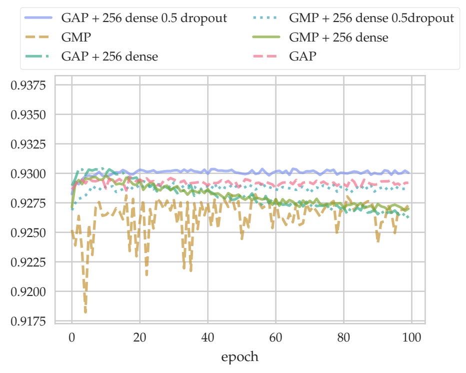


validation

a Dataset *LAB20*

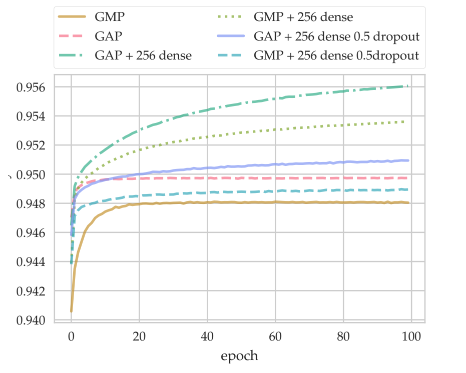


training

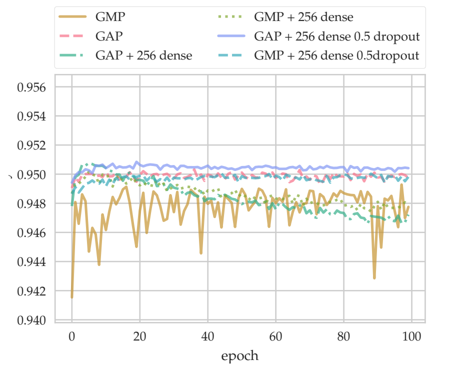


validation

b Dataset *LAB44*

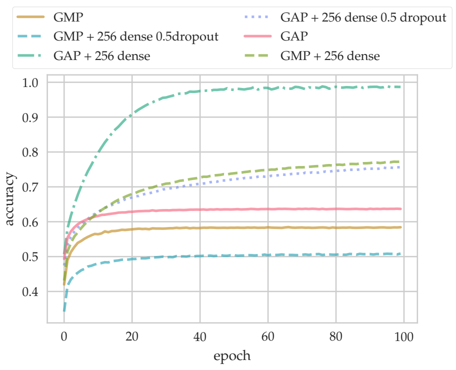


training

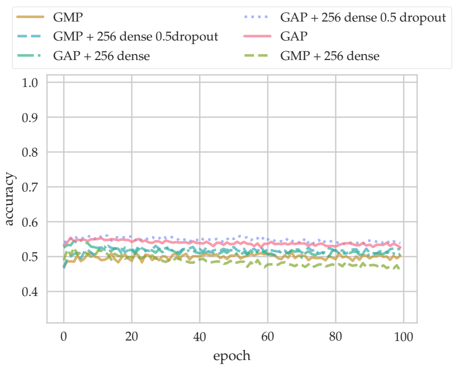


validation

c Dataset *LAB20S44*

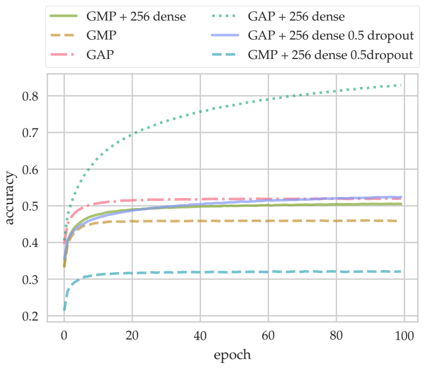


training

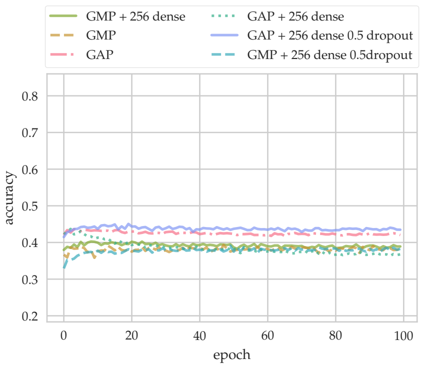


validation

d Dataset *CLS20*



training

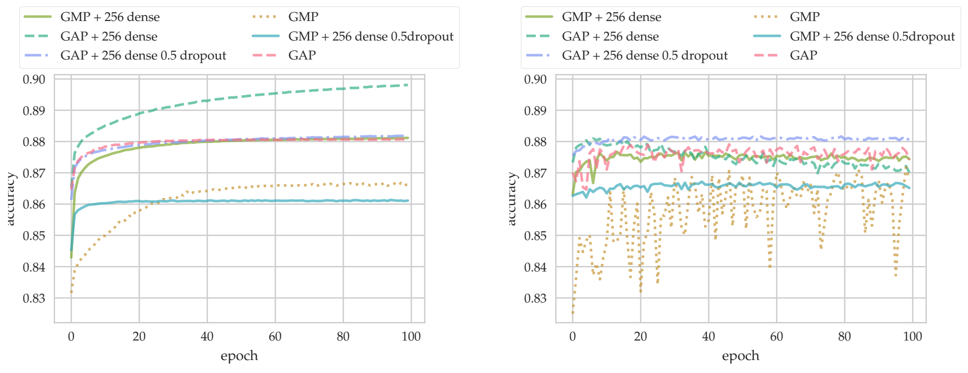


validation

e Dataset *CLS44*

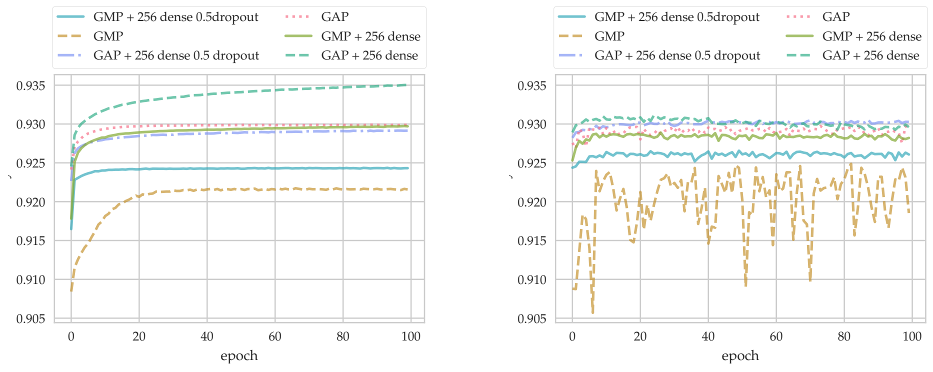
Figure B.10

B.3.5 Different top layers of Inception v3 for transfer learning



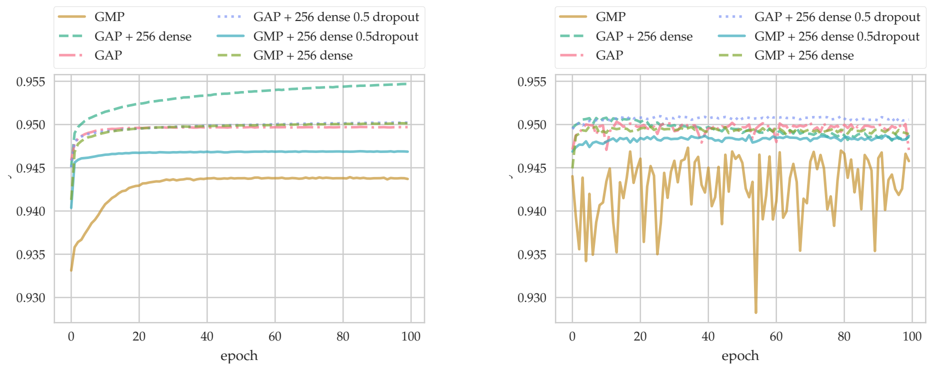
training validation

a Dataset *LAB20*



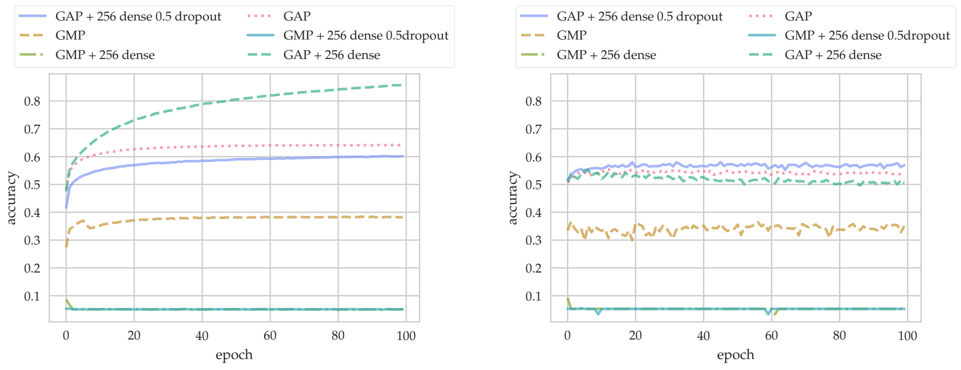
training validation

b Dataset *LAB44*



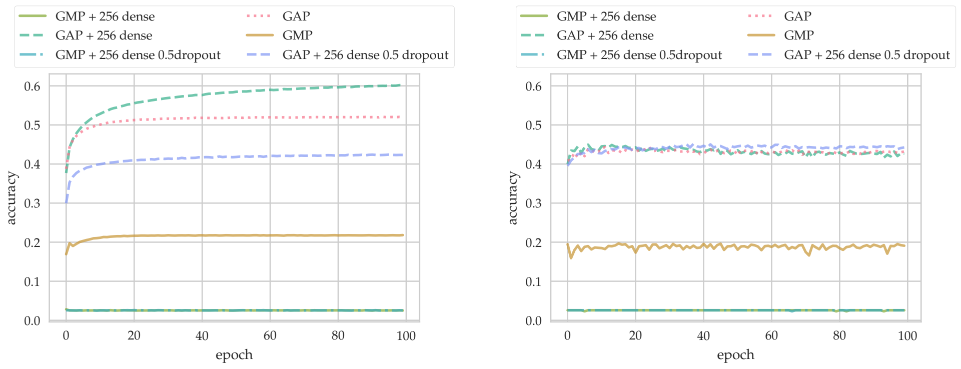
training validation

c Dataset *LAB20S44*



training validation

d Dataset *CLS20*



training validation

e Dataset *CLS44*

Figure B.11