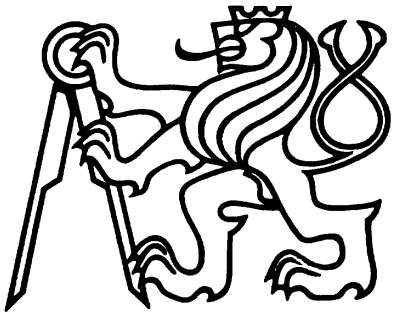
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská



DIPLOMOVÁ PRÁCE

2021 Bc. Adam Novotný

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
Katedra matematiky

DIPLOMOVÁ PRÁCE

Analýza satelitních dat pomocí metod strojového učení

Satellite data analysis using machine learning methods

Autor: Bc. Adam Novotný
Školitel: Ing. Adam Novozámský, Ph.D.
Akademický rok: 2020/2021

Katedra: matematiky Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Adam Novotný
Studijní program:	Aplikace přírodních věd
Studijní obor:	Aplikované matematicko-stochastické metody
Název práce (česky):	Analýza satelitních dat pomocí metod strojového učení
Název práce (anglicky):	Satellite data analysis using machine learning methods

Pokyny pro vypracování:

1) Seznamte se s problematikou strojového učení (metody příznakové i založené na konvolučních neuronových sítích) a jeho využití pro klasifikaci multispektrálních satelitních dat.
2) Nastudujte charakter multispektrálních dat ze satelitu Sentinel-2 a připravte trénovací a testovací sadu dat pro trénování algoritmů.
3) Navrhněte možnost generování trénovacích množin dat pomocí augmentace.
4) Navrhněte a implementujte řešení pro analýzu satelitních dat se zřetelem na jejich možnou variabilitu v rámci jednotlivých tříd, která může narušovat robustnost postupů.
5) Vyhodnoťte úspěšnost navrženého řešení.

Doporučená literatura:

1) W. K. Pratt, Digital Image Processing (3rd ed.), John Wiley, New York, 2001.
2) I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.
3) C. Shorten and T.M. Khoshgoftaar, A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60. 2019.
4) C. M. Bishop, Pattern Recognition and Machine Learning. Springer-Verlag, Berlin, Heidelberg, 2006.

Jméno a pracoviště vedoucího diplomové práce:

Ing. Adam Novozámský, Ph.D.
Ústav teorie informace a automatizace AV ČR, v.v.i., oddělení Zpracování obrazové informace, Pod Vodárenskou věží 4, 182 08 Praha 8

Jméno a pracoviště konzultanta:

RNDr. Michal Šorel, Ph.D.
Ústav teorie informace a automatizace AV ČR, v.v.i., oddělení Zpracování obrazové informace, Pod Vodárenskou věží 4, 182 08 Praha 8

Datum zadání diplomové práce: 31.10.2020

Datum odevzdání diplomové práce: 6.1.2021

Doba platnosti zadání je dva roky od data zadání.

Poděkování

Rád bych touto cestou poděkoval svému školiteli Ing. Adamovi Novozámskému, Ph.D. a konzultantovi RNDr. Michalovi Šorelovi, Ph.D. za odborné vedení diplomové práce a množství cenných rad. Dále bych chtěl poděkovat Státnímu zemědělskému intervenčnímu fondu a konkrétně Mgr. Renatě Bodnárové za poskytnutí plodinové mapy a informací ohledně zemědělských plodin.

Čestné prohlášení

Prohlašuji na tomto místě, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze dne 5. ledna 2021 .....

Bc. Adam Novotný

Název práce: **Analýza satelitních dat pomocí metod strojového učení**

Autor: Bc. Adam Novotný

Obor: Aplikované matematicko-stochastické metody

Druh práce: Diplomová práce

Vedoucí práce: Ing. Adam Novozámský, Ph.D., Ústav teorie informace a automatizace AV ČR, v.v.i., oddělení Zpracování obrazové informace, Pod Vodárenskou věží 4, 182 08 Praha 8

Konzultant: RNDr. Michal Šorel, Ph.D., Ústav teorie informace a automatizace AV ČR, v.v.i., oddělení Zpracování obrazové informace, Pod Vodárenskou věží 4, 182 08 Praha 8

Abstrakt:
Aplikace strojového učení a hloubkového učení vylepšila stávající a umožnila vznik nových technologií v oblasti počítačového vidění. Mise Sentinel-2 je specificky navržena tak, aby poskytovala satelitní snímky s vysokým rozlišením, které jsou vhodné pro pozorování vegetace na povrchu Země. Společně se satelitními snímky nám byly poskytnuty vzorky zemědělských dat z roku 2019 od Českého státního intervenčního fondu. Kombinací těchto dvou zdrojů dat byly připraveny datasety pro klasifikaci plodin do třinácti tříd. Na tyto předpřipravené datasety jsou aplikovány jak příznakové metody, tak i konvoluční neuronové sítě. Výsledky jednotlivých klasifikátorů a jejich výkon je pak analyzován, diskutován a porovnán.

Klíčová slova: klasifikace plodin, strojové učení, konvoluční neuronové sítě

Title: **Satellite data analysis using machine learning methods**

Author: Bc. Adam Novotný

Abstract:
The application of machine learning, and of its subset deep learning, has improved and enabled new techniques in the domain of computer vision. Sentinel-2 mission is specifically designed to provide high-resolution optical satellite imagery, which is suitable for monitoring surface vegetation. In addition to satellite images, we are provided with agricultural field data in 2019 by State Agricultural Intervention Fund. Combining these two data sources, datasets for field crop classification into thirteen classes are prepared. Both feature-based classifiers and convolutional neural networks are then applied on these pre-processed datasets and the results of each classifier and its performance are analyzed, discussed and compared.

Key words: field crop classification, machine learning, convolutional neural networks

Contents

List of notations

Notation	Description
ANN	artificial neural network
AP	average precision score
AUC	area under curve
CNN	convolutional neural network
CART	Classification and Regression Trees
DNN	deep neural network
EU	European Union
EC	European Commission
ESA	European Space Agency
EUMETSAT	European Organisation for the Exploitation of Meteorological Satellites
ECMWF	European Centre for Medium-Range Weather Forecasts
MLE	maximum likelihood estimation
MLP	multi-layer perceptron
MAP	maximum A posteriori
NDVI	normalized difference vegetation index
NIR	near-infrared (NIR)
PDF	probability density function
PR curve	precision-recall curve
ROC curve	receiver operating characteristic curve
SGD	stochastic gradient descent
SWIR	short-wavelength infrared
SAIF	State Agricultural Intervention Fund
UTM	Universal Transverse Mercator
VIS	visible spectrum
WGS84	World Geodetic System 1984

Introduction

Data analysis has grown to play a very important role in many fields of study. This phenomenon has been supported by a rapid development of machine learning, and especially deep learning, techniques over the past few years. Machine learning methods automatically detect hidden patterns in data based on various approaches. These include some long known feature-based algorithms as well as modern algorithms. Especially regarding deep learning, an active research is still ongoing with new findings almost every day. The usage of these algorithms in the field of computer vision not only improved past techniques but also enabled new discoveries. As optical satellite imagery becomes more and more available, the application of machine learning and computer vision techniques on this kind of data is more than fitting. One of the numerous applications is the statistical classification of crops grown on the fields. Field crop classification is a task especially complicated, as it requires sufficient amount of images throughout the whole year with sufficient spectral resolution. Both of the mentioned requirements are fulfilled by the Sentinel-2 mission.

The goal of this thesis is to apply both feature-based classifiers and convolutional neural networks on Sentinel-2 satellite imagery and find the classifier best suited for the task. This thesis begins with the sources of data samples and their description in chapter 1. Sentinel-2 satellite specification and satellite image properties, as well as the annotated agricultural fields data provided by the State Agricultural Invervention Fund, which are used as ground truth, are described in this chapter. Chapter 2 gives the theoretical background of machine learning theory and statistical model evaluation. In addition to this, the chapter contains a summary of selected feature-based algorithms. In chapter 3, artificial neural network s (ANNs) are presented, together with their construction, mathematical optimization and regularization. Convolutional neural networks (CNNs), as a subset of ANNs, their components and examples are also presented in this chapter. Final chapter 4 presents the application of feature-based classifiers and CNNs on prepared datasets. It explains the construction of two multi-temporal multi-spectral datasets, differing in allowed image size, and different steps of image pre-processing. The results of each classifier and its performance are then analyzed, discussed and finally compared.

Chapter 1

Data Description

One of the most important roles in the field of data analysis plays the data itself. There can be no beneficial outcome without the precise and accurate data. As the focus of this thesis is on **field crop classification**, the field satellite imagery and crop grown on the field, the data sources come in two shapes. These are multi-spectral satellite images in the first place and database of agricultural fields in the second.

In this chapter, a description of satellite imagery mission Sentinel-2 and agricultural field data provided by State Agricultural Intervention Fund is presented.

1.1 Sentinel-2 mission

Sentinel-2 is an Earth observation mission systematically providing high-resolution optical satellite imagery, as a member of the series of Earth observation missions. The mission is a part of the Copernicus Programme, which is the European Union (EU) programme managed and coordinated by the European Commission (EC) in partnership with the Member States, the European Space Agency (ESA), the European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT), European Centre for Medium-Range Weather Forecasts (ECMWF), EU Agencies and Mercator Océan. The Copernicus Programme is aimed at developing European information services based on satellite observation and in situ (non-space) data. The services offered by Copernicus address six thematic areas: atmosphere, marine, land and climate change monitoring, emergency and security. There is no restriction on use or reproduction and redistribution, with or without adaptation, for commercial or non-commercial purpose as the provided data are full, free and open. The Sentinel satellites are specifically designed to meet the requirements of the Copernicus Programme [8].

Radar imaging Sentinel-1 mission was launched in 2014 with the Sentinel-1A satellite, Sentinel-1B satellite was later launched in 2016. Sentinel-3 focusing mainly on marine observation was also launched in 2016. The other Sentinel missions (Sentinel-4, Sentinel-5 and Sentinel-6) are planned to be launched in the following years. In addition, it is planned to place a constellation of almost twenty additional satellites in orbit before 2030 [8].

Sentinel-2A and Sentinel-2B, with a multi spectral optical sensor, were launched in 2015 and 2017 respectively, both on a Vega rocket from Kourou, French Guiana [8]. Sentinel-2 is specifically designed to provide optical imaging for land ecosystems monitoring, including imagery of vegetation, soil and water cover, inland waterways and coastal areas. Continuity of the mission will be ensured by the launch of Sentinel-2C and Sentinel-2D.

The Sentinel-2 mission objectives are

- systematic global acquisitions of high-resolution, multi-spectral images with a high revisit frequency,
- enhanced continuity of multi-spectral imagery as a legacy of SPOT missions,
- observation data for the next generation of operational products, such as land-cover maps, land-change detection maps and geophysical variables [10].

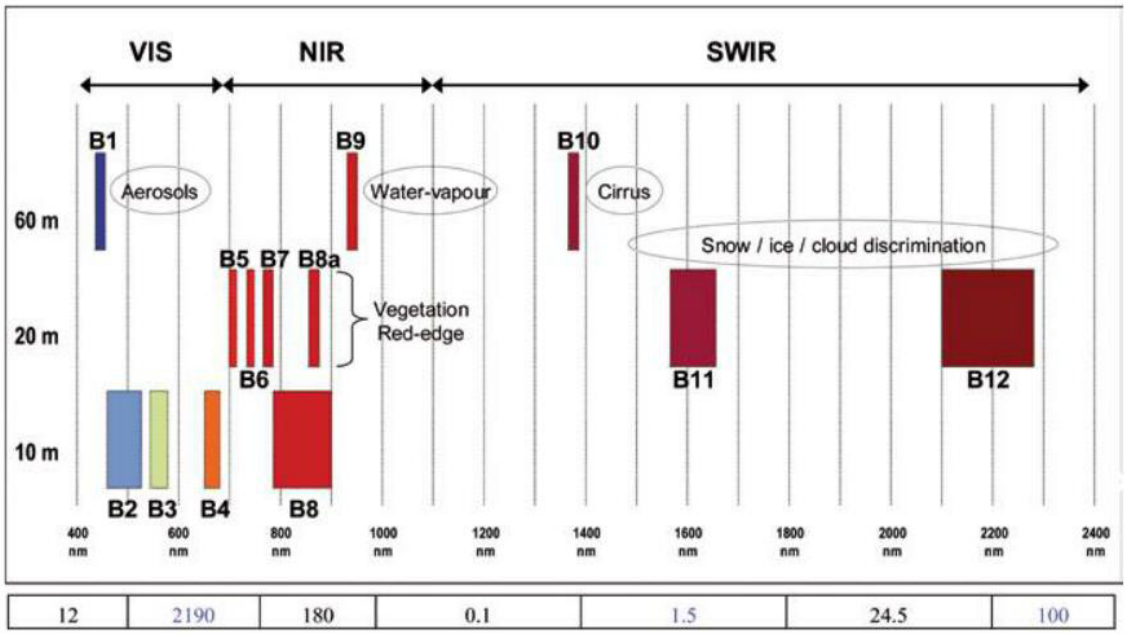
Facing a field crop classification problem, the Sentinel-2 satellite visual imagery makes a fitting and reliable source of image data.

1.1.1 Satellite specification

The Sentinel-2 mission is composed of two polar-orbiting satellites (Sentinel-2A and Sentinel-2B) orbiting Earth in the same sun-synchronous orbit phased by 180° at a mean altitude of 786 km. Revisit time of a single satellite is ten days at the Equator, making it five days for both satellites. This time is even shortened at high latitudes as satellites' swaths overlap. As an example, Prague revisit time is two and three days repeatedly. The latitude sensing interval is between latitudes 56° south and 84° north (Cape Horn in South America and north of Greenland) including major islands (greater than 100 km^2) with the orbital swath width of 290 km [10].

The satellites' optical sensor MSI gives thirteen bands ranging from visible spectrum (VIS) to near-infrared (NIR) and short-wavelength infrared (SWIR) spectrum. Four of the bands have the resolution of 10 meters, six bands of 20 meters and three bands of 60 meters. This sensor is first of its kind, as it contains three so called *vegetation red-edge* bands, with its parameters specifically designed to be ideal for vegetation, forest or leaf chlorophyll monitoring [40]. The full specification can be seen in the figure 1.1.

Figure 1.1: Thirteen spectral bands of Sentinel-2 satellite [40].



In addition to the thirteen bands, another indices can be derived using those bands. These indices, such as normalized difference vegetation index (NDVI), are specifically constructed to obtain particular information, e.g. about vegetation. As an example, NDVI assesses the presence of live green vegetation and is defined as NDVI $= \frac{B8-B4}{B8+B4}$.

1.1.2 Satellite image data

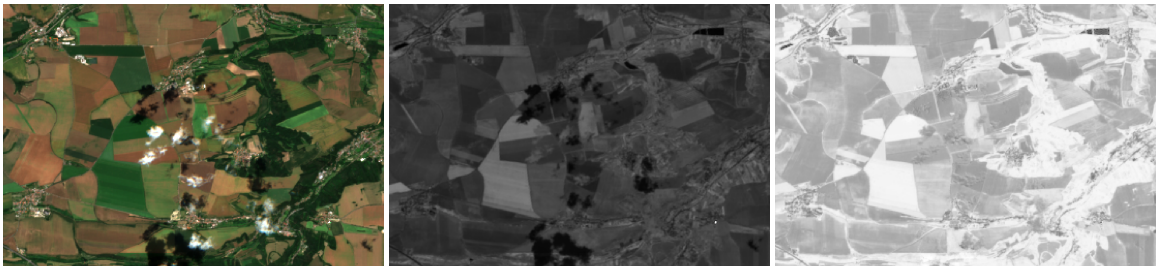
The Sentinel-2 data come in various levels (or products) of pre-processing. Level-0 contains raw compressed data, Level-1A decompresses Level-0 data and computes corresponding radiometric, spectral and geometric (i.e. Earth location) correction and calibration. These two levels are not accessible to the public. Level-1B, the lowest pre-processed product accessible to the public, contains the *top-of-atmosphere radiance* with the corrections and calibration computed in Level-1A and is composed of so called *granules* of size 25×23 km. Level-1C contains the *top-of-atmosphere reflectance* computed from Level-1B data and orthorectified in Universal Transverse Mercator (UTM) and World Geodetic System 1984 (WGS84) coordinate systems with resampled granules resolution of size 100×100 km, called *tiles*.

Level-2A contains the *bottom-of-atmosphere reflectances* with atmospheric corrections with the same size as Level-1C. The atmospheric corrections are computed by the sen2cor module provided by the ESA. A comprehensive study between multiple atmospheric correction algorithms, such as sen2cor or MAJA, and multiple surfaces can be found in [38]. As an example from [38], sen2cor gives the best overall atmospheric correction measured by coefficient of determination R^2 metric between the true and computed atmospheric correction. In addition, Level-2A contains scene classification mask with twelve classes, e. g. vegetation, water, clouds with medium probability, with high probability, snow or others [11]. As the Level-2A is the highest pre-processed product, this level is used as a source of data in this thesis. Example of a Sentinel-2 Level-2A image can be seen in the figure 1.2.

The Copernicus Programme open, full and free policy enables Sentinel-2 data to be accessed straightforwardly. There are two ways to obtain the data as any registered user can access them; by means of graphical user interface or by the APIs. These APIs offer primarily query tool with predefined keywords for extracting unique identification numbers of a given tile. These keywords include among others satellite type, sensing time, geographical location etc. Secondarily, the APIs offer a downloading tool, which is based on the identification number. These services are offered by Copernicus Open Access Hub, but in addition, Czech Sentinel Collaborative Ground Segment (CollGS) [40] and its data storage contains tiles exclusively located within the borders of the Czech Republic and contains even older tiles.

Figure 1.2: Sentinel-2A Level-2A image example sample with resolution of 20 m.

(a) Truecolor (RGB) image (b) B8a vegetation red-edge (c) NDVI



1.2 Agricultural data

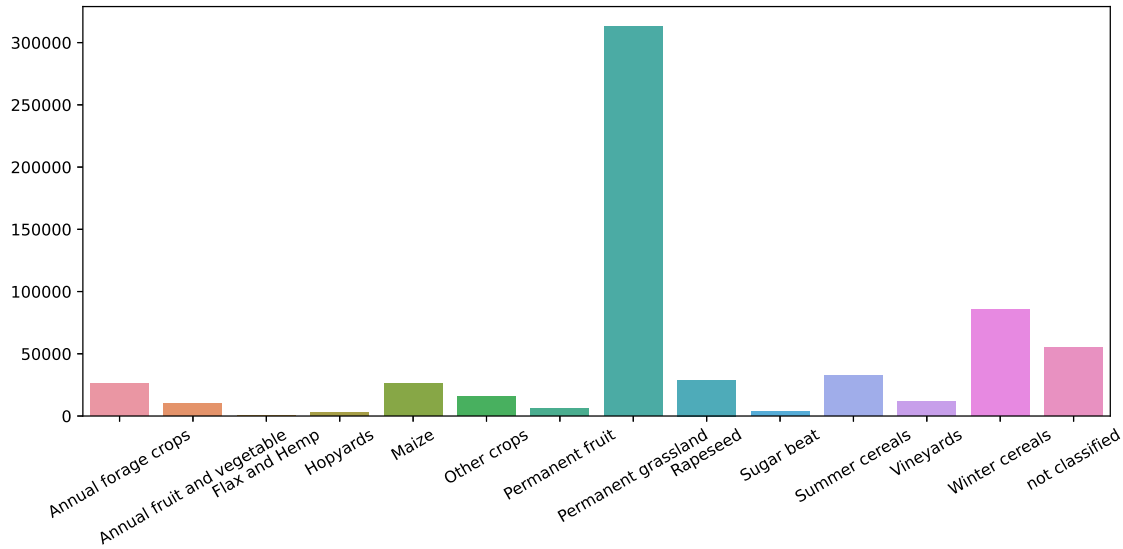
Agricultural data are provided by **State Agricultural Intervention Fund (SAIF)** (***Státní zemědělský intervenční fond (SZIF)***), which are stored in geographical information system (GIS) data format called *shapefile*. The whole shapefile contains 617,203 entries in 2019. These entries are obtained by the sen4cap algorithm [9] by ESA.

Every entry consists of projected coordinates of its bounding box and polygon in UTM coordinate system. In addition, it contains its area, unique identification number called field block (*půdní blok*) and its parts (*díl půdních bloků*), culture code within the Czech agriculture registry Land Parcel Information System (LPIS) [25] and crop grown on the field.

Field block represents the clearly separable area of maintained soil, its parts then contains all information about field maintenance, as the user, culture and other additional information [25]. Culture code represents the type of culture of a given field. Overall, there are fourteen unique culture codes given by LPIS. More information can be found in [25].

There are fourteen unique grown field crops, namely *Annual forage crops*, *Annual fruit and vegetable*, *Flax and Hemp*, *Hopyards*, *Maize*, *Other crops*, *Permanent fruit*, *Permanent grassland*, *Rapeseed*, *Sugar beat*, *Summer cereals*, *Vineyards*, *Winter cereals* and finally *not classified*. The figure 1.3 shows the overall count of each crop in the Czech Republic in 2019. SAIF validated that at least 90 % of the labels are correct by direct observation of limited fields sample.

Figure 1.3: Counts of crops grown in the Czech Republic in 2019.



Chapter 2

Machine Learning

Machine learning, as a subset of **artificial intelligence**, provides automated methods of data analysis and information retrieval and goes hand in hand with the area of **pattern recognition**. It is understood as a set of algorithms that automatically detect patterns and regularities in data and then use the retrieved patterns for further inference or other kinds of decision making under uncertainty without using explicit instructions. Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions.

A machine learning algorithm is an algorithm that is able to learn from data. Learning is in the area of computer science defined as any computer program that improves its performance at some task through experience. More precisely, a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. Examples can be found in [28, 15].

2.1 Statistical learning theory

Machine learning algorithms can be divided into several groups. In the **supervised** learning approach, the goal is to find general mapping of inputs to outputs, given a dataset of corresponding inputs and labeled outputs $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N}$, where $N \in \mathbb{N}$ is the number of observations. As both inputs and outputs can be multidimensional for each observation, these can be written as $\forall i \in \mathbb{N} : \boldsymbol{x}^{(i)} = (x_1^{(i)}, \ldots, x_{\mathcal{I}}^{(i)})$ as a realization of random variable $\boldsymbol{X}$ and $\boldsymbol{y}^{(i)} = (y_1^{(i)}, \ldots, y_{\mathcal{O}}^{(i)})$ of random variable $\boldsymbol{Y}$ respectively, where $\mathcal{I}$ is the dimension of an input vector and $\mathcal{O}$ the dimension of an output vector. In case of statistical **classification**, the output (or response variable) is categorical variable, the mapping can be then written as $f : \mathbb{R}^{\mathcal{I}} \to \{1, \ldots, \mathcal{O}\}$. In case of **regression**, the output can be written as $f : \mathbb{R}^{\mathcal{I}} \to \mathbb{R}^{\mathcal{O}}$.

In the **unsupervised** learning approach, the dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)})\}_{i=1}^{N}$ is only available, as no labels are provided. The goal is to find useful properties of the structure of the dataset. As an example, this could mean discovering hidden patterns, estimating the probability density function (PDF) or clustering. **Reinforcement** learning approach is concerned with behavior of agent with a set of possible actions and rewards for taking such actions. The focus is on finding a balance between exploration of the unknown and exploitation of current knowledge. One can also encounter **semi-supervised** learning, which is hybridization between supervised and unsupervised learning and is concerned with typically small amount of labeled data and large amount of unlabeled data.

As hinted in chapter 1, we are particularly interested in classification task T with experience E gained from labeled dataset, thus supervised learning. Performance P and its measures will be discussed in subsection 2.2.

Let $\mathcal{C}$ be the number of classes in a classification problem. In case of classification into $\mathcal{C} = 2$ classes, which is called **binary** classification, it is convenient to use a one-dimensional output representation in a form of label $y \in \{0, 1\}$ such as that $y = 1$ corresponds to class C_1 and $y = 0$ corresponds to class C_0. If $\mathcal{C} > 2$, thus in **multiclass** classification, it is convenient to use a hot-encoding scheme in which $\boldsymbol{y}$ is a vector of length $\mathcal{C}$ such as that if the observation is of class C_k, then all the values of $\boldsymbol{y}$ are equal to zero except of y_k, which is equal to one. This way, number of classes $\mathcal{C}$ can be perceived as number of outputs $\mathcal{O}$ of a given classifier.

2.1.1 Generalization

Generally in a machine learning problem, we deal with the discrepancy between the true but unknown data generating distribution p_{data}, empirical distribution defined by the dataset $\hat{p}_{data}$ and distribution restrained by our model p_{model}. In a frequentist approach, model distribution is a family of distribution parametrized by random variable $\boldsymbol{\theta}$ under the assumption that the p_{data} lies within the model family. It can be shown [15] that by applying maximum likelihood estimation (MLE) on parameter $\boldsymbol{\theta}$, we minimize the degree of dissimilarity between $\hat{p}_{data}$ and p_{model} measured by the Kullback–Leibler divergence. We can thus see this approach as an attempt to make the model distribution match the empirical distribution $\hat{p}_{data}$. The correspondence between minimizing the KL divergence and maximizing log-likelihood is an important step for learning of machine learning algorithms and construction of the **loss function** L, which captures the discrepancy between output from the model $\hat{\boldsymbol{y}}$ and the output from the data distribution $\boldsymbol{y}$.

Ideally, the p_{model} should match the real data distribution p_{data}. As a result, knowing only dataset $\mathcal{D}$, thus $\hat{p}_{data}$, does not guarantee the classifier to perform well on previously unseen data. This phenomenon is known as **generalization**. For this reason, $\mathcal{D}$ is further divided into three parts: training, validation and testing set. During the training phase, the classifier is trained to achieve desired performance on a training set. The training process is different for each classifier. This will be discussed in section 2.3 for a given classifier and then for artificial neural networks in chapter 3. The **training error** is computed and then further reduced by the means of performance measure. Simply reducing the training error would be an optimization problem, but as mentioned, it is desirable to reduce the **generalization error**, also called **test error**, as well. In order to generalize well, all samples in all three sets are assumed to be *iid* samples. This means that all the samples from dataset $\mathcal{D}$ are sampled independently and from the same distribution. The validation and testing phase are further elaborated on in section 2.2.

In order for the classifier to generalize well, machine learning algorithms need to be able to make the training error small and subsequently make the gap between training and test error small. These correspond to two challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to minimize the training error sufficiently, on the other hand overfitting occurs when the model achieves small training error but retains high test error. This corresponds to common statistical phenomenon called bias-variance trade-off. The properties of both underfitting and overfitting can be described in terms of the **capacity** of the model. Capacity of the model can be altered by a choice of hypothesis space $\mathcal{H}$ which enables only a particular subset of functions for a given

classifier. As an example, linear classifier has only the set of linear functions of its inputs as its hypothesis space.

Typically, during training the training error decreases with more data it trains on and test error has a U-shaped curve. The optimal time to stop the training is when the test error reaches its minimum and starts to rise. There is still an active research concerning so-called double U-shaped curve, which claims that with even higher capacity the test error declines again [4].

Generalization in machine learning offers a look at the difference between pure optimization and machine learning problem. Machine learning usually acts indirectly. The focus is on minimalization of an expected value of the loss function, done with respect to the training set $\hat{p}_{data}$, not with respect to the true underlying data distribution p_{data}. This is the case even though that the performance measure P is what is important to us, but the problem is that it may be intractable [15].

2.1.2 Optimization

Traditional, as well as specialized algorithms are used to solve the arising machine learning problem. So far the parameters $\boldsymbol{\theta}$ were discussed, but not their optimality. As mentioned, loss function L measures the discrepancy between predicted output and actual output of the classifier and is a function of $\boldsymbol{\theta}$.

As the machine learning optimization is done with respect to the $\hat{p}_{data}$, the so called **empirical risk function**, which is defined as a sum of individual losses for each input-output pair, is minimized. The empirical risk function is is also referred to as overall loss function, defined as

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} L\left(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)}\right) = \frac{1}{N}\sum_{i=1}^{N} L\left(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}\right). \tag{2.1}$$

It should be emphasized that any empirical risk, as a sum of individual losses, consisting of a negative log-likelihood is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by model [15]. As a result, minimizing the negative log-likelihood (or maximizing the log-likelihood) is the same as minimizing the cross-entropy between the empirical distribution defined by $\mathcal{D}$ and the probability distribution defined by the model. Therefore, by minimizing the cross-entropy loss function, the negative log-likelihood is also minimized and vice versa.

Examples can be seen in section 2.3. For linear regression, it is common to minimize mean square error, which is the cross-entropy between the empirical distribution and the Gaussian model, which corresponds to minimizing negative log-likelihood of Gaussian model. Also, logistic regression minimizes the negative log-likelihood with Bernoulli model assumption. For artifical neural networks, this is further elaborated on in section 3.3.

2.1.3 Regularization

One of the universal ways to avoid overfitting is **regularization**. As mentioned in 2.1.1, altering and limiting the hypothesis space $\mathcal{H}$ enables the change of the model capacity. Regularization techniques allow to add additional information to the optimization problem, thus alter the $\mathcal{H}$. Regularization is defined as any modification that is intended to reduce test error but not training error [15]. This can be, among others, done by adding a regularization

term R to the loss function L which acts like a penalty. This way, classifier is given a preference of one solution over another.

By adding a regularizing term $\lambda\Omega(\boldsymbol{\theta})$ to the overall loss function, the regularized overall loss function $\widetilde{J(\boldsymbol{\theta})}$ can be defined as

$$\widetilde{J(\boldsymbol{\theta})} = J(\boldsymbol{\theta}) + \lambda\Omega(\boldsymbol{\theta}), \tag{2.2}$$

where $\lambda \geq 0$ is a hyperparameter that weights $\Omega(\boldsymbol{\theta})$ against $J(\boldsymbol{\theta})$. Obviously, for $\lambda = 0$, the regularized loss function yields overall loss function, which is unregularized. Hyperparameters, their importance and choice, are discussed in 2.2.

So far we have talked about frequentist approach. Many regularized estimations, such as MLE, can be interpreted as Maximum A posteriori (MAP) estimation, which equals to the maximum of posterior distribution in Bayesian approach. As an example, MLE with regularization term equal to L_2 norm of parameters can be perceived as MAP estimation.

2.2 Model evaluation

Having discussed task T and experience E in a machine learning approach, the focus will move onto the performance P. Having trained classifier, its performance P needs to be estimated. For that reason, validation of model needs to be carried out. In section 2.1, validation set has not yet been used, but will be used for validation.

2.2.1 Hyperparameters and validation

Most machine learning algorithms have initial settings that need to be set before actual training by the user, which allows to control the behavior of the algorithm. These different settings are called **hyperparameters**. By contrast, the parameters of the model are learned during the training phase. It solely depends on the nature of the algorithm, as some algorithms require none and some require a high number of them as will be seen in chapter 3. For the right setting of hyperparameters, validation set, which was *not* used during the training phase, is used.

Validation set is used to estimate the generalization error, thus acts like a fake test set for a given hyperparameter. After training the classifier for a given hyperparameter and validating, the best performing hyperparameter is chosen. For finding a set of hyperparameters, various techniques are being used. These include brute grid search, random search, evolutionary algorithms or even a nested learning algorithm in which one algorithm learn a set of hyperparameters for another algorithm.

Additionally, a technique called **cross-validation** can be used which provides a stochastic element to enable generalization. In this setting, training set and validation set are repeatedly partitioned into complementary subsets and always trained and validated in a described manner. Exhaustive methods, which partition the sets in every possible combination, is not commonly used due to computational requirements. Instead, non-exhaustive cross-validation like k-fold cross-validation, in which dataset $\mathcal{D}$ without test set is split into k complementary subset with $k-1$ sets acting as training set and always using the final one as validation set. The training and validation is repeated k times.

In case of unbalanced dataset, other techniques like stratified k-fold cross-validation, which subsequently splits the training and validation sets so each fold contains roughly

the same number of classes. Another way to approach unbalanced dataset is to either over-sample or under-sample datapoints in the dataset.

The generalization error can be the estimated as an average of generalization errors across k trials. In summary, cross-validation combines (averages) measures of fitness in prediction to derive a more accurate estimate of model prediction performance. Empirically, cross-validation is highly successful on many real-world tasks [15]. There is a bias-variance trade-off for the choices of k, but it is empirically common to choose $k = 5$ or $k = 10$ as these values suffer neither from high bias nor high variance. It is important to notice that test set is left untouched until the final model evaluation.

2.2.2 Classification metrics

Performance P of a classifier is measured by the classification metrics. Usually the performance P is specific to the task T. As a consequence, there are different metrics for regression and classification. Additionally, there are also a lot of metrics for each of them. For classification, different metrics focus on different aspects of model evaluation and are useful in different scenarios.

Binary classification

Further on, let us consider classification only; and binary classification at first. In this setting, we have labeled output y and output from classifier $\hat{y}$. In many cases, before getting an actual $\hat{y}$ as the output of a classifier, the probability or the normalized score of $y = 1$ (which is a number from $[0, 1]$ which is looked at as probability) is obtained as an output of a classifier. The threshold $t \in [0, 1]$ can be arbitrarily set and whenever a probability (or score) is higher than the threshold, the class is set at $\hat{y} = 1$, and whenever the threshold is smaller or equal than the threshold, the class is set at $\hat{y} = 0$.

The threshold can be moved from zero to one and predicted class $\hat{y}$ set accordingly. For example, with threshold set as zero all observations are automatically predicted as ones and with threshold set as one all are predicted as zeros. To examine all classification scenarios, the threshold t takes values of all probabilities (scores). It is not uncommon to see the threshold set at $t = 0.5$, thus whenever a score is greater than 0.5, the $\hat{y} = 1$. As will be explained, more optimal threshold can be chosen.

The labeled output y and output from classifier $\hat{y}$ is thus obtained. By comparing y and $\hat{y}$, four scenarios might happen. Whenever $y = \hat{y}$, the class was correctly classified, on the other hand when $y \neq \hat{y}$, the class was incorrectly classified. This gives four distinct groups with two types of errors:

1. TP: the number of true positives,
2. TN: the number of true negatives,
3. FP: the number of false positives,
4. FN: the number of false negatives.

Obviously $\mathsf{TP} + \mathsf{TN} + \mathsf{FP} + \mathsf{FN} = N$ holds. These four numbers can be written as a **confusion matrix** for visualization, which can be seen in table 2.1. Diagonal elements represent the correctly classified observations and non-diagonal represent mislabeled observations. Rows in confusion matrix indicate labels and columns indicate predictions. Please

note that one might encounter confusion matrix defined as transposition of our confusion matrix, or rows and columns interchanged. Additionally, confusion matrix can be also normalized so each row equals to 1.

Table 2.1: Confusion matrix for binary classification.

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	true negative (TN)	false positive (FP)
$y = 1$	false negative (FN)	true positive (TP)

The most natural performance metric is **accuracy**, which is defined as the ratio of correctly classified observations against all observations and can be expressed as

$$\mathsf{accuracy} = \frac{\mathsf{TP} + \mathsf{TN}}{\mathsf{TP} + \mathsf{TN} + \mathsf{FP} + \mathsf{FN}}. \tag{2.3}$$

Error is then simply defined as $\mathsf{error} = 1 - \mathsf{accuracy}$, giving the same information as accuracy. This applies for all metrics and their subtraction from one. Now the threshold part comes into play. The threshold t can be chosen so the classifier maximizes the overall accuracy, often being different than 0.5. Accuracy is computed at each threshold and then the threshold yielding maximum accuracy is chosen.

Accuracy and error are widely used performance metrics, but often not sufficient to use. As an example, they are not a good metric in case of unbalanced dataset as one label might outweigh the other one in computing accuracy. As can be seen, confusion matrix holds more information than accuracy. For this reason, weighted alternatives are introduced.

There are several derived metrics from confusion matrix elements. All of them weigh elements of confusion matrix differently and **precision** and **recall** are two of them. Precision takes the mean of correctly classified against all in the first column of confusion matrix and recall takes the mean of correctly classified against all in the first row confusion matrix. They are thus defined as

$$\mathsf{precision} = \frac{\mathsf{TP}}{\mathsf{TP} + \mathsf{FP}}, \quad \mathsf{recall} = \frac{\mathsf{TP}}{\mathsf{TP} + \mathsf{FN}}. \tag{2.4}$$

There is also a metric that combines retrieved information from precision and recall called F_1 **score**. It is defined as harmonic mean between precision and recall, thus

$$F_1 = \frac{2}{\mathsf{precision}^{-1} + \mathsf{recall}^{-1}} = 2\,\frac{\mathsf{precision} \cdot \mathsf{recall}}{\mathsf{precision} + \mathsf{recall}}. \tag{2.5}$$

Similarly to accuracy, F_1 score is also used for choosing an optimal threshold.

Another metrics are **specificity** and **sensitivity**. Specificity takes the mean of correctly classified against all in the second row of confusion matrix and sensitivity is just another term for recall. They are thus defined as

$$\mathsf{specificity} = \frac{\mathsf{TN}}{\mathsf{TN} + \mathsf{FP}}, \quad \mathsf{sensitivity} = \mathsf{recall}. \tag{2.6}$$

Note that identities like $\frac{\mathsf{FP}}{\mathsf{TP}+\mathsf{FP}} = 1 - \mathsf{precision}$ or similarly for other metrics hold. As has been seen, the precision, recall, specificity and sensitivity take the mean of correctly classified against all in either row or column, so not to overly complicate the terminology

and to allow generalization for multiclass classification, the terms are simplified. It is being said that mean of correctly classified against all in first column is precision for the first class and mean of correctly classified against all in second column is precision for the second class; and the same for rows yielding two recalls. F_1 score is computed also for each class. For binary classification, we thus obtain two means in rows, two means in columns and two F_1 scores, thus six metrics. It is common to observe the metrics for the positive class $y = 1$, as positive class is the point of interest of classification.

Receiver operating characteristic (ROC) curve is graphical plot of $1 - \mathsf{specificity}$ against $\mathsf{sensitivity}$ at various classification thresholds. ROC curve thus plots the mean in first row against the mean in the second row of the confusion matrix at various thresholds. In each threshold setting, confusion matrix and thus $\mathsf{specificity}$ and $\mathsf{sensitivity}$ are obtained. This way, ROC curve can be plotted. ROC curve is monotonic. Metric based on ROC is **Area under curve (AUC)**. AUC is simply an integral of ROC curve from zero to one. One should notice AUC does not depend on a given threshold.

Similar plot to ROC curve is **precision-recall (PR)** curve, which uses the same logic and plots recall of positive class on x-axis and precision of positive class on y-axis. For balanced dataset, the graph for $x = 1$ ends in $y = \frac{1}{2}$. PR curve is not monotonic, because $\mathsf{precision}$ is not necessarily a monotonic function. For undefined values for precision, the value is set at 1 for the plot, thus the the first precision and recall in the PR curve values are 1. and 0. respectively (thus for $x = 0$ we have $y = 1$). This ensures that the graph starts on the y axis. Note that PR curve is different for positive and negative class. Empirically, ROC curves should be used when there are roughly equal numbers of observations for each class and precision-recall curves should be used in case of unbalanced dataset. AUC can also be computed PR curve, but it is common to compute average precision score AP defined as $(\mathsf{AP} = \sum_t (\mathsf{recall}_t - \mathsf{recall}_{t-1})\mathsf{precision}_t)$.

In case of k-fold cross-validation, for each fold the given metric is computed. Statistics like mean, standard deviation and confidence intervals of the metrics can then be computed.

Multiclass classification

The situation is a bit more complicated for multiclass classification. Instead of two possible label y and the output $\hat{y}$ values in case of binary classification, the label $\boldsymbol{y}$ and the output $\hat{\boldsymbol{y}}$ are now $\mathcal{C}$-dimensional. In multiclass setting, we obtain probabilities (scores) of each class. Instead of choosing a threshold t, the $\hat{\boldsymbol{y}}$ is taken as maximum class score.

Confusion matrix, accuracy and error can be constructed in a similar fashion. The only difference is in the number of correctly and incorrectly classified labels. Instead of just four numbers (2^2), there is now $\mathcal{C}^2$ numbers.

Precision and recall are simply means of correctly classified against all in columns and rows respectively, as explained before. We thus obtain $\mathcal{C}$ precisions, $\mathcal{C}$ recalls and $\mathcal{C}$ F_1 scores, therefore $3\mathcal{C}$ metrics.

In order to extend ROC curve and PR curve, the label and the output need to be binarized. The curves can be drawn in two different settings. First is to draw the curve for each class, taking the probability of each class and the remaining probability (*one-vs-all* approach), which yields $\mathcal{C}$ curves. In one-vs-all approach, the positive $y = 1$ class is chosen to be the *one* variable. The second is to take two classes against each other (*one-vs-one* approach) yielding $\binom{\mathcal{C}}{2}$ curves. The costly alternative is to train a multitude of binary classifiers, also either in one-vs-all or one-vs-one approach.

The performance of a classifier is estimated during the training phase on the training

set, during the validation on the validation set and during the testing phase on the testing set, thus on the whole dataset $\mathcal{D}$. It is common to observe accuracy on the training and validation sets and a multitude of different metrics on test set.

2.3 Algorithms

In this section, a concrete example of machine learning algorithms and their idea is given. The main focus is on linear models and decision trees and their ensembles. Please note that this list is not at all exhaustive. Algorithms like k-nearest neighbors or support vector machines should be at least mentioned. As this thesis is also interested in artifical and convolutional neural networks, which are thoroughly explained in chapter 3, it is important to understand the mentioned algorithms, as their ideas are used and further extended.

2.3.1 Linear models

The most basic classifiers are linear classifiers. Linear classifier can be perceived as a mapping function $f(\cdot)$ with inputs as linear combinations of the input vector $\boldsymbol{x}$. The output of a linear classifier is basically

$$y = f\left(\sum_{j=1}^{\mathcal{I}} w_j x_j + w_0\right), \tag{2.7}$$

where $\forall j \in \{1, \ldots, \mathcal{I}\} : w_j \in \mathbb{R}$ is vector of weights and $w_0 \in \mathbb{R}$ is a threshold, all often conveniently written as a weight vector $\boldsymbol{w}$, defining $x_0 = 1$, resulting in

$$y = f\left(\sum_{j=0}^{\mathcal{I}} w_j x_j\right). \tag{2.8}$$

In the simplest case, the function $f(\cdot)$ is the Heaviside step function $\Theta(\cdot)$. A subset of linear classifiers is **logistic regression**, often in reality used as an initial classifier. Logistic regression belongs to the class of generalized linear models [18]. Generalized linear models are in literature often rewritten as

$$g(\mu) = \sum_{j=1}^{\mathcal{I}} \beta_j x_j + \beta_0 = \boldsymbol{x}^T \boldsymbol{\beta}, \tag{2.9}$$

where $\mu = \mathbb{E}\boldsymbol{Y}$ and $g(\cdot) = f^{-1}(\cdot)$ is called link function. In case of logistic regression, the *logit* link function takes form of

$$g(\mu) = \text{logit}(\mu) = \log\left(\frac{\mu}{1-\mu}\right). \tag{2.10}$$

Finding the optimal set of $\boldsymbol{\beta}$ can be achieved by taking MLE of Bernoulli distribution. It is not possible to find analytic expression and the parameters are found numerically, often by taking iteratively reweighted least squares algorithm, or by gradient-based optimization techniques. Logistic regression applies for binary classification, but it can be generalized to multinomial logistic regression as well.

Linear classifiers can be regularized. This is done be adding regularizing penalty in form of euclidean norm $\|\boldsymbol{\beta}\|$ to the log-likelihood. Adding L_1 norm $\|\boldsymbol{\beta}\|_1 = \sum_{j=0}^{p} |\beta_j|$ is also

called least absolute shrinkage and selection operator (LASSO), adding L_2 norm $\|\boldsymbol{\beta}\|_2$ is called ridge regression and adding both regularizing norms is called elastic net regression. In case of these regularizing techniques, there are one, one and two model hyperparameters respectively.

Linear models have many limitations, which is historically often portrayed on inability to learn the XOR function defined as $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(4)}) = ([0,0],[0,1],[1,0],[1,1])$ and $(y^{(1)}, \ldots, y^{(4)}) = (0,1,1,0)$. This problem can be addressed by generalizing the linear models to nonlinear models by applying the **kernel method** (also called kernel trick) by replacing the input with its transformation into higher-dimensional feature space. Logistic regression has its own generalized version called kernel logistic regression.

2.3.2 Decision trees and ensemble methods

The alternative to the linear models are the decision trees and their ensembles. Decision trees, as an decision support tool, have commonly been used in the decision making process, often manually constructed. As a machine learning algorithm, they provide a prediction, either in regression or classification setting, based on predefined tree splitting rules. Decision trees have suffered with overfitting so in order to deal with such behavior, ensemble methods were introduced.

Decision trees

Decision tree is a graph $(\mathcal{V}, \mathcal{E})$ of set of vertices $\mathcal{V}$ also called *nodes* and set of edges $\mathcal{E}$ organized in a hierarchical order. Nodes are further divided into root node, internal (also called split) and terminal (also called leaf) nodes. Internal and terminal nodes have exactly one incoming edge and all internal nodes have in case of binary trees exactly two outcoming edges. It is important to note that a there is a bijection between any general ordered tree and binary tree, so the subsequent explanation focuses on binary trees exclusively. The root and each internal node is labeled with an input feature x_j and each leaf gives the prediction y_k, which in case of classification corresponds to scores, which are looked at as probabilities of a given class given training data. The edges correspond to a feature values of a given parent node feature.

There are several advantages and disadvantages of using the decision trees. The main advantages are their simplicity and interpretability and they can be easily understood and visualized. They are non-parametric and can deal with both categorical and continuous variables or with a mix of a both. They use a so called white-box model easily explainable by boolean logic, which mirrors human decision making more than other algorithms. In addition, they can deal with multiclass classification straight away, so there is no subsequent extension from binary to multiclass classification.

The main disadvantages are that they are prone to overfitting. Small change in the training data can result in a large change in the tree. Learning, or induction, of an globally optimal decision tree, which is yet to be explained in terms of impurity function, is known to be NP-complete even for simple concepts. As a result, the learning is based on heuristics such as greedy search. As a result, local optimal decisions at each tree node do not guarantee globally optimal decision tree. As well as with linear classifiers, the decision trees have problems like learning the XOR function. On the other hand, the mentioned disadvantages are addressed by the ensemble learning.

The by far most common decision tree learning is the top-down induction, which constructs the tree by choosing a locally optimal feature at each each step. The entire construction of a tree revolves around selection of the splits, decisions when to declare a node to be leaf or to continue splitting it and prediction of a leaf [7]. The fundamental idea is to select each split so the data in subsequent nodes are *purer* than the data in the parent node [7]. For that reason, **impurity function** $i(s,t)$ and decrease in impurity $\Delta i(s,t)$ are introduced, in which t represents a node and s a split. Decrease in impurity is defined as

$$\Delta i(s,t) = i(t) - p_L i(t_L) - p_R i(t_R), \tag{2.11}$$

where $p_L = \frac{N_{t_L}}{N_t}$ denotes a proportion of learning samples going to left node t_L, $p_R = \frac{N_{t_R}}{N_t}$ proportion of learning samples going to right node t_R and N_t is the size of learning subset at node t [7]. In a root and internal nodes, the search procedure is to find the optimal split s^* which gives the maximal decrease in impurity, thus $\Delta i(s^*,t) = \max_s \Delta i(s,t)$. This procedure is recursively iterated for the created notes.

There are several ways to choose an impurity function. The most common impurity criteria used for classification trees are the Shannon entropy $i_H(t)$ and the Gini index $i_G(t)$ [24] defined as

$$i_H(t) = -\sum_{k=1}^{\mathcal{C}} p(C_k|t) \log_2 p(C_k|t), \tag{2.12}$$

$$i_G(t) = \sum_{k=1}^{\mathcal{C}} p(C_k|t) \left(1 - p(C_k|t)\right) \tag{2.13}$$

where $p(C_k|t)$ denotes a proportion of class C_k at a node t. Decrease in impurity $\Delta i(s,t)$ with Shannon entropy impurity $i_H(t)$ function yields in information gain known from Shannon information theory [36].

As there are several impurity function options, there are also different tree induction algorithms. The ID3 algorithms, developed by Ross Quinlan in 1986 [31], uses the greedy strategy by selecting the locally best split using the Shannon entropy $i_H(t)$. Its extension C4.5, also developed by Quinlan, builds the tree the same way, using Shannon entropy, but the algorithm also accepts continuous variables. This is done by partitioning the particular continuous feature space into discrete set of intervals. The latest extension is C5.0 algorithms, being faster and more memory efficient. Classification And Regression Tree (CART), developed by Leo Breiman in 1984 [7], uses the Gini impurity function and supports continuous variables. In this thesis, we use a CART algorithm.

Stopping criteria provide a means to deal with hypothetical trade-off between a tree too deep and a tree too shallow. There are several heuristic rules, such as: node does not need to be further split when all the output values are homogeneous, as further dividing the subset would bring no additional value. The general idea to avoid overfitting is when the particular subset becomes too small or no sufficiently good split can be found. There needs to be a minimum number of samples required to split an internal node N_{min}, the depth of leaves needs to be smaller or equal to some d_{max}, the decrease in total impurity needs to be less than threshold β and finally terminal nodes should be no further split if both number of right and left samples are at least N_{leaf}. The foursome $(N_{min}, d_{max}, \beta, N_{leaf})$ form a set of hyperparameters which need to be tuned [24]. As stopping criteria might be sufficient with dealing with overfitting, another technique is to use tree *pruning*, removing tree nodes

or branches insignificant in relation to a predefined error rate. This technique is sometimes called post-pruning, calling stopping criteria pre-pruning.

Ensemble methods use multiple decision trees to obtain better predictive and decision abilities, as decision trees usually have low bias and high variance. There are several ensemble methods used specifically for decision trees. There are bootstrap aggregating and boosting among these methods.

Random forests

Random forest consists of a a multitude, or an ensemble, of decision trees grown from a randomized variant of mentioned decision tree induction algorithm. There are several ways to introduce the randomization into the induction algorithm.

One of the way of introducing the randomization is by training the individual decision trees on resampled sets from the training set. Prediction is made either by averaging over the trees in case of regression or by averaging the scores over the trees (or majority voting) in case of classification. Generally, boostrapping is any statistical method using sampling with replacement. Given a training set, bootstrap aggregating, also called *bagging*, is a meta-algorithm that improves the stability of a machine learning algorithm, which is closely connected to the bias-variance trade-off. Bagging procedure includes generating a new re-sampled set from the training set uniformly and with replacement. This set is then used for training a given decision tree. Although bagging is prevalent with decision trees, another machine learning algorithms can benefit by using it. Another way to introduce randomization is to randomize the variable selection at each node. The best split is similarly found as the split maximizing the decrease in impurity function, but only over a random subsample of the variables.

In 2001, Breiman [6] introduced a combination of bagging and random variable selection, calling the algorithm Random Forests. This algorithm yields one of the most effective results not only among different random forests variants but also among machine learning methods generally, competitive even with boosted decision trees.

Boosted decision trees

Boosting is a machine learning method used for converting many weak learners, decision trees in our case, into a strong learner. **Boosted decision trees** consist of iteratively induced decision trees, weighted based on their previous performance, with prediction being the same as with random forests. The two main boosting algorithms are adaptive boosting (AdaBoost) and gradient boosting.

AdaBoost was introduced by Fround and Shapire in 1995 [12]. The algorithm is called adaptive because it adjusts the data weights for a subsequent weak learner based on the predictions of the previous weak learner. In the reweighing process, the data are reweighted so misclassified samples gain a higher importance and correctly classified data lose importance. Weak learners are often very shallow decision trees or decision trees with no internal nodes, which are also called decision stumps. The prediction is then made by weighted average, with weights based on predefined prediction error rate, of the weak learners. More can be seen in [12].

Gradient boosting is a combination of gradient optimization methods and weighing the weak learners. Gradient boosting algorithms train on the incorrect predictions, residuals or misclassification, using a loss function approach. In first iteration, the algorithm instantiates

a weak learner that predicts a constant minimizing the overall loss between $\boldsymbol{y}$ and a range of constants. In a subsequent phase, another weak learner is trained based on so called pseudo-residuals of a preceding weak learner. These pseudo-residuals are computed as a derivative of a chosen loss function. One-dimensional optimization problem of minimizing the loss between $\boldsymbol{y}$ and a sum of preceding weak learner and new weak learner is then performed. This procedure is applied iteratively to find the final model. Similarly, more can be seen in [13]. There are multiple implementations of gradient boosting algorithms, such as *XGBoost* or *LightGBM*.

Chapter 3

Artificial Neural Networks

Artificial Neural Networks (ANNs) have their original inspiration in biological neural networks, but there are many differences, as many biological complexities are not incorporated by the ANNs. Even many features of the ANNs are known to be inconsistent with biological networks. Despite these differences, ANNs show outstanding results in the domains of machine learning. ANN constitutes a mathematical model, more precisely is defined as a graph $(\mathcal{V}, \mathcal{E})$ of set of vertices $\mathcal{V}$ called *neurons* and set of edges $\mathcal{E}$ called *weights*.

Deep learning algorithms, a subset of machine learning algorithms, are concerned with and based on nested hierarchical concepts, often being layers of ANNs. The popularity of deep learning in recent years is on the rise due to increasing of the computational power, enabling a rapid development in the area.

3.1 Perceptron

The history of ANNs dates back to 1943, when McCulloch and Pitts [27] came with model of brain function which was very similar to linear classifier model. The model was not yet adaptive, because weights were set manually. In 1958, the **perceptron** algorithm was introduced by Rosenblatt [33]. It was originally a machine consisting of photocells, weights encoded in potentiometers and weight update was achieved by electric motors. As contrary to previous models, it was one of the first models that could learn the weights adaptively. This was done by updating the weights when encountering the wrong classification by special case of stochastic gradient descent, which is later explained in section 3.3.

In mathematical formulation, perceptron is a linear classifier, thus can be using the notation from (2.8) written as

$$y = f\left(\sum_{j=0}^{\mathcal{I}} w_j x_j\right) = f\left(\boldsymbol{x}^T \boldsymbol{w}\right), \tag{3.1}$$

with $f(\cdot)$ being originally Heaviside step function $\Theta(\cdot)$. Usually, w_0 is called *bias (threshold)* and $(w_1, \ldots, w_{\mathcal{I}})$ is called set of *weights*. In the terminology of ANNs, the function $f(\cdot)$ is called **activation function**. Activation function is an inverse function of link function from section 2.3. The activation function was later changed from simple $\Theta(\cdot)$ to more complex ones. In case of perceptron, the activation function plays the role of the overall mapping. In comparison with ohter linear models, perceptron with identity activation function $f(x) = x$ corresponds to linear classifier with hyperplane decision surface and perceptron with *sigmoid*

activation function $\sigma(\cdot)$ corresponds to logistic regression etc. Sigmoid activation function is an inverse of logit link function, therefore defined as

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}. \tag{3.2}$$

Another nowadays predominantly used activation function is *rectified linear unit* $\mathrm{ReLU}(\cdot)$ defined as

$$\mathrm{ReLU}(x) = \max(0, x). \tag{3.3}$$

Historically, *hyperbolic tangent* activation function, which can be rewritten as a function of $\sigma(\cdot)$, defined as

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1, \tag{3.4}$$

was also being used, but together with $\sigma(\cdot)$ is not being widely used due to optimization issues explained in 3.3. In implementation of this thesis, $\tanh x$ as an activation function is not used.

Perceptron can be, as a linear classifier, extended to nonlinear model thanks to kernel method, making the algorithm kernel perceptron.

3.2 Multi-layer perceptron

Another way to solve the limitation of linear models (e g. learning the XOR function) is by introducing the **multi-layer perceptron (MLP)**. MLP, often called feedforward neural network, as a machine learning classifier, has its goal in finding the mapping function of inputs to outputs. These models are also called feedforward, because there are no feedback connections or loops in the neural network graph.

These models are called MLPs because the overall mapping is formed by composing many simpler functions between the input and the output. The chain of these functions are called the layers of the ANN. The length of the chain gives the depth of the model, indicating the name deep learning.

The first layer $l = 1$ of the network resembles linear model, using the notation from (2.8), thus having definition $\boldsymbol{z_1} = (z_0^1, \ldots, z_{n_1}^1)$ with elements

$$z_{k_1}^1 = f_1\left(\sum_{j=0}^{\mathcal{I}} w_{j,k_1} x_j\right), \tag{3.5}$$

and index $k_1 \in \{0, \ldots, n_1\}$, n_1 meaning number of *first layer outputs* and being dependent on layer $l = 1$. The only difference between the perceptron and first layer of MLP is the number of outputs. Instead of one, there are now more, with weights connecting all inputs $\boldsymbol{x}$ and all *first layer outputs* $\boldsymbol{z_1}$, first weights index indicating the connection from inputs and second index indicating connection to the neuron in first layer. These *first layer weights* can be rewritten as the *first layer weight matrix* $\boldsymbol{W_1} = (w_{j,k_1})_{j,k_1=0}^{\mathcal{I},n_1}$.

Of course. there are more layers in *multi*-layer perceptron, thus yielding $\boldsymbol{z_l} = \left(z_0^l, \ldots, z_{n_l}^l\right)$ with elements

$$z_{k_l}^l = f_l\left(\sum_{j=0}^{n_{l-1}} w_{j,k_l} z_j^{l-1}\right), \tag{3.6}$$

for $l \in \mathcal{L}$, where $\mathcal{L}$ is number of layers and index $k_l \in \{0, \ldots, n_l\}$, n_l meaning number of *l-th layer outputs* and being dependent on layer l. Similarly, *l-th layer weight matrix* can be written as $\boldsymbol{W_l} = (w_{j,k_l})_{j,k_l=0}^{n_{l-1},n_l}$.

The last layer of the MLP yields the output $\hat{\boldsymbol{y}} = \boldsymbol{z_{\mathcal{L}}} = \left(z_0^{\mathcal{L}}, \ldots, z_{n_{\mathcal{L}}}^{\mathcal{L}}\right)$, with elements defined as

$$y_{k_{\mathcal{L}}} = z_{k_{\mathcal{L}}}^{\mathcal{L}} = f_{\mathcal{L}} \left(\sum_{j=0}^{n_{\mathcal{L}-1}} w_{j,k_{\mathcal{L}}} z_j^{\mathcal{L}-1} \right), \tag{3.7}$$

with $n_{\mathcal{L}} = \mathcal{O}$.

In the matrix notation, the *l-th layer output* can be written as

$$\boldsymbol{z_l} = f_l \left(\boldsymbol{W_l}^T \boldsymbol{z_{l-1}} \right) \tag{3.8}$$

The subsequent activation functions give the overall mapping $f(\cdot)$. In a two-layer perceptron, as can be seen in the figure 3.1, there is one hidden layer and an output layer. There is an alternative definition that includes the so-called *input layer* in the total number of layers, but as there are two activation functions f_1, f_2, it is common to call the model only two-layered. Having the matrix notation from (3.8) in each layer, the two-layer perceptron can be rewritten as

$$\hat{\boldsymbol{y}} = f_2 \left(\boldsymbol{W_2}^T f_1 \left(\boldsymbol{W_1}^T \boldsymbol{x} \right) \right). \tag{3.9}$$

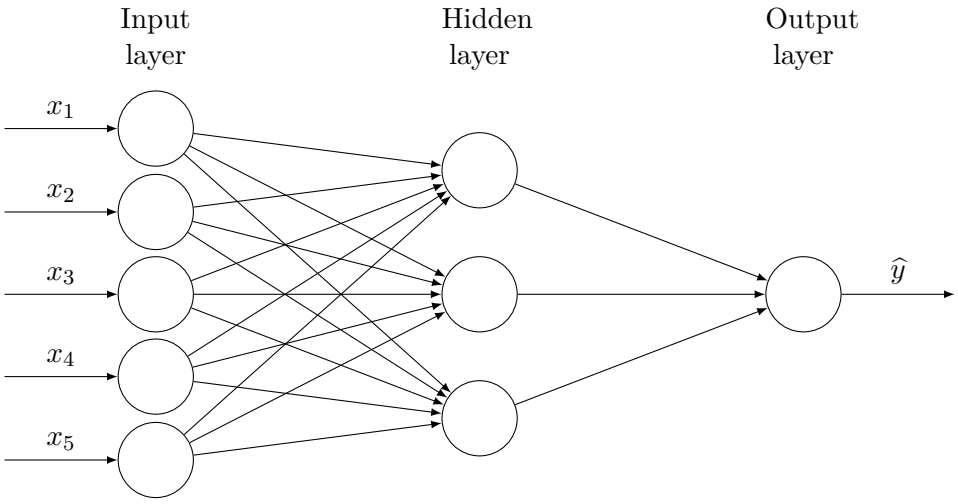


Figure 3.1: A two layer MLP with five inputs and one output.

Simply put, MLP can be perceived as stacking up linear classifiers with different activation functions. Linear activation functions are not being used because stacking up linear functions gives another linear function.

In each layer, same activation functions as were introduced for perceptron, are being used. These include ReLU$(\cdot)$, $\sigma(\cdot)$ and $\tanh \cdot$. The only difference is that these activation function are often multidimensional. In classification setting, it is common to have sigmoid activation function $\sigma(\cdot)$ as the activation function in the last layer of the MLP. The output

$\hat{y}_k \in [0,1], k \in \mathcal{O}$ is **score**, which is looked at as probability of a class given training data, similarly to the decision tree and their ensembles.

For a multidimensional output, the sigmoid activation function is replaced with *softmax* activation function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$, which normalizes the output, defined as

$$\sigma(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \forall i \in \hat{K}. \tag{3.10}$$

Parameters of the whole MLP can be then compactly written as $\boldsymbol{\theta} = (\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{\mathcal{L}})$, in which elements of $\boldsymbol{\theta}$ are random variables. The optimal parameters are obtained by the optimization process, which is explained in subsequent section 3.3.

3.3 Optimization for deep learning

Finding the optimal parameters $\boldsymbol{\theta}$ of ANN mapping f consisting of all the weights and biases of the network can be perceived as a problem of mathematical optimization. The process is known as learning of training the parameters. Generally, analytical solution of optimal $\boldsymbol{\theta}$ can not be found (which is the case even for logistic regression, which uses e. g. the iterative weighted least squares [18]), so numerical nonlinear optimization needs to be used.

In case of ANNs, cross-entropy loss function approach is used. In the community, it is common to call cross-entropy loss function only the cross-entropy between the empirical distribution and the Bernoulli (or multinomial for multiple classes) distribution. By taking the same MLE of Bernoulli distribution as for logistic regression, we end up with *binary cross-entropy* loss function defined as

$$L\left(\hat{y}^{(i)}, y^{(i)}\right) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right). \tag{3.11}$$

Note the similarity between logistic regression and binary-cross entropy loss function. Also, note that $\hat{\boldsymbol{y}}$ is a function of $\boldsymbol{\theta}$, all parameters of the ANN. As a consequence, solving the optimization problem ANN is not so straightforward as for logistic regression.

Cross-entropy for multi-class classification, a logical extension to multiple classes, is called *categorical cross-entropy* loss function and is similarly defined as

$$L\left(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}\right) = -\sum_{k=1}^{\mathcal{O}} y_k^{(i)} \log\left(\hat{y}_k^{(i)}\right) \tag{3.12}$$

with hot-encoded label output.

3.3.1 Backpropagation

With bearing in mind the differences between pure optimization and machine learning optimization, and correspondence of MLE and loss function, as discussed in chapter 2, the overall loss function is to be minimized. This is achieved by gradient-based algorithms. Their functionality is enabled by **backpropagation** algorithm [34], a method of computing the respective gradients [15].

At first, inputs $\boldsymbol{x}$ provide initial information and flow through the MLP as in (3.6). This is called *forward pass*, yielding in $\hat{\boldsymbol{y}}$ and producing $J(\boldsymbol{\theta})$. Computing the analytic expression (not solution) of gradient

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{3.13}$$

is straightforward, using derivative rule of chain rule, always using gradients from the layers closer to the output to compute the gradients farther from the output. This is called *backward pass* and it allows the information from $J(\boldsymbol{\theta})$ to flow backwards to compute gradients. The backward pass starts by computing $\frac{\partial J(\boldsymbol{\theta})}{\partial \hat{\boldsymbol{y}}}$ and then passes the derivative to preceding layer. This way, computed gradient may be repeatedly used for computing gradients in preceding layers. Subsequently, these gradients may be repeatedly used in a similar fashion and so forth.

The analytic expression for derivatives of sigmoid $\sigma(\cdot)$ and of ReLU$(\cdot)$ activation function are easily written as

$$\frac{\mathrm{d}\sigma(x)}{\mathrm{d}x} = \sigma(x)\left(1-\sigma(x)\right), \quad \frac{\mathrm{dReLU}(x)}{\mathrm{d}x} = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \tag{3.14}$$

with the undefined derivative at 0, which does not heuristically cause problems and is set at right-derivative.

The problem with $\sigma(\cdot)$ and $\tanh\cdot$ is the so-called *exploding* and *vanishing* gradients. Sharp nonlinearities in parameter space result in very high derivatives in some places of the space, causing the gradient update to catapult the parameters to a totally different location.

The problem with ReLU$(\cdot)$ is gradient equal to 0, causing *dead* neurons. This issue has been addressed by introducing modifications, e. g. leaky ReLU defined as linear function for $x \geq 0$ and $0.01x$ for $x < 0$, but the modifications are, based on their performance, not being used. Overall, ReLU$(\cdot)$ activation functions are predominantly used in hidden layers and sigmoid activation function is used in the last layer.

3.3.2 Optimization algorithms

Training the ANN is generally a non-convex optimization problem, and a lot of challenges complicate the whole process. These include poor conditioning of hessian matrix defined as $\boldsymbol{H} = \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2}$, many local minima, saddle points and flat plateaus in the parameter space, and computed gradients being vanishing or exploding [15].

The simplest algorithms are first-order derivative optimization methods, among them is the age-old algorithm called gradient (or steepest) descent, with computed gradient pointing directly uphill (in parameter space) and negative gradient pointing directly downhill. Gradients of empirical risk function can be written as

$$\begin{aligned} \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &\equiv \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \\ \frac{\partial \left\{ \frac{1}{N} \sum_{i=1}^{N} L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)} \right) \right\}}{\partial \boldsymbol{\theta}} &\equiv \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)} \right), \end{aligned} \tag{3.15}$$

therefore the gradient of empirical risk function is

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)} \right). \tag{3.16}$$

Because of the many mentioned challenges, gradient descent cannot be easily applied. Instead, it is possible to obtain an unbiased estimate of the gradient by taking the average not on the whole dataset $\mathcal{D}$, but on its subsamples called minibatches $\mathcal{B}$ with *iid* drawn

samples. Each minibatch is defined as *iid* $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(B)}\} = \mathcal{B}$, with B being the *minibatch size*, so $\text{card}(\mathcal{B}) = B$ obviously holds. Usually, the overall count of minibatches passes are called *iterations*, thus defined as N/B; *epoch* is pass of all samples from $\mathcal{D}$.

The algorithms, which use the minibatch approach, are usually called *minibatch*, or simply *stochastic*. Minibatch gradient is less accurate, but much faster, often enabling parallel computation (especially when using GPUs), and also offers regularizing effect, which is to be discussed more in 3.4. Minibatches need to be selected *iid* introducing the stochastic element into the training.

Stochastic gradient descent

The widely used stochastic modification of gradient descent is **stochastic gradient descent (SGD)**, which is a lot important in terms of deep learning and which is gradient descent performed on the minibatches $\mathcal{B}$. The gradient is then computed as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})_{\mathcal{B}} = \frac{1}{B} \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}\right), \tag{3.17}$$

compare to (3.16). During the training, minibatches firstly perform forward pass and secondly they perform backwards pass, the weight update is iterative and is for iteration $k \in \mathbb{N}$ following

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})_{\mathcal{B}}, \tag{3.18}$$

where $\eta_k \in \mathbb{R}^+$ is *learning rate* and $\boldsymbol{\theta}_0$ is initialized beforehand, which is discussed in 3.5. Learning rate is hyperparameter, which should decrease in time to reach global minimum and offers another source of stochasticity. A sufficient condition to guarantee convergence of SGD is that $\sum_{k=1}^{\infty} \eta_k = \infty$, $\sum_{k=1}^{\infty} \eta_k^2 < \infty$ [15].

Momentum and Nesterov momentum

The method of momentum was designed to accelerate learning and avoid oscillations while using the SGD [30]. The momentum algorithm accumulates an exponentially decaying moving average of computed negative minibatch gradients [15].

Formally, it introduces a new variable called *velocity* that plays the same role as mechanical velocity in Lagrangian mechanics. In the momentum algorithm, we assume unit mass, so velocity may be regarded as momentum.

The weight update for momentum SGD is also iterative in two steps following

$$\begin{aligned} \boldsymbol{v}_{k+1} &\leftarrow \alpha \boldsymbol{v}_k - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})_{\mathcal{B}}, \\ \boldsymbol{\theta}_{k+1} &\leftarrow \boldsymbol{\theta}_k + \boldsymbol{v}, \end{aligned} \tag{3.19}$$

where hyperparameter $\alpha \in [0, 1)$ determines the decay of gradients and η denotes learning rate. The algorithm can be generalized to increase in time, while it is not as important as decreasing η in time. For $\alpha = 0$, the momentum SGD becomes SGD with fixed learning rate η.

Another optimization method inspired by Nesterov's accelerated gradient method [29] is **Nesterov momentum**, which has a lot of similarities with classical momentum method. The difference between Nesterov momentum and classical momentum is in the time when

the gradient is computed. In case of Nesterov momentum, the gradient is computed after the velocity is applied, so the weight update is following

$$\begin{aligned} \boldsymbol{v}_{k+1} &\leftarrow \alpha \boldsymbol{v}_k - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} + \alpha \boldsymbol{v})_{\mathcal{B}}, \\ \boldsymbol{\theta}_{k+1} &\leftarrow \boldsymbol{\theta}_k + \boldsymbol{v}, \end{aligned} \tag{3.20}$$

with α denoting the decay of gradients and η denoting learning rate.

Adaptive algorithms

Adaptive algorithms offer a different look which address the issues of setting the proper learning rate other than momentum, which is at the cost of introducing another hyper-parameter to tune. Adaptive algorithms offer adaptation of learning rate throughout the learning without needing to tune it.

There has been ongoing research to modify the momentum SGD, which include namely e.g. **AdaGrad**, **RMSProp** or **Adam**. AdaGrad individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values. RMSProp modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average. Adam is yet another variant on the combination of RMSProp and momentum. More with details on exact algorithms of adaptive AdaGrad, RMSProp, Adam and others can be found in [15].

Second order derivative methods

Instead of using only first derivative of empirical loss function, one can use higher derivatives, which is unfortunately at cost of computation time. Second order optimization methods use the hessian matrix $\boldsymbol{H}$ defined as $\boldsymbol{H} = \frac{\partial^2 J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2}$. These algorithms include the simplest method, which is called the **Newton's method**. Newton's method update based on second order of Taylor expansion, and is

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)_{\mathcal{B}}. \tag{3.21}$$

To deal with saddle points, hessian matrix can be regularized, thus adding $\alpha \boldsymbol{I}$ to $\boldsymbol{H}$, which is known as **Levenberg-Marquart** method [23].

The problem with second order derivative methods is computation speed. As large training sets are necessary, they are computationally expensive. Additionally, these methods usually need much higher batch size due to better approximation of $\boldsymbol{H}$. As a consequence for for large datasets $\mathcal{D}$, they are not usually preferred due to high computation time.

The choice of optimization algorithm is dependent on the classification problem and there is no best algorithm to use. The choice is often up to the user and users' familiarity with specific algorithms.

3.3.3 Parameter initialization

Deep learning optimization algorithms commonly depend on the parameter initialization. The whole optimization process is still not fully understood, so the initialization strategies are mostly heuristic. There are some exceptions to the understanding, as it is for example clear that two hidden units with same activation functions, which are connected to the same

inputs, should be initialized differently. Typically, biases are initialized heuristically to a chosen constant and weights are initialized randomly. The random distribution for weights is most commonly Gaussian or uniform. Specific initialization methods are also chosen to tackle the vanishing or exploding gradient problem. This leads to the assumption that the variance of the outputs of each layer needs to be equal to the variance of its inputs, and also the gradients should have equal variance before and after flowing through a layer during backpropagation.

For uniform distribution, this has led to initialize the layer l weights' with $\mathcal{I}_l$ inputs and $\mathcal{O}_l$ outputs as

$$\boldsymbol{W}_l \sim U\left(-\sqrt{\frac{6}{\mathcal{I}_l + \mathcal{O}_l}}, \sqrt{\frac{6}{\mathcal{I}_l + \mathcal{O}_l}}\right). \tag{3.22}$$

This initialization is called Glorot [14], but sometimes also Xavier uniform initialization.

For Gaussian distribution, the Glorot (also Xavier) initialization abides

$$\boldsymbol{W}_l \sim \mathcal{N}\left(0, \frac{2}{\mathcal{I}_l + \mathcal{O}_l}\right). \tag{3.23}$$

He [16] normal initialization takes a slightly different approach than Glorot does, finally yielding

$$\boldsymbol{W}_l \sim \mathcal{N}\left(0, \frac{2}{\mathcal{I}_l}\right). \tag{3.24}$$

3.4 Regularization for deep learning

As discussed in 2.1, the classifier needs to perform well not only on the training set, but also on previously unseen samples, which is simulated by the test set. Regularization techniques are designed to reduce error on the test set.

A lot of different strategies exist, often putting some extra constraints on the model itself or on the model parameters. This includes incorporating some prior knowledge either about the problem or about the model; or the preference for a simpler model, which should generalize well, as a final model (compare to Occam's razor). These constraints are believed to lead to improving the performance measure P of a given classifier on the test set.

3.4.1 Loss function regularization

While parameter $\boldsymbol{\theta}$ includes all the weights and biases, it is common to include only the *weights* $\boldsymbol{w}$ in the regularization term. Typically, biases require less data to fit accurately. Empirically, this is due to the fact that by leaving biases unregularized higher bias variance is not induced. Additionally, regularizing the biases, significant amount of underfitting is introduced [15].

While it might be reasonable to have different regularization in each layer, it is expensive to search for a multitude of hyperparameters. This often leads to having the same regularization at all layers for the sake of simplicity [15].

In context of deep learning, adding L_2 norm to the loss function is called **weight decay** expressed as

$$\widetilde{J(\boldsymbol{w})} = J(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2. \tag{3.25}$$

This regularization term drives the weights *closer* to the 0. MLE with regularization term equal to L_2 norm of parameters can be perceived as MAP estimation with Gaussian prior, a maximum of posterior distribution in Bayesian approach.

Adding L_1 norm results in solution that is more *sparse* in $\boldsymbol{w}$, which yields

$$\widetilde{J(\boldsymbol{w})} = J(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1. \tag{3.26}$$

Compare to LASSO, ridge regression and elastic net regularization in 2.3.

3.4.2 Other approaches to regularization

While loss function regularization is the most prevalent regularization technique to use, there are also other regularization techniques. These can and are used in unison with loss function regularization.

The best way for ANN to generalize well is to have large dataset $\mathcal{D}$. One way is to obtain larger dataset in a natural way, which is due to various reasons not always feasible. However, larger dataset can be obtained by artificially enhancing the dataset itself. This is done by a technique called **dataset augmentation**. Dataset augmentation basically create new fake data, which resemble the original samples. A new pair of generated input $\boldsymbol{x}$ and output $\boldsymbol{y}$ exploit the fact that ANN should be invariant to a specific kind of transformation. As an example, image data, as is our case, can be translated, rotated and scaled. There is a warning though, one should be always very careful to the input transformation, as misleading fake data can be introduced. Rotating image data, in which direction is important feature, is not correct. Another way to enhance the dataset to inject artificial noise to the input $\boldsymbol{x}$. Even another approach is to use deep learning itself to generate fake data. Past years, it has been proven successful to use adversial ANNs or variational autoencoders, which mimic the distribution of the original dataset to create new fake data [15].

When training a ANN over a large multitude of epochs, the phenomenon of training error declining and validation error firstly also declining and afterwards rising again is well studied. This can be exploited by a heuristic called **early stopping**. With usage of early stopping, the algorithm terminates training when no parameters have improved over the initially preset number of iterations. This accelerates the whole training process, but also acts as a regularizer.

Another very powerful technique is **dropout** [39]. Dropout can be seen as a process of constructing new inputs by multiplying by noise. Dropout offers a computationally inexpensive alternative of regularization by randomly selecting the input and hidden neurons, which are excluded during training. This is done by multiplying the corresponding layer output by 0. Whenever a new minibatch $\mathcal{B}$ is used, a binary mask for each neuron in all input and hidden layers is sampled from Bernoulli (or alternative) distribution $A(p)$. The activation is then multiplied with the binary mask, yielding in including and excluding some of the eurons. Hyperparameter $p \in (0, 1]$ needs to be set. Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5 [15]. During testing phase, no neurons are excluded. Dropout can be perceived as bootstrap aggregating (bagging) a multitude of neural networks, which is very used and successful method for decision trees, without needing to train and test all of them, although with dependent weak learners.

3.5 Optimization and regularization aspects

3.5.1 Batch normalization

Batch normalization is an elegant method of model reparametrization, which introduces both additive and multiplicative noise on the hidden units at training time [19]. The goal of batch normalization is to improve optimization process, although the added noise also improves the generalization, thus has regularization effect. Sometimes, as it will be seen in the section 3.6, batch normalization can make dropout unnecessary, so it is used instead. The motivation comes from normalizing the inputs $\boldsymbol{x}$ as in a common classifier, but in case of batch normalization the normalization is also applied to hidden layers.

A general deep ANN is composed of many layers and many parameters $\boldsymbol{\theta}$, which are updated simultaneously under the assumption that the other parameters do not change, which is not the case. This is addressed by batch normalization.

Batch normalization normalizes the layer activations, so the transformed activations have zero mean and standard deviation equal to one. Let $\boldsymbol{A}$ be a minibatch of concrete layer activations arranged as matrix, with rows corresponding to minibatch sample and column to activation. Normalizing the $\boldsymbol{A}$ yields $\boldsymbol{A'}$, thus $\boldsymbol{A'} = \frac{\boldsymbol{A}-\boldsymbol{\mu}}{\boldsymbol{\sigma}}$, where $\boldsymbol{\mu}$ is a vector containing the neuron mean and $\boldsymbol{\sigma}$ is a vector containing the neuron standard deviation.

3.6 Convolutional neural networks

Convolutional Neural Networks (CNNs) are special kinds of ANNs, which build upon all the aspects mentioned in this chapter. They were presented by LeCun [22] as a specialized kind of ANNs that are specially fit to process visual image or time-series data.

Past couple of years, CNNs have been extraordinary successful in the practical application, particularly in the domain of computer vision. Both ANNs and CNNs have been criticized for their inability to interpret results sufficiently and their black-box behavior. Despite the criticism, the continual work on the theory of CNNs continues and new and new knowledge is being gradually found.

The name *convolutional* comes from the operation of *convolution*, widely used method in the domain of image analysis. As [15] states, CNN is an ANN that uses convolution layer instead of general matrix multiplication in at least one of its layers.

In case of MLP, the network takes no spatial information into consideration, but CNN exploits this kind of property. It exploits the fact that nearby pixels are more strongly correlated than the further ones. It does so by extracting only local features that are dependent on small sub-regions of the image [5]. This is achieved by **convolution layers**. Subsequently, by subsampling the image, which is done by a **pooling layers**, different features can be extracted. By applying multitude of both convolution and pooling layers, a set of different types of image features can be extracted. This procedure leads to extracting the multitude of image features. This process is sometimes called *feature extraction* and convolution and pooling layers are responsible for extracting the appropriate features. Subsequently, the selected features are so-called *flattened*, which means they are rearranged to a long vector $\boldsymbol{x}$. This vector acts as an MLP input $\boldsymbol{x}$, called **fully connected layers**, following the processes explained in chapter 3, and resulting in the image *classification*.

3.6.1 Convolution

The convolution operation is a widely used operation in image analysis. Two-dimensional convolution for image input I is defined as

$$(I * K)(x, y) = \int_{\mathbb{R}} \int_{\mathbb{R}} I(w, z) K(x - w, y - z) \, \mathrm{d}w \, \mathrm{d}z \tag{3.27}$$

where K is called convolutional **mask**, or often called convolutional **kernel** in the deep learning domain.

The convolution operation is commutative, yielding

$$(I * K)(x, y) = (K * I)(x, y) = \int_{\mathbb{R}} \int_{\mathbb{R}} I(x - w, y - z) K(w, z) \, \mathrm{d}w \, \mathrm{d}z, \tag{3.28}$$

which arises from the fact that the kernel K is flipped relatively to the input. While the commutative property of convolution is theoretically important, flipping the kernel during training adds a bit of computational complexity. Instead, in the CNN implementations, the cross-corelation is used. The only difference is that the kernel K is not flipped, so cross-correlation is defined as

$$(I \circledast K)(x, y) = \int_{\mathbb{R}} \int_{\mathbb{R}} I(x + w, y + z) K(w, z) \, \mathrm{d}w \, \mathrm{d}z \tag{3.29}$$

The outcome of convolution and cross-correlation is either way the same, because in the former case the weight kernel K would be only flipped as compared to the latter case.

Of course, image data is discretized, so for centered $K \in \mathbb{R}^{2k+1,2l+1}$ the cross-correlation takes the form of

$$(I \circledast K)(i, j) = \sum_{u=-k}^{k} \sum_{u=-l}^{l} I(i + u, j + v) K(u, v). \tag{3.30}$$

Image I can be either original input image or already preprocessed image in one of the hidden layers. For the following purpose, let the shapes of image I be $n_h \times n_w$ and the shapes of kernel K be $k_h \times n_h$. The output image $H = I \circledast K$ would have shapes $n_h - k_h + 1 \times n_w - k_w + 1$.

Convolution, as used in image analysis, can serve as a way of edge detection, sharpening or gaussian blurring etc. The difference between image analysis and CNN is that in the CNN application the kernel K plays the role of trainable parameter $\boldsymbol{\theta}$.

Each input pixel from I is not connected to every output pixel from H, only the so-called *local receptive field*, which size is given by kernel size, is. Convolution also enables parameter sharing, as all the pixels of I share the same kernel matrix K, which significantly reduces the memory footprint of the model. This means that the same feature is computed over different locations in the input, so the point of interest in I can be translated, resulting in the same image feature found. The same kernel K is used repeatedly, which also enables faster computation, because multiplication of image and kernel values in convolution operations often overlap.

Of course, there are usually more than one kernels applied to the image I or hidden layer image I, sometimes even tens. For correct application of kernel matrices, they need to be initialized differently, as explained in chapter 3. Convolution can also be rewritten in terms of sparse matrix multiplication.

Another important aspects of convolution, or cross-correlation, are **padding** and **stride**. Convolution lowers the image dimension and suffers from boundary effects, as the kernel in the boundary area can be applied in several ways. Padding is used to prevent the dimensionality reduction and to deal with boundary effects. Most commonly, zero padding, which adds additional zeros around the image I, is used. Zero padding with p_h rows and p_w columns results in H having shapes $n_h - k_h + p_h + 1 \times n_w - k_w + p_w + 1$. This way, the convolution input and output image dimensionality can be preserved. Additionally, kernel K can not only shift by one pixel per one computation, which is understood as a stride of 1, but also by more pixels with higher stride. The output image H with vertical stride s_h and horizontal stride s_w would have shapes $\frac{n_h - k_h + p_h + s_h}{s_h} \times \frac{n_w - k_w + p_w + s_w}{s_w}$.

Convolution layer is understood as application of convolution operations with the count of kernels used, kernels' size, respective padding and stride. After applying the convolution layer, the output is activated through an activation function, most commonly $\mathrm{ReLU}(\cdot)$, to introduce non-linearity.

3.6.2 Pooling

Pooling operation was introduced both to reduce the dimensionality, also called down-sampling, of the image I and to help make the images approximately invariant to small translations. This means that if the image is slightly translated, the output of a pooling operation would not change. Pooling enables small translations in the original images, which can be caused by lighting or object positioning. Pooling can also even deal with scaling and slight rotation.

The most common technique for pooling is so-called *max-pooling*, which takes the maximum value in the pooling window A, which is applied to slide over image I the same way the convolution kernel K slides over I. The size of the pooling window A needs to be set beforehand. The output of a pooling operation is following

$$z = \max_{a \in A} a, \tag{3.31}$$

where A is the pooling window. With moving of the window A across the image I, the output of a pooling layer is an image of a lower dimensionality, given by the pooling window size.

Another technique for pooling is *average pooling*. As opposed to the max-pooling, instead of taking the maximum value in the pooling window A, average of all elements of A is taken. Average pooling smooths out the image so sharp features may not be sufficiently identified. As a result, average pooling is not widely used as it used to be a few years ago.

Regarding padding and strides, the same as was mentioned for convolution can be applied for pooling. **Pooling** layer consists of pooling type, window size, padding and strides. It is important to notice that there are no learnable parameters $\boldsymbol{\theta}$ in the pooling layer.

3.6.3 Network architectures

Even though the CNNs were introduced by the end of 1990s [22], they gained the widespread popularity in the last decade. It was the ImageNet object recognition challenge [35], which showed the enormous potential of using CNNs. Initially, the winning models of the challenges were based on feature engineering, but in 2012 the CNNs took over. There are a lot of architectures named after the ImageNet challenge.

Massive breakthrough in the usage of CNNs models started when AlexNet [20] architecture won the challenge in 2012 with much better results than previously. After 2012, winners of the ImageNet challenge always used a specific architecture of CNN, always perfecting the overall performance every year.

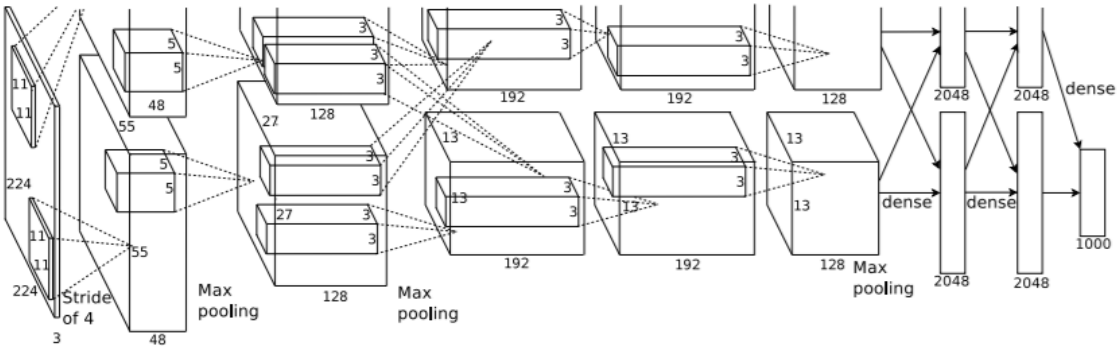
In 2015, a new type of CNN won the ImageNet challenge, which is called residual neural network [17], which skips in between the layers. Squeeze-and-Excitation residual neural network followed in 2017.

New and new architectures are being presented every month and year, as different architectures are fit for different kinds of problems. For results regarding the ImageNet challenges, see [35]. This thesis is concerned with AlexNet, residual neural network and Squeeze-and-Excitation residual neural network.

AlexNet

The architecture, which enabled the breakthrough in the area of CNN, is called **AlexNet** after one of authors of paper [20], Alex Krizhevsky. The network has been proposed to be very similar to the original [22], with the difference in it was much larger and had a lot of more trainable parameters. The architecture is fairly simple, it is constituted of five convolution layers, three overlapping max-pooling layers and three fully connected layers. As an example, the first convolution layer has 96 kernels of 11×11 size with stride 4 without padding.

Figure 3.2: AlexNet architecture [20].



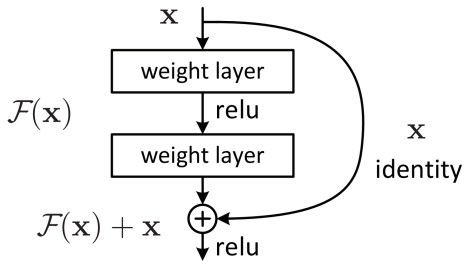
The breakthrough was made possible by using GPUs, especially by dividing the whole architecture into two, as can be seen in 3.2. The whole training process with 90 epochs over 1.2 million images in the ImageNet challenge took five to six days on two NVIDIA GTX 580 3GB GPUs [20].

Residual neural network

Residual neural network (ResNet) [17] introduced the notion of *deep residual learning*, which modifies the learning process. Instead of learning the mapping $\mathcal{H}(\boldsymbol{x})$ of layer input to a layer output by altering $\boldsymbol{\theta}$, another mapping $\mathcal{F}(\boldsymbol{x}) = \mathcal{H}(\boldsymbol{x}) - \boldsymbol{x}$ is learned instead. The original mapping is recast to $\mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$. The ResNets work under the assumption that residual mapping is easier to optimize than an original one, claiming that in extreme, it is easier to learn the residual to be zero rather than to learn the identity.

The formulation of $\mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$ is accomplished by neural network with *shortcut connections*, which can be seen in the figure 3.3. This is a special case of shortcut, as there can be much more complicated layers within mapping $\mathcal{F}(\boldsymbol{x})$, such as multitude of convolution and pooling layers or batch normalization. The whole process of forward and backward pass remains the same.

Figure 3.3: ResNet block within the architecture [17].



In case of ResNet, batch normalization is used instead of dropout. The whole architecture is composed of multitude of residual blocks, batch normalizations, activations and final fully connected layers. There are several common variants of ResNet, most commonly ResNet-18, ResNet-34 and ResNet-50, where the number indicates the number of convolution layers.

Chapter 4

Field Crop Classification

The field crop classification is a challenging problem, which has been tackled by a multitude of various approaches. None of the approaches have proved to be ultimately the best, which is even complicated by numerous image pre-processing techniques and algorithms selection.

This chapter is divided into a brief overview of approaches towards the field crop classification problem and evaluation of our experiments, which builds upon the knowledge described in previous chapters.

4.1 Overview of approaches

The field crop classification experiment can usually be conducted in two settings: the first being the pixel-wise classification, the second being the object-based (field-based in our case) classification. Pixel-wise approach disregards the variability within an object and takes a single pixel as a representative of a given class. On the other hand, object-based approach allows to look at the full image information, with some algorithms even taking image properties like size, shape or texture into consideration. The performance of these approaches generally differ, with some sources claiming object-based classification tends to be more accurate [3] and some sources claiming the same for pixel-wise classification [21]. It usually depends considerably on the dataset size.

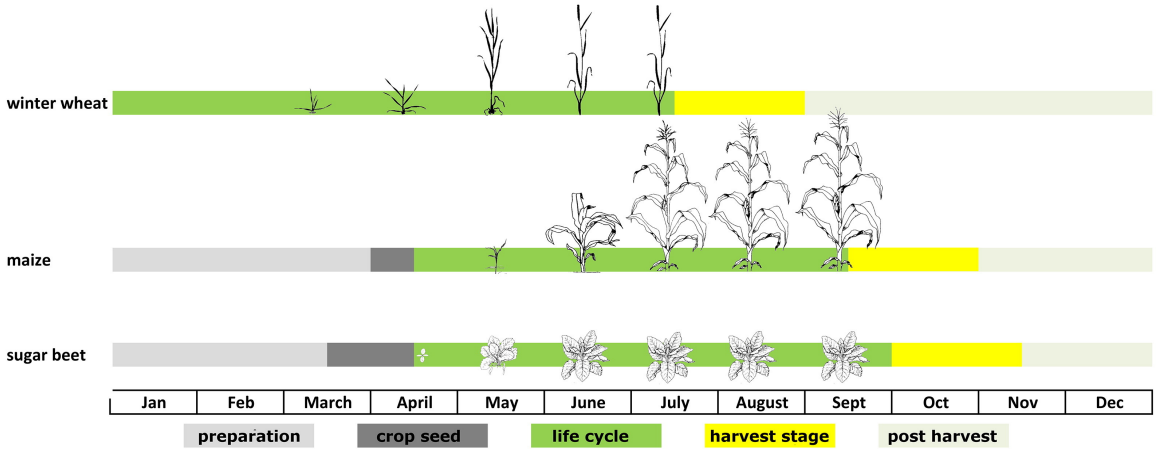
In case of object-based classification, various image features can be computed. These include features like mean and standard deviation, which is the case for e. g. sen4cap toolbox by ESA [9]; or specifically band (or index NDVI specifically [3]) time series etc. In addition to the mentioned features, visual features can be extracted [1] by various CNNs. With respect to the computed features, various algorithm are being used for final classification. To give a more concrete example, random forest is a commonly used classifier [3, 21, 9].

An important role in the classification process plays the crop phenological phase. Each crop has its own phenological phases which naturally repeat each year. These include procedures like sowing, plant growth or harvesting. This means that two given crops can be indistinguishable in one period of the year (e. g. month), but perfectly distinguishable in another period of the year [2]. A figure 4.1 shows phenological phases of three crops, concretely winter wheat, maize and sugar beet. It can be seen that winter wheat is distinguishable from the other two in winter months, on the other hand maize and sugar beet have very similar phenological phases.

The crop phenological phases explain the results differences between so called monotemporal classification, which uses imagery only in one concrete time, and so called multi-

temporal classification, which uses imagery, e. g. throughout the whole year. Naturally, the performance of multi-temporal classification is better than mono-temporal. This has been proven by e. g. [21].

Figure 4.1: Phenological phases of winter wheat (a subset of winter cereals), maize and sugar beet [2].



Belgiu et al. [3] compares pixel-based approach and object-based approach, by using two classification algorithms. These include time-weighted dynamic time warping (TWDTW) method and random forest. It uses Sentinel-2 Level-2A (obtained by sen2cor algorithm) NDVI time series in three different test areas (TA), namely parts of Romania (TA1), Italy (TA2) and California in the USA (TA3). The tile cloudiness level was set to be smaller than 10 %, yielding in 13 images for TA1, 13 for TA2 and 21 for TA3 throughout the year 2016. The crops are different in each test areas, with e. g. *Wheat*, *Maize*, *Rice*, *Sunflower* and *Forest* in TA1. The test accuracy for both pixel-based and object-based approach is higher for random forest in TA1 and TA3, ranging from 88.28 to 97.62. The test accuracy for TA2 was best for object-based TWDTW, concretely 89.75. Overall, TWDTW and random forest achieved comparable results.

Bargiel [2] classifies ten crops based on dense time series of Sentinel-1 data. The area of interest lay in northern Germany and the selected crops were *Grasslands*, *Potatoes*, *Maize*, *Canola*, *Sugar beets*, *Summer barley*, *Winter barley*, *Rye* and *Winter wheat*. It compares maximum likelihood classifier, random forest and phenological sequential patterns, which incorporates phenological information into the classification process, based on a given classifier. It shows similar results for the classifiers, with phenological sequential patters clearly working better on cereal crops.

Račič et al. [32] compares the gradient boosting algorithm, *LightGBM*, with two CNNs, namely TempCNN architecture and temporal convolutional network type of architecture. It uses the Sentinel-2 data in Slovenia in 2017 and provided agricultural ground truth data. More than 200 crops were reduced to thirteen crops, namely *Meadows*, *Grassland*, *Winter rape*, *Maize*, *Winter cereals*, *Leafy legumes and/or grass mixture*, *Pumpkins*, *Summer cereals*, *Vegetables*, *Potatoes*, *Vineyards*, *Soybeans* and *Orchards*. The performance metric used by the authors, weighted average, was 87 % for both CNNs and was significantly higher than *LightGBM's*.

Mazzia et al. [26] introduces a novel deep learning model using a combination of recur-

rent and convolutional neural networks, which uses multi-temporal Sentinel-2 imagery in northern Italy. The dataset consisted of ten images from years 2015 and 2016 and focused on fifteen classes, namely *Tomatoes*, *Artificials*, *Trees*, *Rye*, *Wheat*, *Soya*, *Apple*, *Peer*, *Temp Grass*, *Water*, *Lucerne*, *Drum Wheat*, *Vineyard*, *Barley* and *Maize*. This algorithm achieved the test accuracy of 96.50.

When one tries to compare the results from different sources, one has to bear in mind that different crops, and thus different number of crops, different geographical areas and different dataset size are commonly studied, resulting often into incomparable results.

In this thesis, the focus is on multi-temporal object-based classification with both computed object features and visual object features. As pixel-wise classification performed worse measured by the classification metrics, these experiments are not included in this thesis. All the computations were implemented in the programming language Python 3.6.8 within the scikit-learn 0.23.2 and TensorFlow 2.3.0, with ANNs deployed on the GPUs.

4.2 Dataset description

The dataset was created by combining Sentinel-2 satellite imagery and SAIF agricultural data. The region of interest is Stredni Cechy region within the Sentinel-2 tile encoded as T33UVR. This way, approximately 80,000 agricultural fields were prepared.

Sentinel-2 Level-2A top-of-atmosphere reflectance (computed by sen2cor algorithm) in thirteen times within the year 2019 were selected, concretely eight Sentinel-2A and five Sentinel-2B tiles. The exact dates can be seen in the table 4.1. These tiles were selected based on the tile cloudiness, which is computed as a sum of clouds with medium probability percentage and clouds with high probability percentage from the Level-2A scene classification mask. The selected tiles had cloudiness smaller than 20 %. This selection of months also takes the phenological phases into consideration and gives the emphasis on the time period between April and September. For each date, ten Sentinel-2 spectral bands in 20 m resolution were prepared. These are three bands from 10 m resolution: B2, B3 and B4 (which are available in 20 m resolution in Level-2A); and six bands from 20 m resolution: B5, B6, B7, B8a, B11 and B12. In addition to these nine bands, NDVI was computed.

Based on the bounding box and geographical coordinates of each field from the agricultural data, the field was firstly cropped from the tiles and secondly the surrounding pixels which did not belong to the field were excluded. In addition to the overall tile cloudiness filtering, pixels in each field were filtered based on Level-2A cloud percentage mask, only keeping the pixels with cloud percentage probability smaller than 5 %.

A notable problem of the multi-temporal approach is the cloudiness of the tiles and fields. This means that there is a limited number of tiles and fields without clouds and varying time frequency. One extreme approach filters all the clouded tiles, which yields in all fields without clouds. On one hand, this approach keeps only a limited number of original fields and disregards the importance of time frequency. On the other hand, sufficient number of original fields would ensure sufficient number of unclouded fields. The other extreme approach is to keep all the clouded tiles and fields, which keeps the same time frequency but the multitude of clouded fields would inevitably confuse the classifier. It is important to note that by allowing clouded tiles and fields, a field image might be unclouded in one time and clouded in other. As the number of original fields is not excessively high and as we would like to utilize as much fields as possible, we choose a combination of the extremes. As mentioned, we keep tiles with at most 20 % estimated cloudiness and field pixels with

at most 5 % estimated cloudiness. Choosing thirteen times also ensures sufficient time frequency, at least to some degree. By choosing the proposed cloudiness levels, we expect the classifier to deal with missing pixel values.

Table 4.1: Dataset Sentinel-2A (2A) and Sentinel-2B (2B) tile dates within the year 2019.

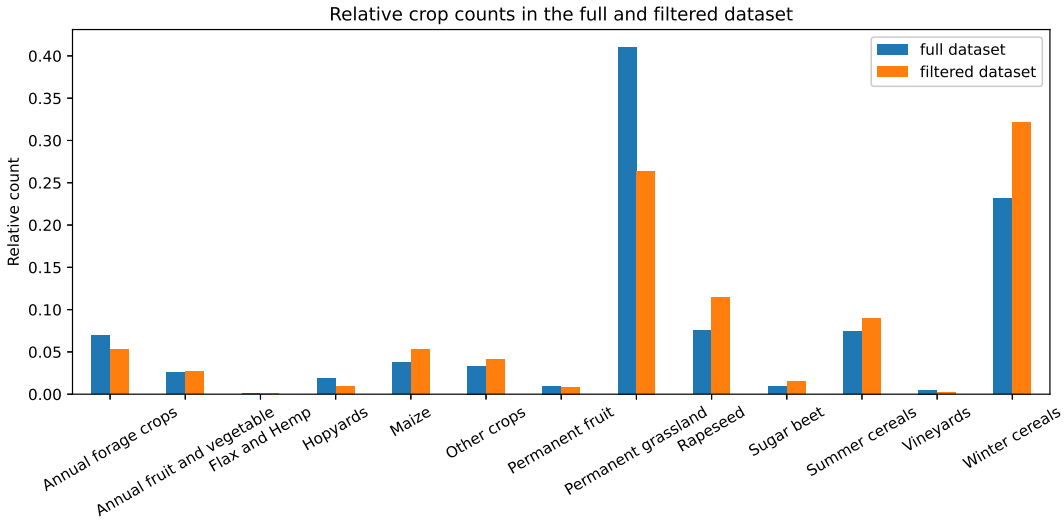
2B	2B	2B	2A	2A	2A	2B	2B	2A	2A	2A	2A	2A
2/18	2/28	4/19	4/24	6/3	6/13	6/18	6/28	7/23	9/1	9/21	10/31	12/20

This procedure yields in multi-temporal multi-spectral dataset $\mathcal{D}$ of images of approximately 80,000 agricultural fields. Being aware of the fact that some labels might be incorrect, we rather work with a larger dataset. Having the images prepared, they need to be pre-processed. First layers of pre-processing are already included in the Level-2A algorithm. Another pre-processing steps were mathematical morphology, image scaling and classifier-dependent pre-processing. Mathematical morphology (erosion concretely) experiments, which excluded even the field boundary, were conducted, but this did not bring any performance improvements. Various kinds of image scaling, such as both band-wise and image-wise min-max scaling, L_2 scaling and normalization, also did not bring any performance improvements.

For classification, we excluded the *not classified* class, the overall dataset crop counts can be seen in figure 4.2, which can be compared to figure 1.3. It should be stated that we have no preference in either crop. As can be seen in the figure 4.2, both datasets are quite imbalanced, which are dealt with by the classifiers differently. In addition to the mentioned, experiments with a dataset including only certain selected crops were conducted. These experiments are not included in this thesis.

All of the experiments were conducted in two settings, with the full dataset $\mathcal{D}_1$ and a dataset $\mathcal{D}_2$ containing only field images' size greater than 10×10 pixels, which roughly corresponds to 200×200 meters. In subsequent text, these are referred to as full dataset and filtered dataset respectively.

Figure 4.2: Relative crop counts in datasets $\mathcal{D}_1$ and $\mathcal{D}_2$.



For correct statistical evaluation, the datasets were randomly divided into training, validation and testing sets in 64:16:20 ratio. This yields 74,243 images (24,278,165 pixels), concretely 47,515 training images, 11,879 validation images and 14,849 testing images in the dataset $\mathcal{D}_1$. For the dataset $\mathcal{D}_2$, 36,129 images, concretely 23,122 training images, 5,781 validation images and 7,226 testing images are obtained. In case of k-fold cross-validation, the datasets were divided into the same training and testing set in 80:20 ratio and training set was further divided k times into separate validation sets.

4.3 Feature-based classifiers

The selected feature-based classifiers are random forests in subsection 4.3.1 and XGBoost implementation of gradient boosting in subsection 4.3.2.

The selected features are sample mean and sample standard deviation for each time and each band. Experiments including other m-th sample central moments with $m > 2$ were conducted, which did not bring any performance improvements. In addition to the mentioned features, field size is also included as a feature, resulting in 301 features for each field. In subsequent subsections, confusion matrices, test metrics are presented, with ROC and PR curves presented for the best-performing classifier.

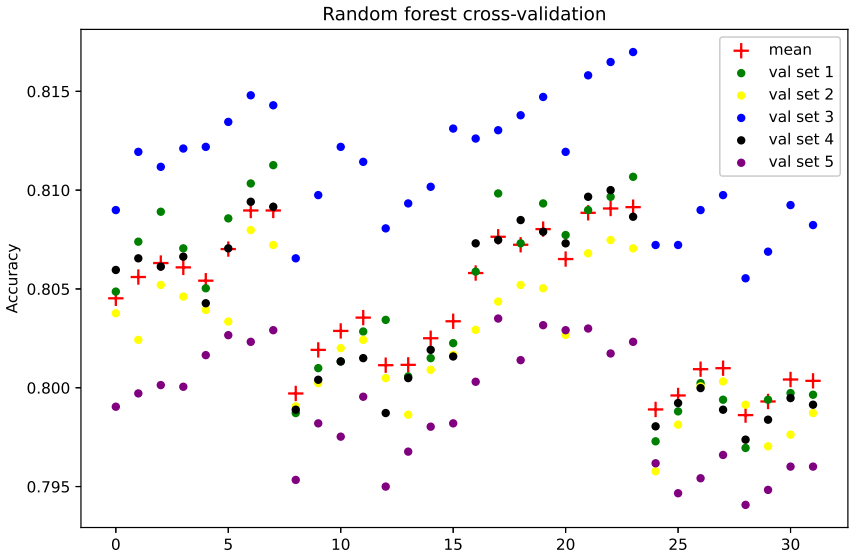
4.3.1 Random forests

In order to deal with the imbalanced dataset, each field entry is weighted by the class proportion in the dataset. This procedure gives more emphasis on the undersampled classes and less emphasis on the oversampled classes.

Full dataset

For optimal hyperparameter selection, 5-fold cross-validation experiment with selected hyperparameters was conducted. Figure 4.3 shows the results for 32 combinations of hyperparameters.

Figure 4.3: Random forest 5-fold cross-validation metrics.



The selected hyperparameter in the figure 4.3 are

- maximal depth d_{max}: unlimited (indices 1-16), 20 (17-32),
- number of trees in random forest: 100, 200, 1000, (repeating indices),
- samples in leafs N_{leaf}: 1 (1-8, 17-24), 4 (9-16, 25-32),
- samples in split N_{min}: 2 (1-4, 9-12, 17-20, 25-28, 31-32), 4 (5-8, 13-16, 21-24, 29-32).

with the the optimal selected hyperparameters $d_{max} = 20, N_{leaf} = 1, N_{min} = 4$ and number of trees equal to 1000.

The figure 4.4 shows the confusion matrix and test metrics, including accuracy, precisions, recalls and F_1 scores for each class, for the random forest with found optimal hyperparameters.

Figure 4.4: Random forest confusion matrix and test metrics.

The overall accuracy is **0.8161** and weighted F_1 score is **0.7987**. It can be seen that some classes, such as *Hopyards*, *Permanent grassland*, *Rapeseed*, *Sugar beet* and *Winter*

cereals are more separable than other classes, with each F_1 score higher than 0.80 and also each precision and recall (except *Sugar beet*) also higher than 0.80.

On the other hand, classes such as *Annual forage crops*, *Flax and Hemp*, *Other crops* and *Permanent fruit* have F_1 score smaller than 0.5. It can be seen that the classifier has problems separating between *Permanent grassland* and other classes, such as *Annual forage crops*, *Other Crops* or *Winter cereals*. This can be explainable by the visual similarity between the crops. As e. g. *Flax and Hemp* class has only thirteen entries in the test set, both the precision and recall are zero.

Filtered dataset

The figure 4.5 shows the confusion matrix and test metrics, including accuracy, precisions, recalls and F_1 scores for each class, for the random forest with found hyperparameters applied on filtered dataset.

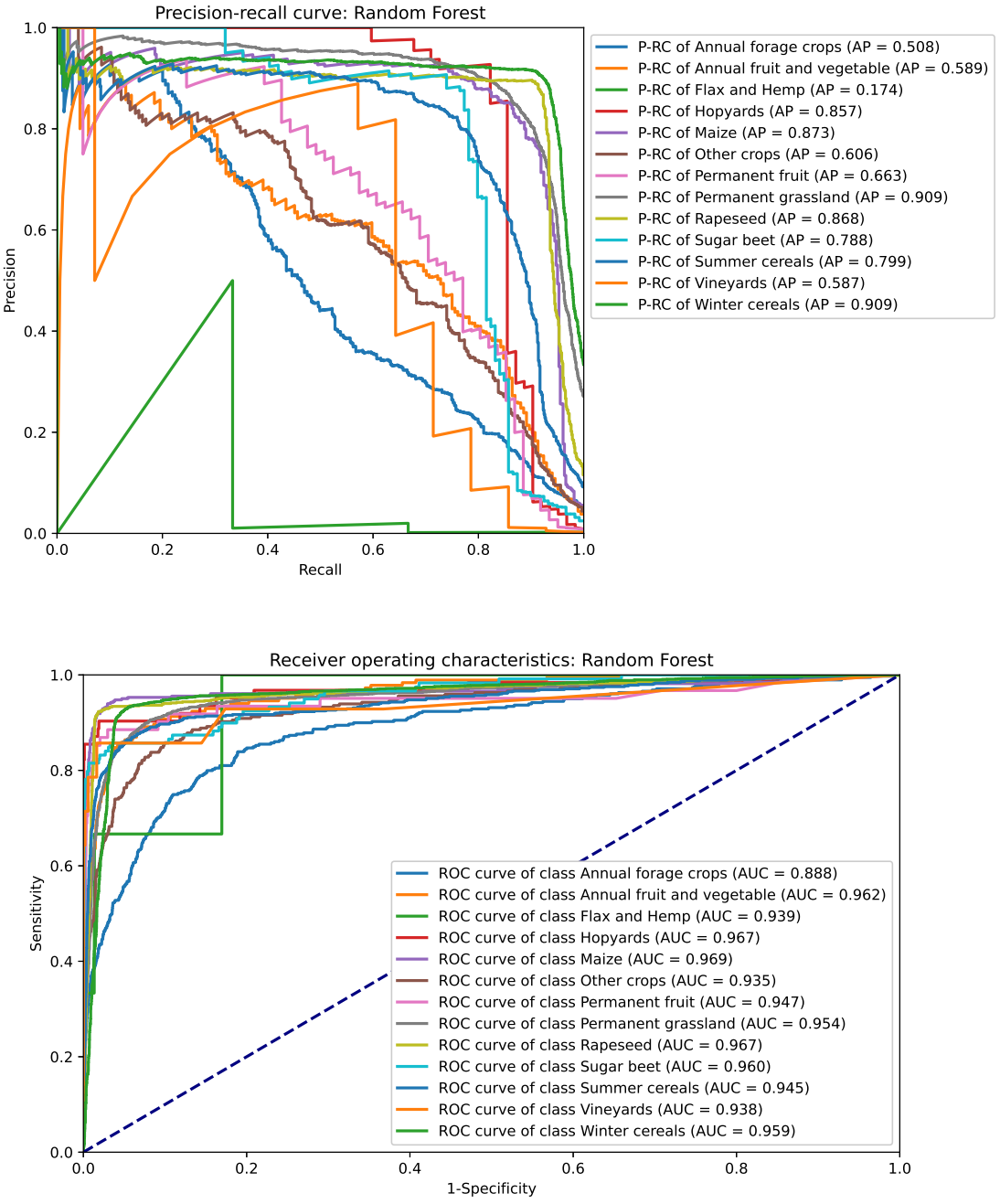
Figure 4.5: Random forest confusion matrix and test metrics applied on filtered dataset.

The overall accuracy is **0.8370** and weighted F_1 score is **0.8272**. The filtered dataset shows improvement of metrics or at least comparable metrics for all classes except of un-

dersampled *Vineyards*. One has to bear in mind that the datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ generally differ.

ROC and PR curves can be seen in the figure 4.6.

Figure 4.6: Random forest ROC and PR curves applied on filtered dataset.



As we have unbalanced dataset, it is much more convenient to look at the PR curve. PR curve gives similar answers like confusion matrix, with separable classes like *Hopyards*,

Maize, *Permanent grassland*, *Rapeseed* or *Winter cereals*. Less separable are *Sugar beet* and *Summer cereals*, which have similar PR curves. The remaining show poorer results, with *Flax and Hemp* being the by far worst.

4.3.2 XGBoost

Similarly to random forest, in order to deal with the imbalanced dataset, each field entry is weighted by the class proportion in the dataset. The XGBoost tends to struggle in time computation with multi-class classification with a lot of classes. As this algorithm is much slower than random forest, no cross-validation experiment was conducted. The chosen fixed hyperparameters are $d_{max} = 3$, 100 trees and learning rate $\eta = 0.1$.

Full dataset

The figure 4.7 shows the confusion matrix and test metrics, including accuracy, precisions, recalls and F_1 scores for each class, for the XGBoost classifier.

Figure 4.7: XGBoost confusion matrix and test metrics.

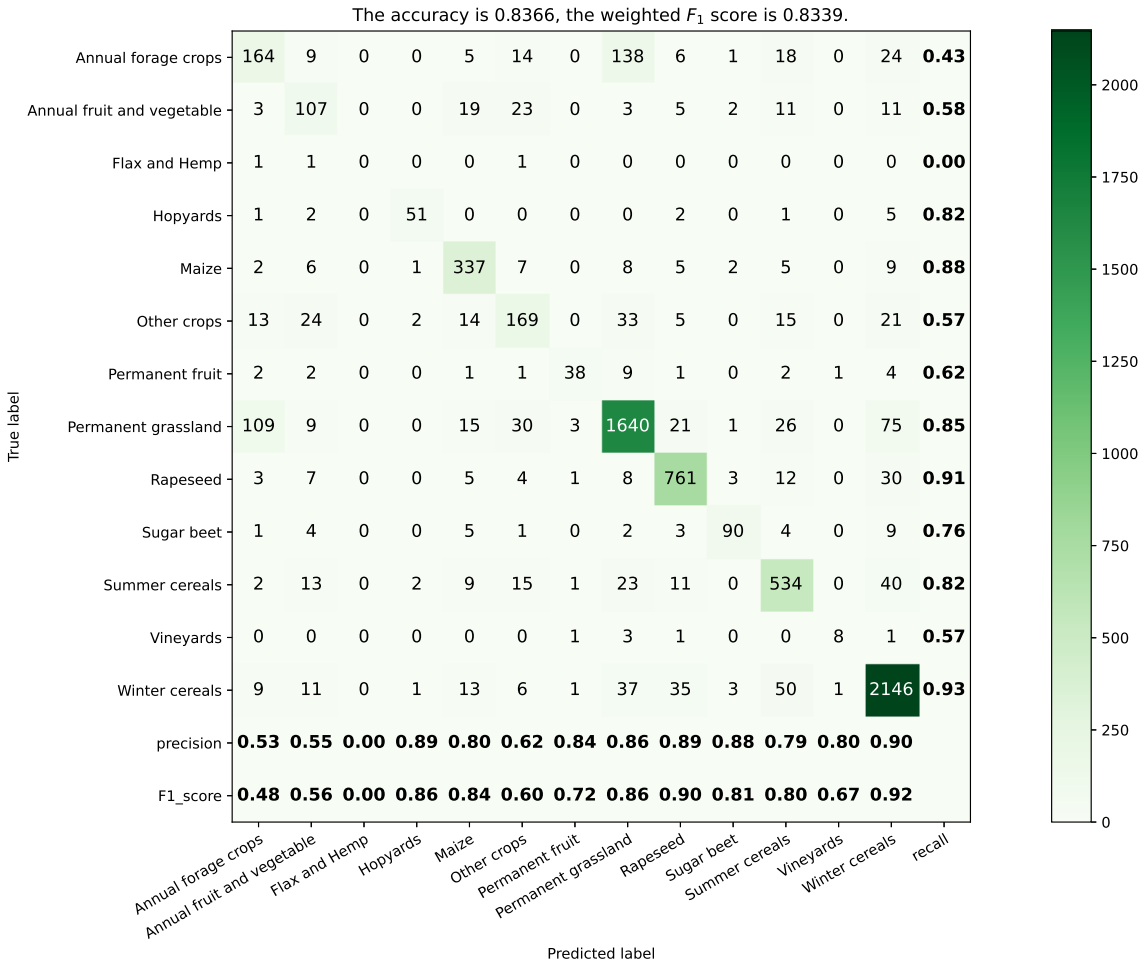
The overall accuracy is **0.7962** and weighted F_1 score is **0.7911**. The overall accuracy

is lower than the accuracy of random forest, but weighted F_1 and most of the F_1 scores are quite comparable. The classifier shows the same problems with separating *Permanent grassland* and other classes like *Annual forage crops*, *Other Crops* or *Winter cereals*. When one compares *Permanent grassland* and these classes, XGBoost gives much more emphasis on these classes (most visibly on the *Annual forage crops* class) than random forest. This explains the lower accuracy.

Filtered dataset

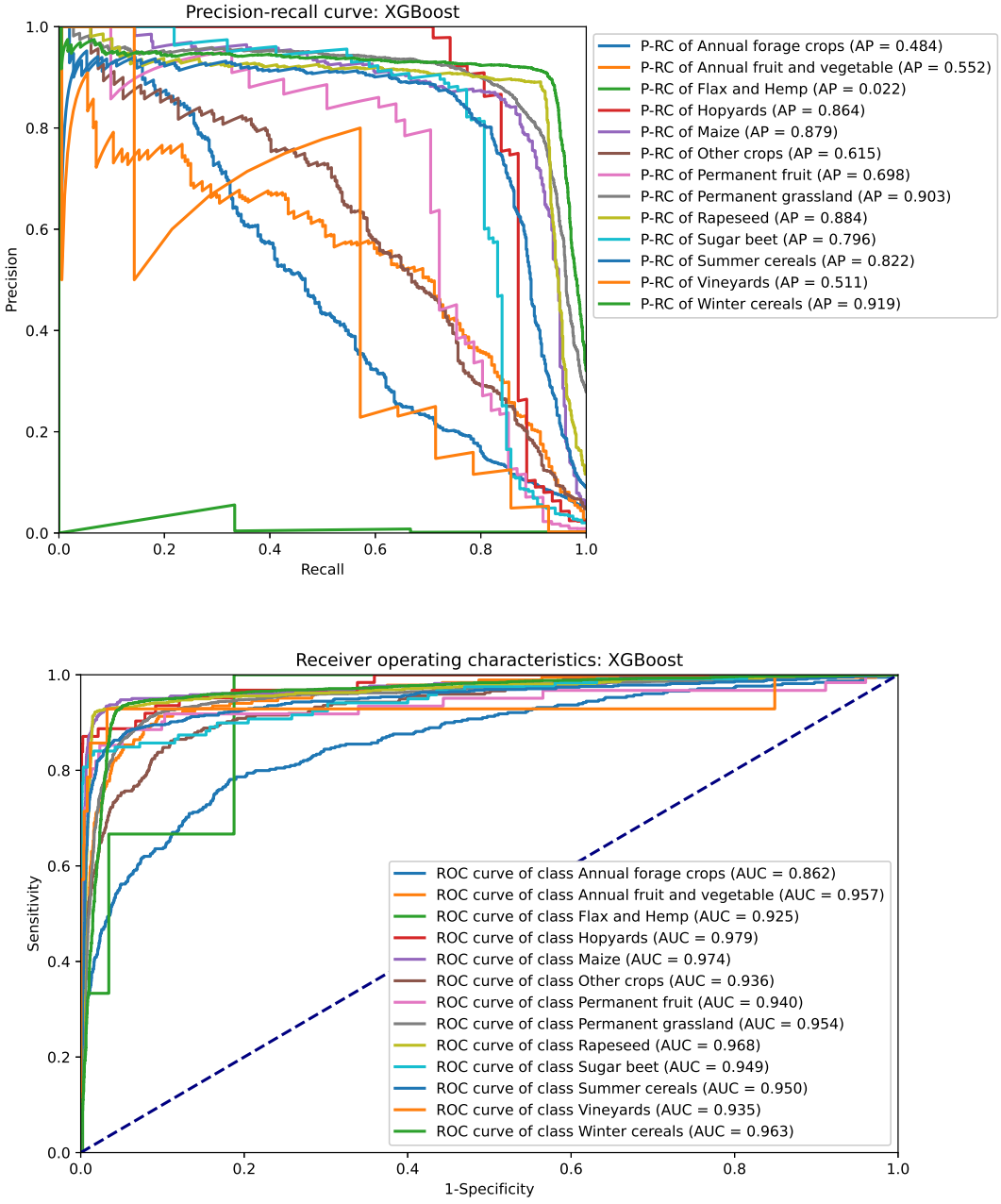
The figure 4.8 shows the confusion matrix and test metrics, including accuracy, precisions, recalls and F_1 scores for each class, for the XGBoost classifier applied on filtered dataset.

Figure 4.8: XGBoost confusion matrix and test metrics applied on filtered dataset.



The overall accuracy is **0.8366** and weighted F_1 score is **0.8339**. Again, the improvement from full dataset can be seen, with the similar behavior regarding *Permanent grassland*. In comparison with the random forest results, the accuracy is quite similar and weighted F_1 score is a bit higher. ROC and PR curves can be seen in the figure 4.9. They are quite similar to the random forest curves, with the same well-separable classes and less-separable classes.

Figure 4.9: XGBoost ROC and PR curves applied on filtered dataset.



4.4 Convolutional neural networks

There are several specific complications regarding using the CNN. CNN requires the input images to be the same size. This is unpleasant because the size of images varies significantly. Two approaches were attempted to tackle this problem. The first approach simply resizes all images to a given size, which loses some of the spatial information. The resized image size was set as 32×32 pixels, a few pixels more than field height and width median. The second

is to pad smaller images with zeros and crop the center of larger images to achieve same image size. As the experiments using resizing showed better overall performance metrics, experiments using padding and cropping are not included in this thesis.

The architecture used in the final CNN computations is ResNet-50 with layers applied to each time slice and additional convolutional layer before the network beginning which reduces the input band number from ten to three, all in all with 23,964,719 trainable parameters. Additional CNN hyperparameters are Adam optimizer with initial learning rate $\eta = 0.005$ and Glorot uniform weight initializer. For each experiment, we let the classifier train for 50 epochs without using early stopping, as we rather overfit and pick an optimal model later after training. Samples are reshuffled after each epoch to generalize more. Loss and performance metrics like accuracy, aggregate AUC and AP are monitored during the training. The best-performing model is selected based on the validation loss.

Experiments with the class weights (the same as for feature-based classifiers in section 4.3) were conducted, but this even worsened the performance metrics. CNN without augmenting the training set is presented in subsection 4.4.1 and with augmentation of the training set in subsection 4.4.2.

4.4.1 Without augmentation

Full dataset

Figure 4.10 shows the overall training and validation loss, accuracy and AP. For sake of simplicity, AUC is omitted. There is an epoch number on x-axis and corresponding loss, accuracy and AP on y-axes. Red line shows the optimal model in terms of overall validation loss. Validation loss is minimal at epoch 31, so we choose model 31 as our final model. U-shaped loss curve explained in chapter 2 can visibly be seen.

The figure 4.11 shows the confusion matrix and test metrics, including accuracy, precisions, recalls and F_1 scores for each class, for the CNN classifier. The overall accuracy is **0.8044** and weighted F_1 score is **0.7859**. The overall accuracy and weighted F_1 is lower than both of the random forest and XGBoost. The classifier predictions resembles the worsened ones of random forest, as it gives more emphasis on the *Permanent grassland* class and less emphasis on the other classes like *Annual forage crops*, *Other crops* or *Winter cereals*.

Filtered dataset

Figure 4.12 shows the overall training and validation loss, accuracy and AP. Validation loss is minimal at epoch 38, so we choose model 38 as our final model. There are three epochs with validation loss spikes, but after each classifier predictions rapidly improved. U-shaped loss curve is less visible, as the validation loss stays at approximately 0.75 throughout the whole training.

The figure 4.13 shows the confusion matrix and test metrics, including accuracy, precisions, recalls and F_1 scores for each class, for the CNN classifier applied on filtered dataset. The overall accuracy is **0.8195** and weighted F_1 score is **0.8095**. Similar behavior as for full dataset can be seen. The overall accuracy and weighted F_1 is lower than both of the random forest and XGBoost with problems with separating *Permanent grassland* and other classes like *Annual forage crops*, *Other Crops* or *Winter cereals*.

ROC and PR curves can be seen in the figure 4.14. They resemble the ones of random forest and XGBoost, but with all classes less separable.

Figure 4.10: CNN training plots.

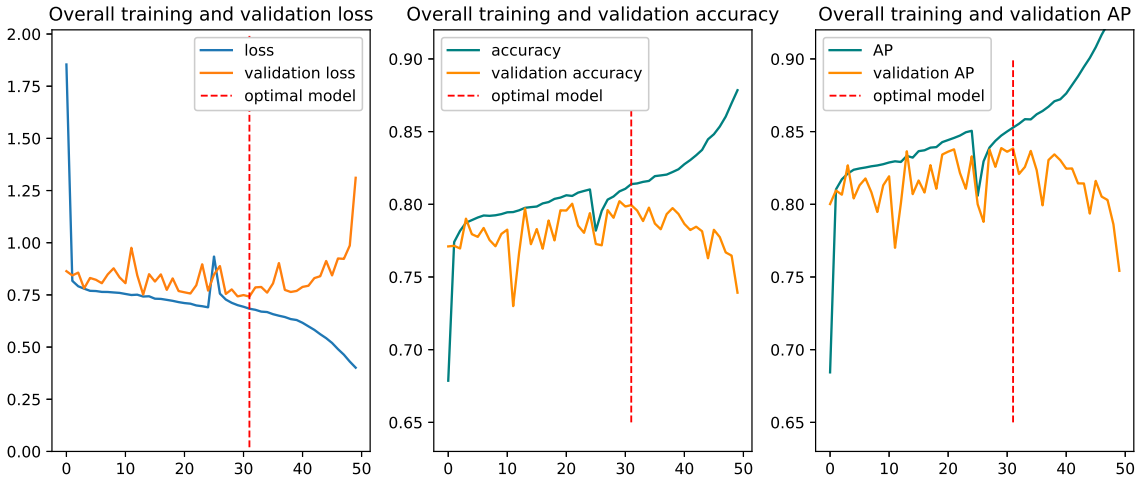


Figure 4.11: CNN confusion matrix and test metrics.

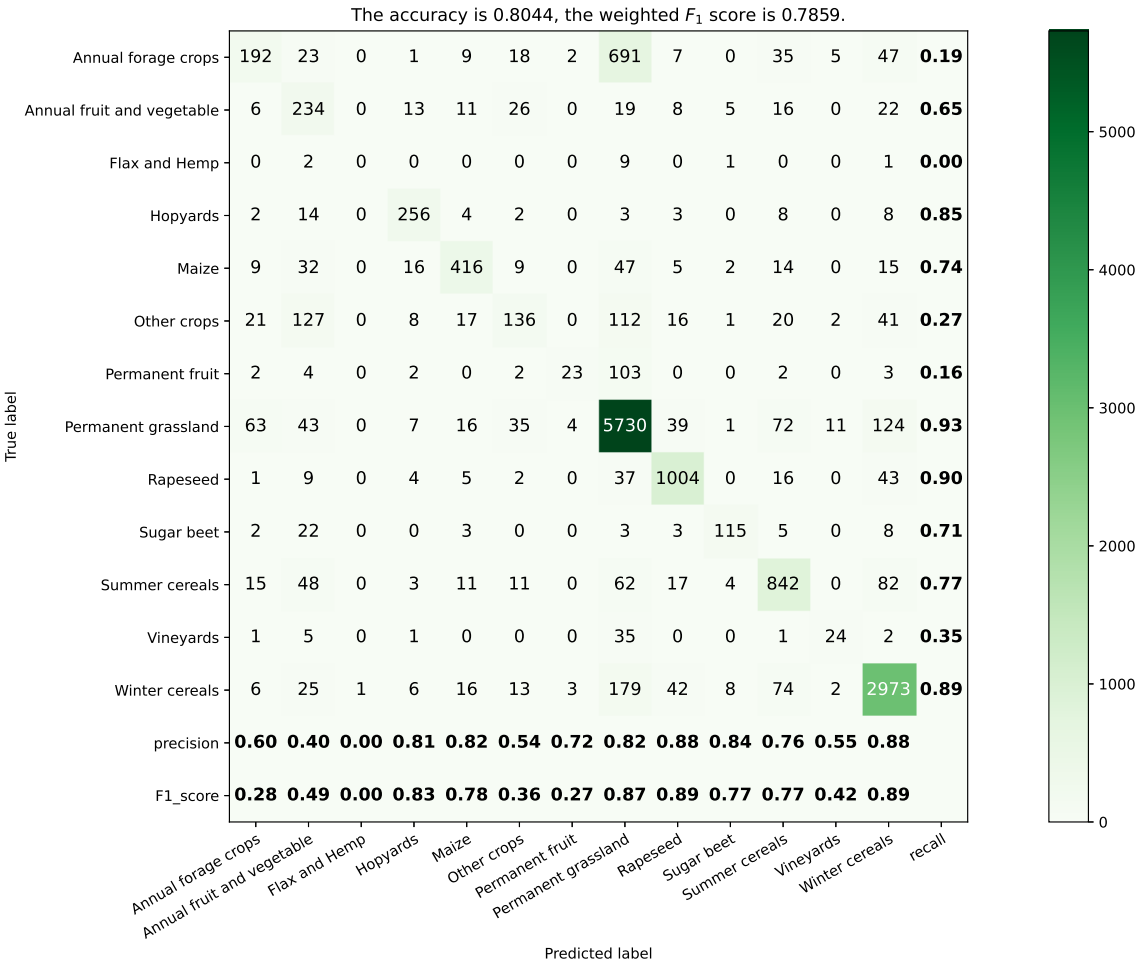


Figure 4.12: CNN training plots applied on filtered dataset.

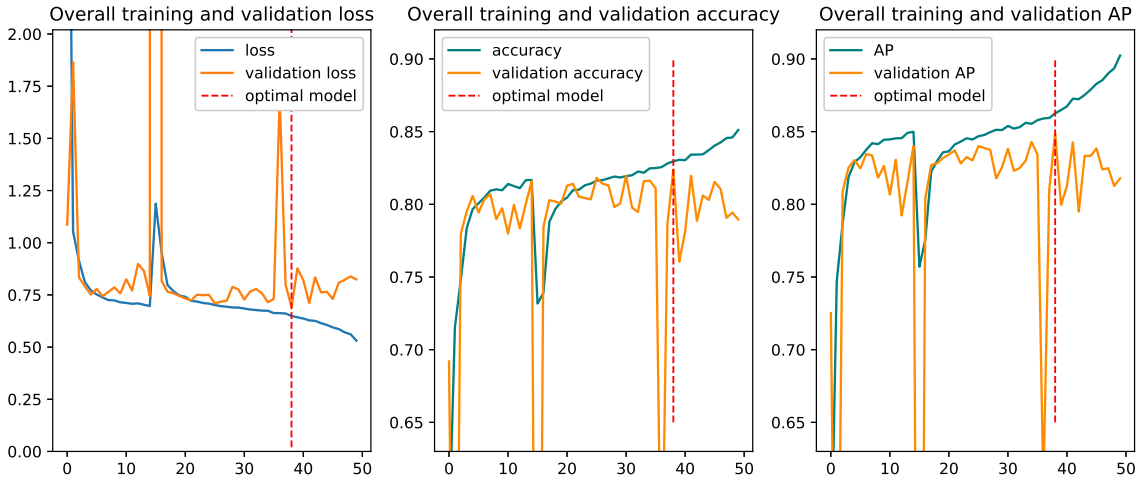


Figure 4.13: CNN confusion matrix and test metrics applied on filtered dataset.

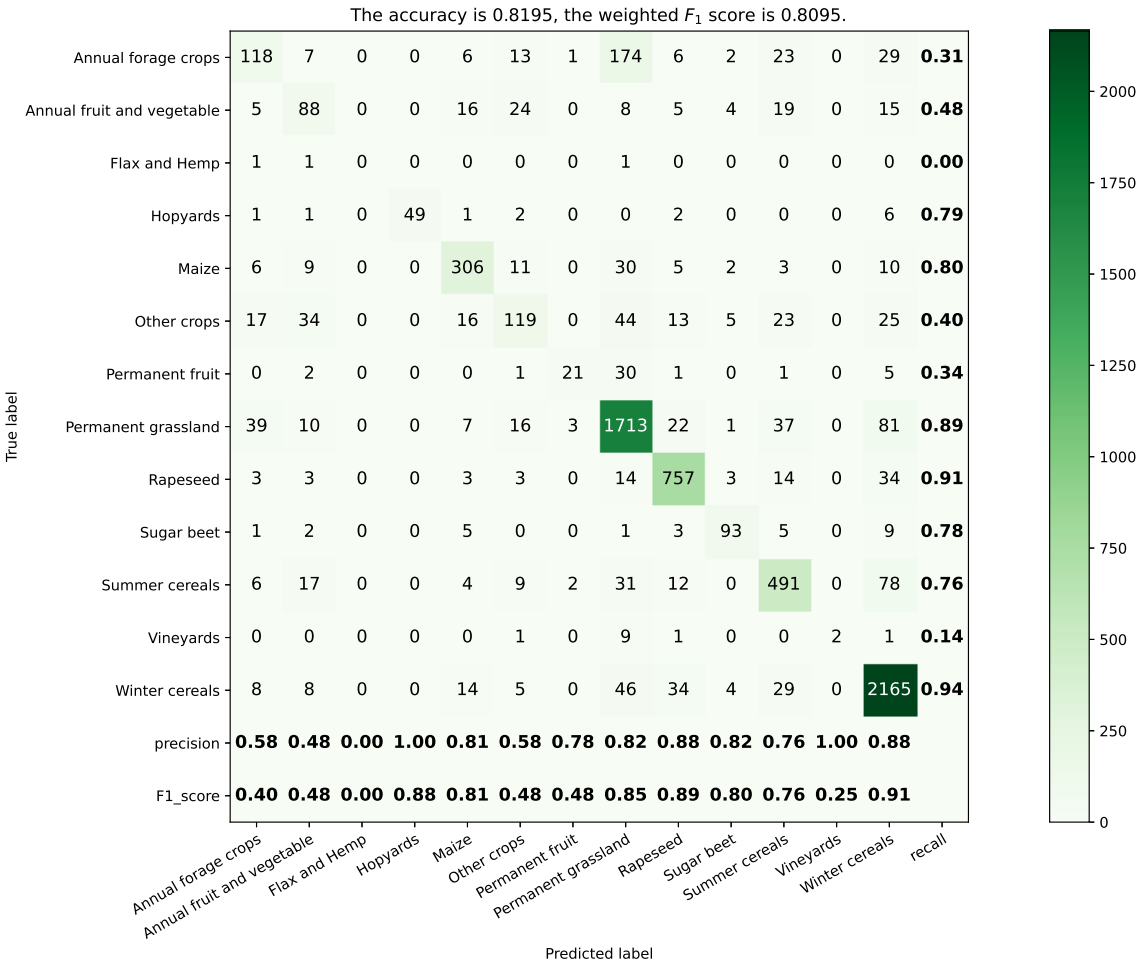
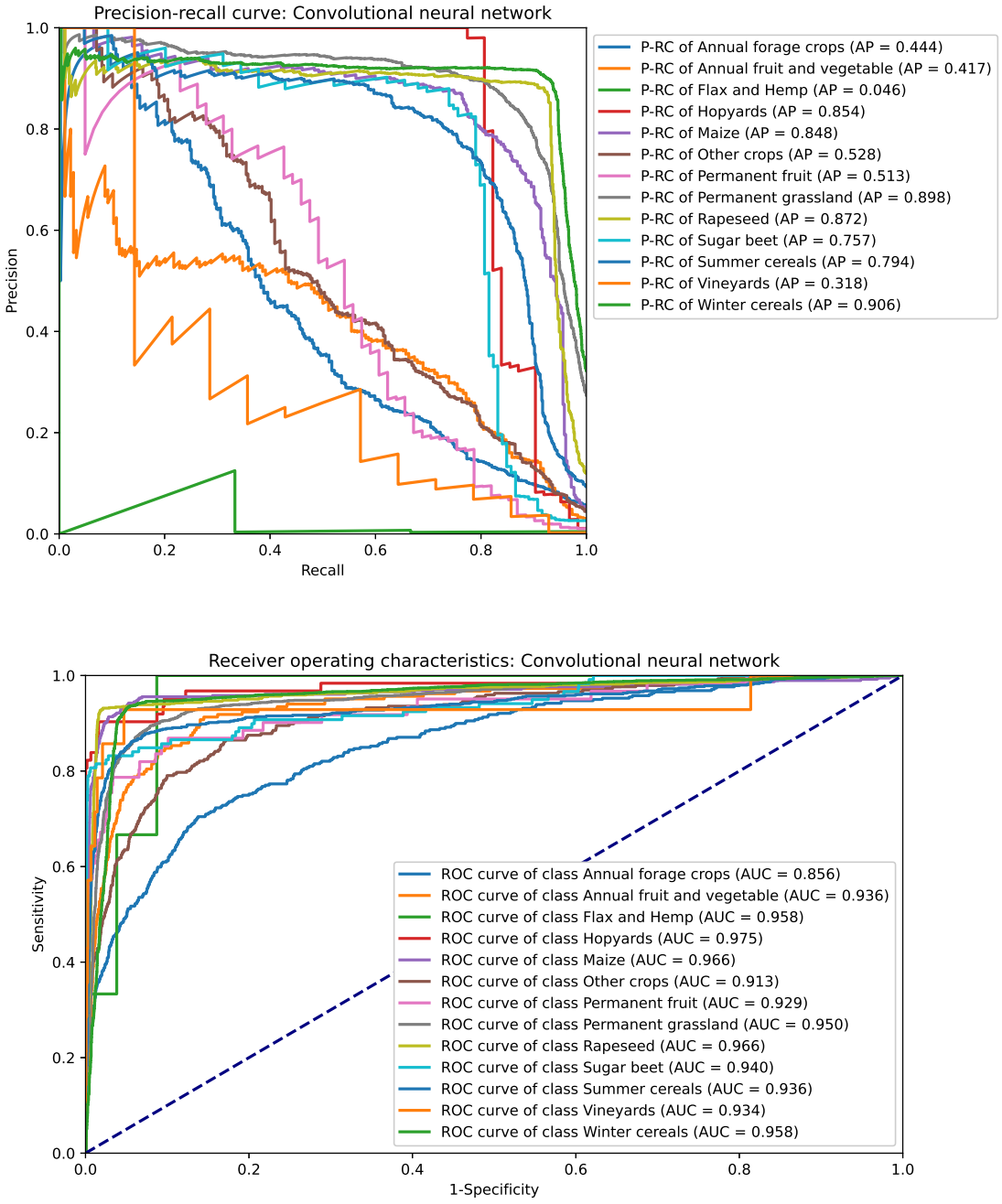


Figure 4.14: CNN ROC and PR curves applied on filtered dataset.



4.4.2 Augmented training set

In this subsection, we present a way of possible improvement of CNN from the previous subsection. As will be seen, the improvement was unfortunately not achieved.

Original training set was augmented to enlarge the number of training images with validation and test set staying the same. For undersampled classes additional augmented

images were generated. For oversampled classes, all images were kept in the training set.

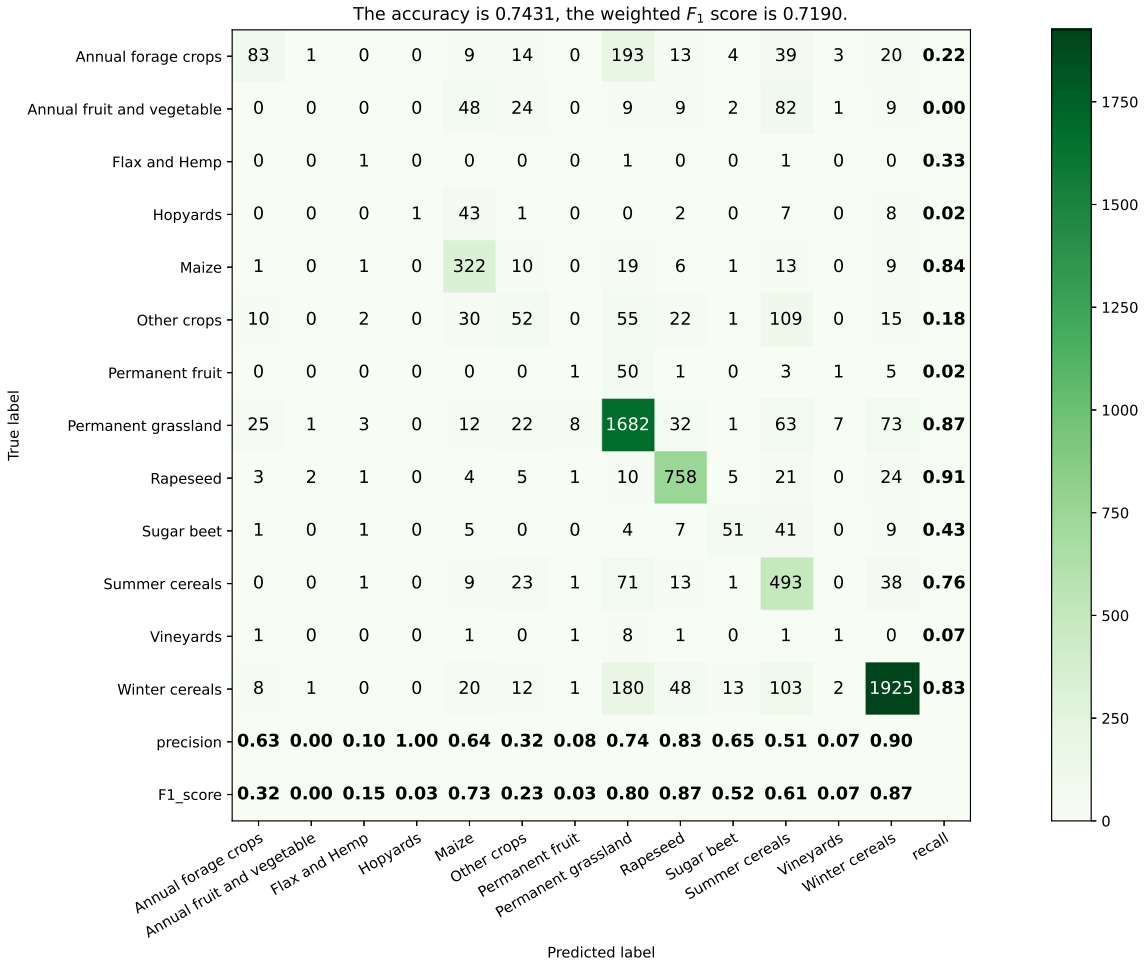
Results for only filtered dataset $\mathcal{D}_2$ are presented in this thesis. The limit of training images per class was arbitrarily set to 1,000. This means that seven out of thirteen classes were augmented. This also means that original training set of 23,122 images was enlarged to 27,722 training images.

The augmentation techniques were based on basic image manipulation and taken from [37]. This included randomly flipping the image both from left to right and up to down, random cropping and affine transformations (only scaling, translating and rotating the image). Image was augmented with the same augmentation settings in each time slice.

Overall training and validation loss, accuracy and AP can be seen in the appendix A.2 with the minimal validation loss at epoch 7 and clearly visible U-shaped loss curve.

The overall accuracy is only **0.7431** and weighted F_1 score is only **0.7190**, which is a rapid decrease. This shows that augmentation merely confuses the classifier.

Figure 4.15: CNN confusion matrix on filtered dataset.



4.5 Classifier comparison

The table 4.2 shows the overall results of three classifiers from previous sections, in terms of test accuracy and test weighted F_1 score on full dataset $\mathcal{D}_1$ and filtered dataset $\mathcal{D}_2$.

Table 4.2: Classification performance metrics for given classifiers on two datasets.

	full dataset $\mathcal{D}_1$		filtered dataset $\mathcal{D}_2$	
	test accuracy	test F_1 score	test accuracy	test F_1 score
random forest	**0.8161**	0.7987	**0.8370**	0.8272
XGBoost	0.7962	**0.7991**	0.8266	**0.8339**
CNN	0.8044	0.7859	0.8195	0.8095

The highest value in column is bold. It can be seen that in terms of test accuracy, random forest is the best-performing classifier, on the other hand, in terms of test weighted F_1 score, XGBoost is the best-performing classifier. However, the differences in metrics are minimal. The worst-performing classifier is CNN, even with suggested training set augmentation. These results are supported by confusion matrices, additional metrics like individual recalls, precisions, F_1 scores, ROC and PR curves and their AUCs and APs. All ROC and PR curves for full dataset can be seen in A.1.

Similar behavior of all classifiers on full dataset $\mathcal{D}_1$ and filtered dataset $\mathcal{D}_2$ was observed, so different-sized images do not alter the classification decisions. Results of feature-based classifiers are better than results of CNN and even results of CNN which uses resizing input images are better than results of CNN which uses padding and cropping approach. This indicates that spatial information has little to no importance in the classification.

In prediction of classes *Annual forage crops*, *Other Crops*, *Permanent grassland* and *Winter cereals*, random forest and CNN give more emphasis on the *Permanent grassland* class and less emphasis on classes *Annual forage crops*, *Other Crops* or *Winter cereals*. XGBoost gives the emphasis on the classes the other way around.

In situation when one needs fast algorithm and possibly cross-validation, random forest is based on our analysis preferred. On the other hand, if computation time is not an issue for the user, XGBoost with optimal hyperparameters would perform even better.

Conclusion

The goal of this thesis was to apply feature-based classifiers and convolutional neural networks (CNNs) on satellite optical imagery and find the classifier best suited for the task. In order to reach the mentioned goal and to apply the classifiers correctly, the data sources were described, the theoretical background of both machine learning and deep learning was presented and comprehensive study of selected machine learning algorithms and CNNs was given.

For the classification, two datasets with images of fields in Stredni Cechy in Czech Republic region were prepared. Both of these were created by combining Sentinel-2 satellite images and agricultural data provided by the State Agricultural Invervention Fund, which were used as ground truth. Each data sample contained field images in thirteen times throughout the year 2019 each with ten bands with cloud filtering, both Sentinel-2 whole tile filtering and pixel filtering. The first dataset was kept unchanged, the second was filtered so only fields with image size greater than roughly 200×200 meters were kept. Both datasets were later divided into training, testing and either validation set or k validation sets for k-fold cross-validation.

Feature-based classifiers used sample mean and sample standard deviation for each time and each band as features. The selected feature-based classifiers were random forest and XGBoost, the selected CNN architecture was modified ResNet-50 with training and testing performed on GPUs. 5-fold cross-validation hyperparameter search was conducted for random forest classifier.

Random forest was the best-performing classifier based on the test accuracy with 81.61 % on the full dataset and 83.70 % on the filtered dataset. XGBoost was the best-performing classifier based on the test weighted F_1 score with 79.91 % on the full dataset and 83.39 % on the filtered dataset. The worst-performing classifier was the CNN, even with suggested training set augmentation. The most separable classes in terms of F_1 score for all classifiers and all datasets were *Hopyards*, *Maize*, *Permanent grassland*, *Rapeseed*, *Sugar beet* and *Winter cereals*. The least separable classes were *Annual forage crops*, *Annual fruit and vegetable*, *Other crops* and *Permanent fruit* and by far the worst *Flax and Hemp*. By judging all the classification performance metrics and algorithm speed, the random forest is the most preferred classifier. The feature-based classifiers performed better than the CNN, which indicates that spatial information has little to no importance in the classification process.

Of course, there are still some improvements which could be made. Above all, preference of classes and classes selection should be examined thoroughly. The performance could also improve with larger geographical area, and thus more data samples, and with more timestamps throughout the year. Another types of pre-processing and algorithms, such as recurrent ANN long short-term memory or recurrent convolutional neural network could lead to performance improvements. In order for the found best-performing classifier

to be fully applicable on an unknown field sample, all of the pre-processing steps, which include obtaining the fields image in the same timestamps and feature computation, would have to be subsequently applied. For a user-friendly classification application, it would be convenient to implement a graphical user interface.

Bibliography

[1] Audebert, N., Le Saux, B., and Lefevre, S. Deep learning for classification of hyperspectral data: A comparative review. *IEEE Geoscience and Remote Sensing Magazine 7*, 2 (2019), 159–173.

[2] Bargiel, D. A new method for crop classification combining time series of radar images and crop phenology information. *Remote Sensing of Environment 198* (2017), 369 – 383.

[3] Belgiu, M., and Csillik, O. Sentinel-2 cropland mapping using pixel-based and object-based time-weighted dynamic time warping analysis. *Remote Sensing of Environment 204* (2018), 509 – 523.

[4] Belkin, Mikhail and Hsu, Daniel and Ma, Siyuan and Mandal, Soumik. Reconciling modern machine learning practice and the bias-variance trade-off, 2018.

[5] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[6] Breiman, L. Random forests. *Machine Learning 45* (2001), 5–32.

[7] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[8] Copernicus team. The EU Earth observation and monitoring programme. https://www.copernicus.eu/sites/default/files/2019-06/The_EU_Earth_Observation_and_Monitoring_Programme-EN-20190405-WEB.pdf, 2019. online, cit. 20. 12. 2020.

[9] ESA Sen4cap Team. Quick user guide for crop diversification use case 1.2. http://esa-sen4cap.org/sites/default/files/Sen4CAP_UserGuide_CropDiversification_V1.2.pdf, 2020. online, cit. 20. 9. 2020.

[10] ESA Sentinel-2 Team. Global monitoring for environment and security Sentinel-2 mission requirements document. http://esamultimedia.esa.int/docs/GMES/Sentinel-2_MRD.pdf, 2010. online, cit. 20. 12. 2020.

[11] ESA Sentinel-2 Team. Sentinel-2 msi technical guide. https://earth.esa.int/web/sentinel/technical-guides/sentinel-2-msi, 2020. online, cit. 20. 9. 2020.

[12] Freund, Y., and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci. 55*, 1 (Aug. 1997), 119–139.

[13] Friedman, J. H. Greedy function approximation: A gradient boosting machine. *Annals of Statistics 29* (2000), 1189–1232.

[14] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics* (2010).

[15] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. The MIT Press, 2016.

[16] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1026–1034.

[17] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[18] Hosmer, D. W., and Lemeshow, S. *Applied Logistic Regression, Second Edition*. John Wiley & Sons, 2000.

[19] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (2015), ICML'15, p. 448–456.

[20] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 1097–1105.

[21] Kuthan, T. Classification of selected agricultural crops from time series of Sentinel-2 and Planetscope imagery in Kutnohorsko model area. Master's thesis, Charles University, 2018.

[22] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE* (1998), pp. 2278–2324.

[23] Levenberg, K. A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathematics 2* (1944), 164—168.

[24] Louppe, G. *Understanding Random Forests: From Theory to Practice*. PhD thesis, University of Liege, Belgium, 10 2014.

[25] LPIS. Uživatelská dokumentace pLPIS- Veřejný Registr půdy. http://eagri.cz/public/web/file/2091/Uzivatelska_prirucka_pLPIS.pdf, 2020. online, cit. 20. 9. 2020.

[26] Mazzia, V., Khaliq, A., and Chiaberge, M. Improvement in land cover and crop classification based on temporal features learning from Sentinel-2 data using recurrent-convolutional neural network (r-cnn). *Applied Sciences 10*, 1 (Dec 2019), 238.

[27] McCulloch, W. S., and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*, 4 (Dec 1943), 115–133.

[28] Mitchell, Thomas M. *Machine Learning*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 1997.

[29] Nesterov, Y. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady 27* (1983), 372–367.

[30] Polyak, B. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics 4* (12 1964), 1–17.

[31] Quinlan, J. R. Induction of decision trees. *Machine Learning 1* (1986), 81–106.

[32] Račič, M., Oštir, K., Peressutti, D., Zupanc, A., and Čehovin Zajc, L. Application of temporal convolutional neural network for the classification of crops on Sentinel-2 time series. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLIII-B2-2020* (2020), 1337–1342.

[33] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* (1958), 65–386.

[34] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning Representations by Back-propagating Errors. *Nature 323*, 6088 (1986), 533–536.

[35] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[36] Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal 27*, 3 (7 1948), 379–423.

[37] Shorten, C., and Khoshgoftaar, T. A survey on image data augmentation for deep learning. *Journal of Big Data 6* (07 2019).

[38] Sola, I., García-Martín, A., Sandonís-Pozo, L., Álvarez Mozos, J., Pérez-Cabello, F., González-Audícana, M., and Montorio Llovería, R. Assessment of atmospheric correction methods for Sentinel-2 images in mediterranean landscapes. *International Journal of Applied Earth Observation and Geoinformation 73* (2018), 63 – 76.

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

[40] Žubrietovský, L., Švábová, L., and Šváb, O. Sentinel 2 – datové specifikace. https://collgs.czechspaceportal.cz/wp-content/uploads/2020/08/sentinel2czfinal.pdf, 2017. online, cit. 20. 12. 2020.

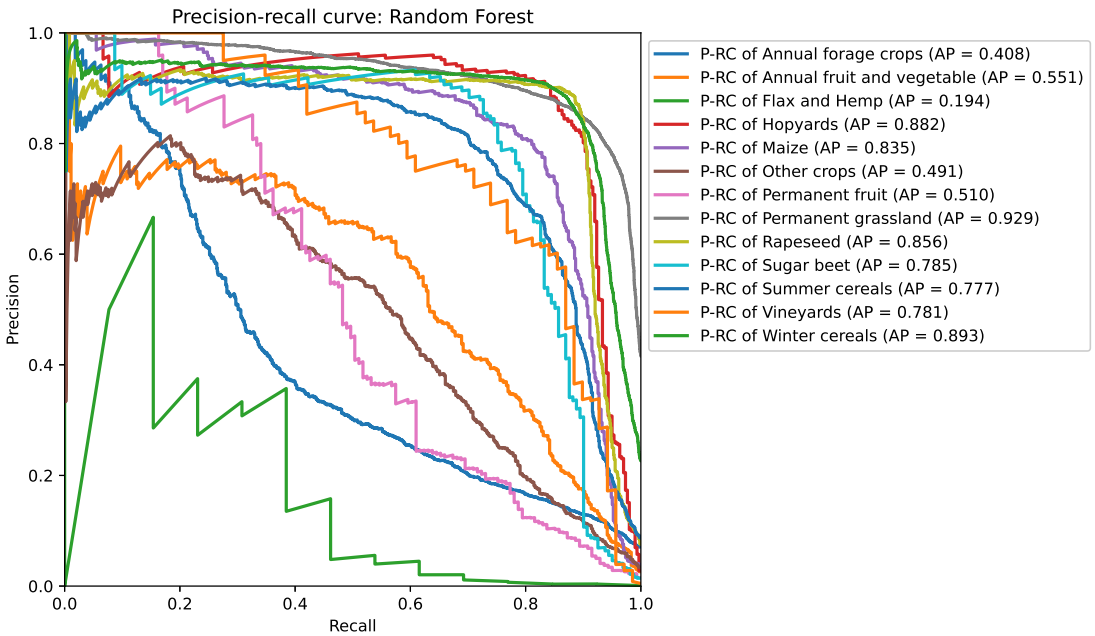
Appendix A

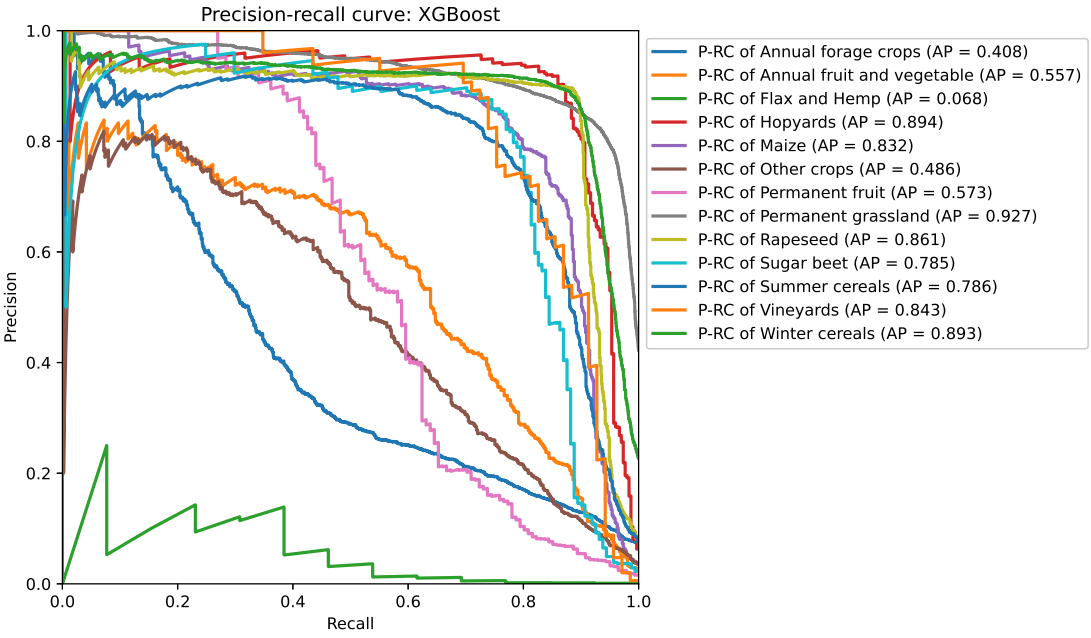
Plots

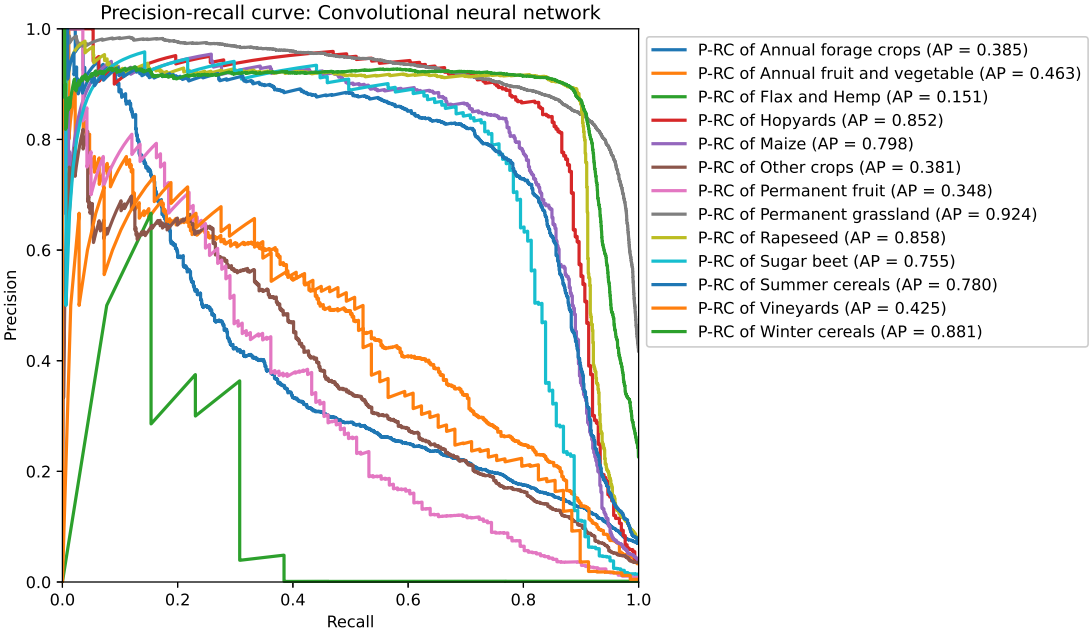
A.1 Precision-recall curves applied on full dataset

The different curve behavior between the more separable and less separable classes is much more apparent.

Figure A.1: Full dataset random forest, XGBoost and CNN PR curves.




Precision-recall curve: XGBoost
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
P-RC of Annual forage crops (AP = 0.408)
P-RC of Annual fruit and vegetable (AP = 0.557)
P-RC of Flax and Hemp (AP = 0.068)
P-RC of Hopyards (AP = 0.894)
P-RC of Maize (AP = 0.832)
P-RC of Other crops (AP = 0.486)
P-RC of Permanent fruit (AP = 0.573)
P-RC of Permanent grassland (AP = 0.927)
P-RC of Rapeseed (AP = 0.861)
P-RC of Sugar beet (AP = 0.785)
P-RC of Summer cereals (AP = 0.786)
P-RC of Vineyards (AP = 0.843)
P-RC of Winter cereals (AP = 0.893)


Precision-recall curve: Convolutional neural network
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
P-RC of Annual forage crops (AP = 0.385)
P-RC of Annual fruit and vegetable (AP = 0.463)
P-RC of Flax and Hemp (AP = 0.151)
P-RC of Hopyards (AP = 0.852)
P-RC of Maize (AP = 0.798)
P-RC of Other crops (AP = 0.381)
P-RC of Permanent fruit (AP = 0.348)
P-RC of Permanent grassland (AP = 0.924)
P-RC of Rapeseed (AP = 0.858)
P-RC of Sugar beet (AP = 0.755)
P-RC of Summer cereals (AP = 0.780)
P-RC of Vineyards (AP = 0.425)
P-RC of Winter cereals (AP = 0.881)

A.2 Convolutional neural network training plots

Figure A.2: CNN training plots with augmentation on filtered dataset.