Bioinformatics

Freie Universität Berlin

Master Thesis

Taxonomic Rank Classification of Fungi using CNN Image Classifiers

---

Author:

Stefan Stiller, # 3902442
Bioinformatics, Institute of Bioinformatics
Freie Universität Berlin

Acedemic Supervisor:

Masahiro Ryo, Ph.D.
Institute of Biology, Berlin-Brandenburg Institute of Advanced Biodiversity Research
Freie Universität Berlin

Acedemic Advisors:

Matthias Rillig, Prof. Dr.
Institute of Biology, Rillig Group - Ecology of Plants
Freie Universität Berlin

Tim Conrad, Prof. Dr.
Institute of Mathematics, Medical Bioinformatics Group
Freie Universität Berlin

Submitted: Friday 17th July, 2020

Contents

Before I start I want to state my deepest gratitude and respect to the following people

to my beloved friend Nadine Thomé for her never ending support through a most challenging time even against her better judgment, but even more for her true friendship and the path we share,

to Masahiro Ryo for his guidance, patience and kindness, the words of encouragement and his remarkable and beautiful gift to support people in personal growth on an eye to eye level.

You are shining lights.

Part I

Introduction

Plant and wildlife categorization goes way back to ancients Greece. In the 18th century it was Carolus Linnaeus who more or less invented our modern system of taxonomy and classification, which was then the idea of arranging them into logical classes based on their appearance and characteristics. Though the concepts behind taxonomic classification prevail, modern species identification is based on DNA sequencing methods. With recent breakthroughs in artificial intelligence and image classification by deep learning, within this study I am going back to visual classification.
In biological classification, **taxonomic rank** is the relative level of a group of organisms (a **taxon**) according to a taxonomic hierarchy (Turland, Wiersema, Barrie, Greuter, Hawksworth, Herendeen, Knapp, Kusber, Li, Marhold, May, Mcneill, Monro, Prado, Price, and Smith 2018). Advancing the taxonomic tree from more general to more specific, certain defining traits and features become more differentiated. Likewise do common ancestors within the same super class share common traits and features. The major taxonomic groups are Life, Realm, Kingdom, Phylum, Class, Order, Family, Genus and Species. Given an organism, **taxonomic classification** is the task of classifying that organism to a taxon according to a taxonomic rank.
Soil fungi are microscopic cells that can be single celled or grow in long threadlike structures or hyphae that make a mycelium. They play a crucial role in soil health and biodiversity, (Frac, Hannula, Belka, and Jedryczka 2018). (Lehmann, Zheng, Ryo, Soutschek, Roy, Rongstock, Maaß, and Rillig 2020) says that hyphal growth speed and the complexity of the hyphal structure are phylogenetically conserved at least at phylum level. I am going to study the application of deep learning models in taxonomic classification of soil fungi images and explain the reasoning behind the model's decision. I conclude their application to be valid and moreover find explanations that support (Lehmann, Zheng, Ryo, Soutschek, Roy, Rongstock, Maaß, and Rillig 2020) work.
Working on taxonomic classification data in microbial ecology in general is a challenge to work with for a number of characteristics. On the one hand dataset **size** is usually fairly small, especially for image classification. On the other hand data validity is challenged by **missing values** that are not easy to impute in a meaningful way and by class imbalance. As in many bioinformatics applications **class imbalance** poses a huge problem, as it introduces a bias or drift in learning. The usability of techniques that address this problem, such as over- and undersampling however underlie limitations

of size as well as bio-morphological traits that constraint augmentation. Methods and metrics must be carefully chosen as most the most established once are prone to class imbalance.
Like many real world examples taxonomic classification is **hierarchically structured**, whereas machine learning algorithms work flat. To include the information content that the hierarchical structure offers algorithms need to be designed in a special way. The most difficulty as well as hook for this study is the fact that fungi do offer very few traits for **visual classification**. The hope however is to distinguish certain visual traits that are influential for classification. The last but not least there is the challenge of **standardization** of the biological experiments for easier machine learning application, as like many others this study was not planned to include machine learning in the first place. For fungi factors influencing standardization may for instance be the growth time or medium, that effect the surface cover or the color. Moreover other non-biological effects could influence the outcome of the machine learning experiment, such as lighting or the usage of different scanners, to name only some.
In this work I am going to study the applicability of deep learning for taxonomic classification of mycorrhizal fungi and to find explanations behind classification. This work comprises three main parts. The data part, in which I am going to explain the data, necessary pre-work and how to cope with missing values and class imbalance. The deep learning and explanation part are structured in *Theory and Experimental Setup* and *Results*. I explain the theory behind deep learning and am going to build three models with respect to the hierarchical structure. Those models are going to be tested for performance and stability and their results are going to be compared. To find explanations I first explain the model agnostic approach LIME which I will then apply for the trained models, to find visual explanations for the classification.

Part II

Data

1 FunTrait

The FunTrait dataset stems from and subsets the FunTrait project, which is funded by the *German Biodiversity Exploratories*. This particular part originates from soil screenings in northern Germany and is made available by courtesy of Dr. Stefan Hempel and Juan Deñas at "Institut für Biologie, Freie Universität Berlin". The dataset is unpublished as of July 2020, but it can be available from the authors. *FunTrait* studies "the functional significance of fungi in soil based ecosystem services, such as soil structure formation, nutrient cycling, and plant productivity" (Stefan Hempel 2020). Isotopes are identified using a DNA based method and the cultures are screened "for enzymatic activities related to the degradation of complex organic compounds and nutrient cycling in the soil".

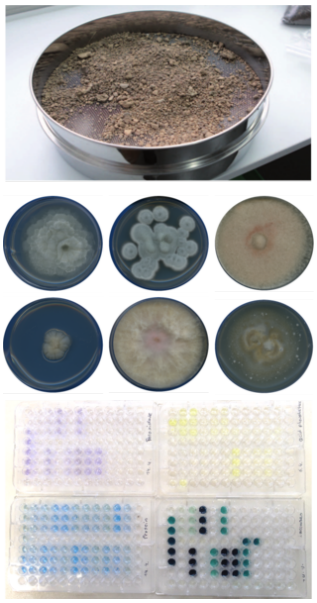


Figure 1: FunTrait development process. Cultures are bred in Petri dishes, identified using a DNA based method and its enzymatic activities analysed.(Illustration by (Stefan Hempel 2020))

1.1 Funtrait Dataset

Initially this work was based on a partial release of the FunTrait set, from 19.03.2019 consisting of 270 samples. As a newer version of the data was released on 18.03.2020, comprising of 606 samples, this work was updated to this new version. The cultures used in this dataset grew for approximately 10 days under a lab-controlled condition with an ambient temperature (approx. 15-20 degrees celcius) given sufficient nutrients and water. They were scanned and partly analyzed, i.e. a subset has been classified according to taxonomy using the molecular technique. Since the majority of soil fungi are taxonomically unclassified yet, we used data which are only taxonomically distinguishable.

1.1.1 Taxon-Trait Database

The dataset comes in two parts, a csv-file that we call database, containing label information for each isotope, i.e. taxonomic data as well as enzymatic activities and for the other part an image set. The isotope's taxons that are listed in the database are defined according to six taxonomic ranks in hierarchic order; phylum > class > order > family > genus > species.

1.1.2 Image Dataset

The images come as grouped scans of up to 12 cultures in Petri dishes put before a blue background. Only for the genetically identified cultures we can say, that the Petri dishes are opened for photo taking and that the image are taken in direction top to bottom. The rest of the set includes photos that are taken in flipped direction, may have a closed lid as well as a non standardized growth time.

In the group image the Petri dishes are arranged in a semi standardized way, three dishes in a row, counting from right to left in a 3 x 4 matrix, though their exact location may vary.

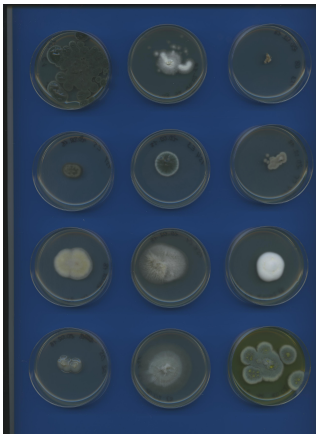


Figure 2: Group Image as provided by the FunTrait dataset.

1.2 Statistics

The original image data set consists of 221 grouped images with up to 12 isotopes. However only 606 cultures where genetically identified and labeled according to taxonomic rank. The taxon labels include missing data as listed in table 1. The highest amount of missing values is in species with 181, followed by genus with 131 and after a huge gap 38 for family.

Figure 3 (Top) shows the distribution of unique taxonomic classes according to rank. As with increasing taxonomic rank specializations increase a broader amount of classes emerges, for example from 5 in rank phylum, 11 in class, to 166 in species level. Figure 3 (Bottom) shows the distribution within a taxonomic rank. We can see that especially for higher taxonomic ranks most samples lie within one dominant class and the average samples per class decrease. The classes are not uniformly distributed, we speak of **class imbalance**. In lower taxonomic ranks diversification increases and

the phenomenon of class imbalance decreases. Taxonomic relations are shown in figure 1.2.

Taxonomic Rank	Missing Values
Phylum	3
Class	11
Order	15
Family	38
Genus	131
Species	181

Table 1: Missing values per taxonomic rank

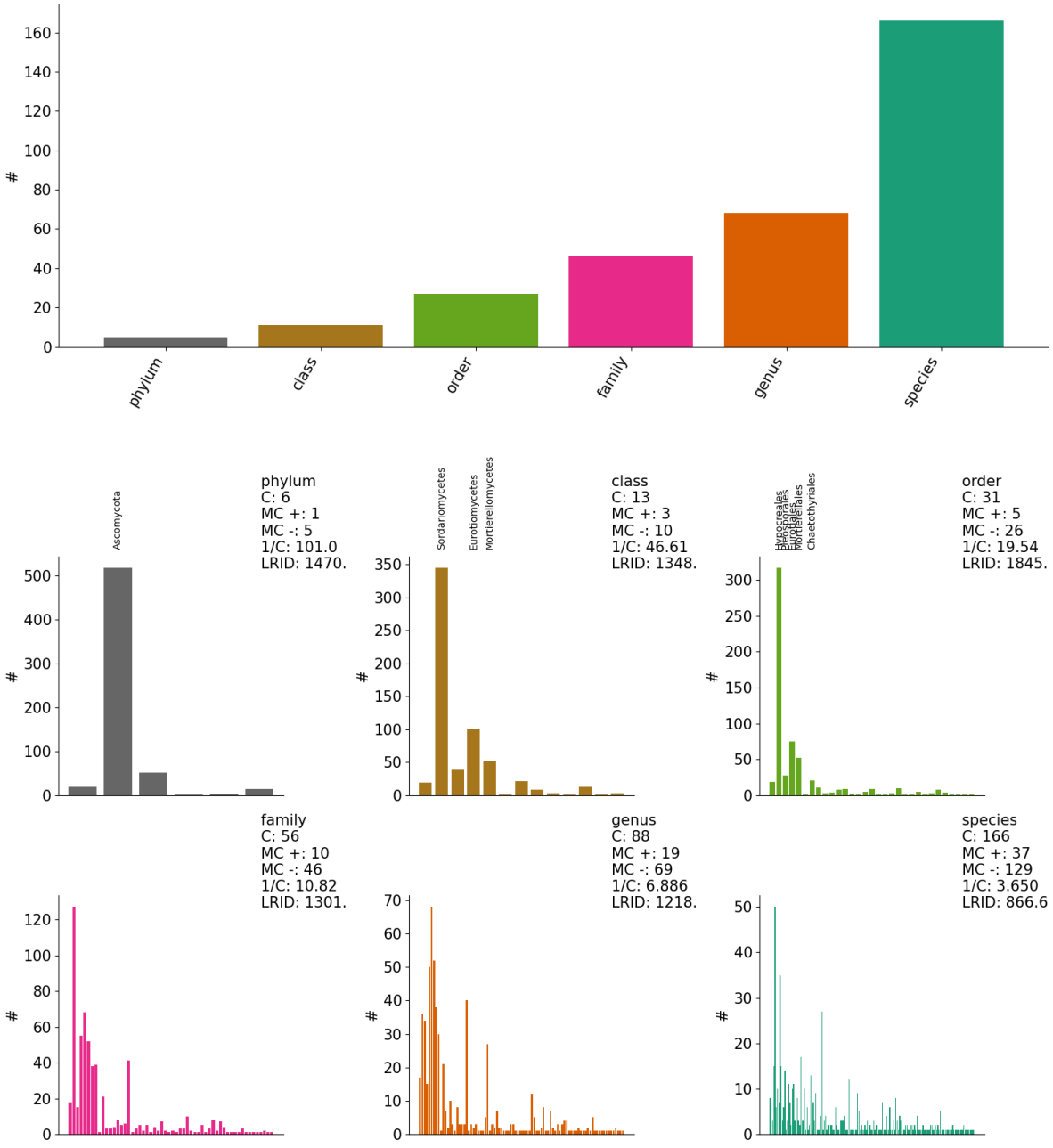


Figure 3: *Top*: Number of unique taxons/classes according to taxonomic rank. *Bottom*: Taxon/Class distribution within each taxonomic rank. C is the number of classes, $MC+$ is the majority class count, MC - is the minority class count. The expected number of uniformly distributed samples is given by $\frac{1}{C}$. *LRID* is the likelihood ratio imbalance degree. For phylum, class and order majority classes are labeled.

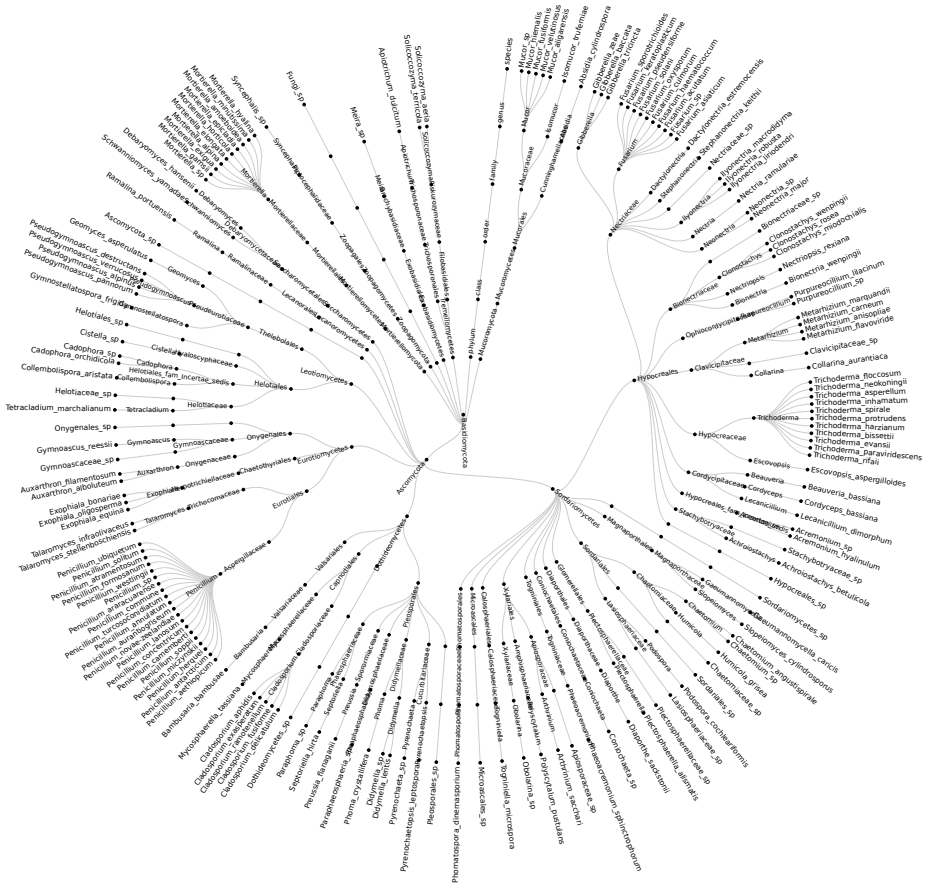


Figure 4: Taxonomic tree chart shows the relations and the diversification of taxonomic classes.

2 Preprocessing

At first the group image is cut, such that each sub image contains one Petri dish scan. The images are resized to 224x224 pixels using pixel area relation, except for *augmented random oversampling* (chpt.2.4.1) to 356x356 pixels .

2.1 Image Separation using Circle Hough Transform

The group images containing up to 12 isolate images, provide a challenge to cut as the exact position withing the image is not fixed and thus unknown. Moreover images comprising only some of the 12 images may drastically differ in terms of grid position.

The only stable characteristics across all images is the fact that Petri dishes are round and the numbering runs from top to bottom and right to left. **Circle Hough Transform** allows us to exploit those characteristics to detect the circles' position. I use the implementation by the *Open CV Project* (ope 2020), which uses the more sophisticated method **Hough Gradient** (Tomislav Petković 2015) .

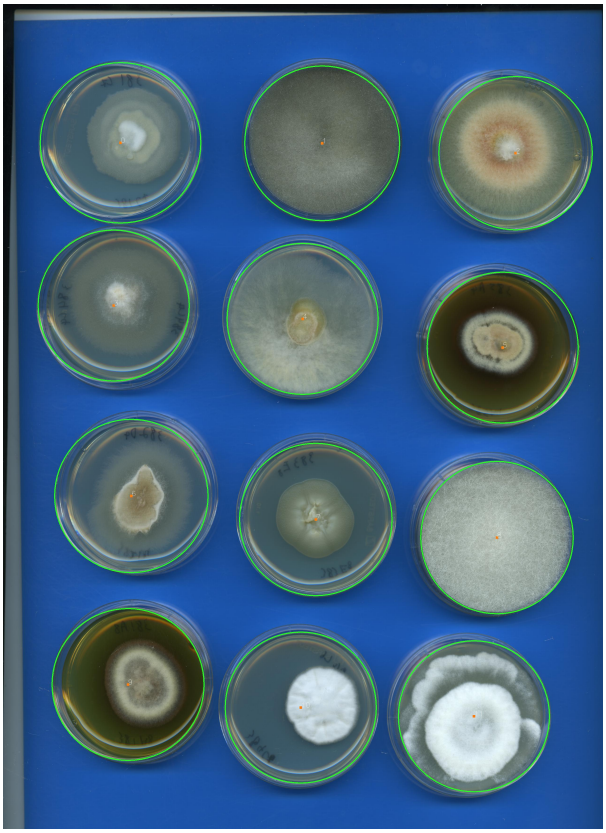


Figure 5: Group image comprising 12 isolates. Petri dish borders are detected using circle Hough Transform and highlighted in green. Row numbering is inverted.

2.2 Building a Unique Identifier over Database and Imageset

Each sample is annotated by a set of six hierarchically nested labels, each representing a taxon given a taxonomic rank, that is phylum > class > order > family > genus > species. As there is no single unique identifier to discriminate samples,

a grouped unique identifier is formed by creation
date of the grouped image scan, file name of that group image and position within the group image.

2.3 Missing Labels

Leaving the missing labels that are listed in table 1 blank or otherwise unimputed, can cause the undesirable side effect of the classifier to learn a "new" class or taxon representation, namely *NaN*. It means that all unclassifiable groups are considered as the same group which is called *NaN*, even though they are taxonomically different from each other. Whereas three missing values over an average samples per class count of 120 for phylum does not introduce any kind of a strong bias, this dramatically changes for higher taxonomic ranks and especially for genus. In that manner the influence on learning of this fake class increases. Hence it is important to introduce a technique to impute missing values. The task of assigning true values however is not trivial. For instance a culture might simply not be identified or identifiable to a certain taxonomic rank. The simple but sufficiently good approach that I apply is to take the taxon of the lower rank and add a suffix indicating the current rank. In contrast to no imputation or imputation with a single term this method has the advantage of separating lineages where otherwise a false dependence might be introduced.

2.4 Class Imbalance

Class imbalance is the unequal distribution of the classes and causes problems for the classification task. We call a taxon within a taxonomic rank a **class**. While class imbalance is observable by comparing class frequencies to the average frequency of uniformly distributed classes, it is advantageous to have a single value metric. Let's formulate the problem. Given data vector $x \in \mathbb{R}^{p1}$ and its label y, a generative classification model learns the joint distribution:

$$p(x, y) = p(y)p(x|y), \tag{1}$$

with $p(y)$ being the prior knowledge on the probability of label y. For C classes there are C possible outcomes of y: $y = [y_1, y_2, ..., y_C]$. Each outcome y_i is associated with a probability p_c, s.t. $\sum_{c=1}^{C} p_c = 1$. Thus the frequencies of the possible labels, $n = [n_1, n_2, ..., n_C]$, can be modelled using a multinomial distribution, $Multinomial(N, p)$, with parameters N and $p = [p_1, p_2, ..., p_C]$. Given

a dataset $\{x_i^C | i = 1, ..., n_C, c = 1, ..., C\}$ we can derive $N = \sum_{c=1}^{C} n_c$. We can estimate p as $\hat{p}$ as the fraction of the number of observations in the c-th class $\hat{p_c} = \frac{n_c}{N}$. For an exactly balanced dataset $p_c = \frac{1}{C}$, $\forall c = 1, ..., C$. Let's denote $b = [\frac{1}{C}, \frac{1}{C}, ..., \frac{1}{C}]$ as the class distribution vector for exactly balanced data. For imbalanced data, the class with $\hat{p_c} \geq \frac{1}{C}$ is defined as the **majority class** while that with $\hat{p_c} < \frac{1}{C}$ is defined as the **minority class**.

For a binary classification problem *imbalance ratio (IR)* is used as a metric to measure class imbalance in a single value. It is defined as $IR = \frac{\hat{p}_{min}}{\hat{p}_{max}}$, with $\hat{p}_{min}$ and $\hat{p}_{max}$ being the maximum and minimum values in $\hat{p}$. As a multi-class metric (Ortigosa-Hernández, Inza, and Lozano 2017) suggest *imbalance degree (ID)*, which is however extremely prone to the concise usage of a distance measure, that is utilized to measure the deviation from a best possible distribution. To solve this issue (Zhu, Wang, Ma, Wang, and Xue 2018) suggest to use **likelihood ratio imbalance degree (LRID)** to measure the class-imbalance extent of multi-class data. The LR test statistic is used to test whether p can be well fitted by b, i.e. the balanced class distribution. Thus we test $H0 : p = b$ against $H1 : p = \hat{p}$. The LR test statistic is

$$-2ln\frac{L(b|n)}{L(\hat{p}|n)}, \tag{2}$$

with L being the likelihood function. Thus for balanced data, $L(b|n) = L(\hat{p}|n)$ and the value of the test statistic is 0; while for imbalanced data, $L(b|n) < L(\hat{p}|n)$ and the value of the test statistic is larger than 0. When $\hat{p} = \frac{n_c}{N}$, LRID can be written as,

$$LRID = -2\sum_{c=1}^{C} n_c ln\frac{b_c}{\hat{p_c}} = -2\sum_{c=1}^{C} n_c ln\frac{N}{Cn_c} \tag{3}$$

Figure 3 (*Bottom*) shows the class distribution according to taxonomic rank and their LRID values. Throughout the taxonomic ranks we can observe the prevalence of a few **majority classes** and many **minority classes**. Decreasing values of LRID with lower taxonomic ranks suggest that **class imbalance** decreases yet still persists. It is problematic in the sense that it introduces a strong bias for learning algorithms, such that a learner may learn the representation of the dominant class but not the one of the minor classes. Hence it is important to tackle the problem thoroughly. I engage class imbalance on the level of training with the utilization of different random oversampling techniques. It is however also important to address it on a metrics level to assess model performance,

as the most common metrics such as accuracy and F-score are prone to class imbalance too. It means that, for instance, a high accuracy can be readily achieved for a highly imbalanced dataset if the trained model always predict the most abundant category, ignoring the problem of inaccurate prediction for less abundant categories.

2.4.1 Three Datasets on Training Level

Original Data

I utilize three datasets for training the models. First the original data but additionally, as such are few and imbalanced, two from that original set artificially generated datasets.

Naive Random Oversampling

One approach to class imbalance is to use under- and oversampling to equalize the class distributions, (Batista, Bazzan, and Monard 2003). In undersampling, samples are randomly drawn from the original set down to a lower limit. Oversampling is it's counterpart, where samples are randomly drawn and re-added to the set, up until a higher limit. Undersampling is clearly not the approach of choice due to the already small size of the dataset. The problem would almost become a *one-shot learning problem*, which is increasingly difficult to solve. In naive random oversampling the distribution is balanced out by adding slightly altered samples to the set, that are drawn from the minority classes. Those selected samples have their axes randomly flipped and their lighting adjusted.

Augmented Random Oversampling

Augmented random oversampling also applies naive random oversampling but with addition of two image augmentations. For this technique the images were initially cut to 356 x 356, so that we can now take a **random crop** of size 224x224. A second augmentation is added by using a color jitter with respect to brightness, contrast and saturation.

2.4.2 Matthews Correlation Coefficient (MCC) - a Robust Performance Metric

Working with imbalanced data requires a metric that is robust to class imbalance. Such a metric is the Matthews correlation coefficient, which is "derived from a discrete solution to the Pearson

correlation coefficient" (mxn 2020). A more detailed description of MCC is provided in section 3.3.

2.5 Data Provision

The experiments require different data provision. Model performance testing is done on data that is simply split into a **train** and **test part** according to a ratio of 70% for training data and 30% for test data. The stability of a model is tested using **5-fold cross-validation**. The data is split into five parts of equal size. In five iterations, in an alternating order one such part is denoted as test data while the others accumulate to the training data. Although typically image classification studies do not employ cross-validation to estimate the stability of learning, we consider this technique is important for small datasets.

Part III

Theory and Experimental Setup

3 Basics for Classification with Learning Algorithms

In classification we task a computer program to specify to which of k categories an input belongs to. We speak of **binary classification** if we want the algorithm to discriminate whether or not the input belongs to $k = 1$ category, and of **multiclass classification** for $k > 1$ categories respectively. Usually the input is described by a label which indicates if that input belongs to one (of the) class(es). Having one label for each sample is called a **single-label** problem, while if we have more than one we call it a **multi-label** one.
We can train a learning algorithm to do this classification task by learning it from examples. The particular mathematical implementation we call a model. In **supervised learning** we provide labeled data, the so called **training set** for instance comprising of images each annotated by a taxon given a taxonomic rank, so that the algorithm may learn a *discriminant function* to discern between classes. Minimizing a **cost function** drives the algorithm's learning ability, as it tries to keep the error between prediction and observation small. Having a trained model we can than make predictions on a **test set** to assess the model's classification error using a **classification score** such as MCC (chpt.3.3).

3.1 Maximum Likelihood Estimators to derive a Cost Function

The **cost function** (see chpt.4.2.1) is one key element that drives learning forward. It is decisive not to just randomly pick and evaluate such function, but to follow a principle. **Maximum Likelihood Estimators** are such a principle from which we can derive specific functions that are good estimators for different models. "Consider a set of m examples $x = x_1, ..., x_m$ drawn independently from an unknown data generating distribution $p_{data}(x)$. Let $p_{model}(x; \theta)$ be a parametric family of probability distributions over the same space indexed by θ."(Goodfellow, Bengio, and Courville 2016, chpt.5.5)

The Maximum Likelihood Estimator is than defined as

$$\begin{aligned}\theta_{ML} &= \arg\max_{\theta} p_{model}(X, \theta) \\ &= \arg\max_{\theta} \prod_{i}^{m} p_{model}(x_i, \theta)\end{aligned} \tag{4}$$

Since this product over many probabilities might lead to numerical underflow, rather than using eq. 4 we can use the equivalent but more convenient optimization problem

$$\theta_{ML} = \arg\max_{\theta} \sum_{i} log p_{model}(x_i, \theta). \tag{5}$$

to design a cost function, see chpt.4.2.1 for specifics.

3.2 Stochastic Gradient Descent (SGD) with Momentum

Deep *learning* involves optimization, which is the task of minimizing or maximizing some function $f(x)$, which we call **objective function**. We want to minimize a function that penalizes the difference between prediction and observation, which we derive from maximum likehood estimators (chpt.3.1) and that are explained in more detail in chpt.4.2.1. This function we call **cost function**, **loss function** or **error function**. In this work I use those terms interchangeably. Moreover let's denote the value that minimizes a function with a superscript *, such that $x^* = arg\,min f(x)$.
$f(x)$ is smaller than $f(x - \epsilon \nabla_x f(x))$ for small enough ϵ, with $\nabla_x f(x)$ being the gradient of f and ϵ being the **learning rate**. We can decrease f by moving in the direction of the negative gradient. This is known as the **method of steepest descent** or **gradient descent**, (Goodfellow, Bengio, and Courville 2016, chpt.4.3). Following the direction of the negative gradient minimizes our cost function.

Following equation 5 we can write the negative conditional log-likelihood of the training data of size m as

$$J(\theta) = \mathbb{E}_{x,y\sim\hat{p}} L(f(x;\theta), y) = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta), y^{(i)}) \tag{6}$$

where L is the per-example loss $L(x, y, \theta) = -logp(y|x, \theta)$ (see chpt.4.2.1).
For these additive loss functions gradient descent requires computing

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(f(x^{(i)};\theta), y^{(i)}) \tag{7}$$

A problem arises however, as the computational costs of gradient descent are prohibitively high, (Goodfellow, Bengio, and Courville 2016, chpt.5.9). Hence in **stochastic gradient descent** the gradient is an expectation. The expectation may be approximately estimated using a small set of samples. For such a minibatch of examples $\mathbb{B} = \{x^1, ..., x^{m'}\}$ drawn uniformely from the training set, the estimate of the gradient is formed as

$$g = \frac{1}{m'}\nabla_\theta \sum_{i=1}^{m'} L(f(x^{(i)};\theta), y^{(i)}). \tag{8}$$

To accelerate learning, especially in the face of high curvature, small but consistent gradients or noisy gradients (Polyak 1964) introduces the method of **momentum**. The momentum algorithm introduces a variable v that plays the role of velocity, the speed and direction at which the parameters move through parameter space. It is set to an exponentially decaying average of the negative gradient. A hyperparameter $\alpha \in [0, 1)$ determines how quickly the contribution of previous gradients exponentially decay. The update rule is given by:

$$v \leftarrow \alpha v - \epsilon g \tag{9}$$

$$\theta \leftarrow \theta + v. \tag{10}$$

The step size depends on how large and how aligned a sequence of gradients are. The models I built are trained with $\alpha = 0.8$.

3.3 Performance Metrics

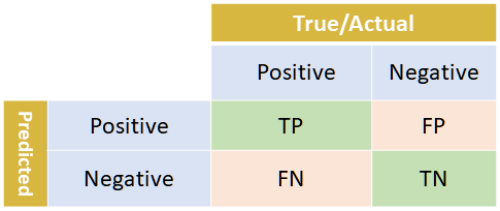


Figure 6: Confusion Matrix for binary classification. (Illustration by (Shmueli 2019))

Once we have trained a model we can evaluate it's classification performance. Let's first define the **confusion matrix** for binary classification. In the two-class case - one positive and one negative class - the confusion matrix has four values, corresponding to the four combinations of true and predicted classes (Fig.6a). True positives (TP) are the positive predictions which are in the actual positive class. True negatives (TN) are correctly predicted as negatives, whereas false positives (FP) are wrongly predicted as positives. False negatives (FN) are predicted to belong to the negative class though they belong to the positive one. A multiclass confusion matrix is defined respectively.

Once we have the confusion matrix, we want to compute a single score as performance measure. But which metric to use? Screening the landscape of performance metrics, one quickly finds the holy four, *Accuracy, Precision, Recall and F1-score*. Though being well established metrics, they all come with downsides. All are prone to class imbalance. Accuracy will fail when looking for rare events. Precision, Recall and thus F1-score only use three values of the confusion matrix. The true negatives are not used at all and those three metrics are not symmetric.

3.3.1 Accuracy (ACC)

Accuracy is a well known and established metric. It calculates the number of correct predictions divided by the total number of predictions and accounts for only TP and TN,

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}. \tag{11}$$

3.3.2 Matthews Correlation Coefficient (MCC)

Facing the drawbacks of the well established metrics (Chicco 2020) et.al suggest to use **Matthews Correlation Coefficient** (MCC) instead, as originally defined by (Mathews 1975). It " is a more reliable statistical rate which produces a high score only if the prediction obtained good results in all of the four confusion matrix categories [...], proportionally both to the size of positive elements and the size of negative elements in the dataset"(Chicco 2020).

Having the confusion matrix we can define the Matthews Correlation Coefficient as

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{12}$$

for the binary case. We treat the true and the predicted class as two binary variables and compute their correlation coefficient. The higher the correlation between true and predicted values, the better the prediction. When the classifier is perfect (FN = FP = 0) the value of MCC is 1, indicating perfect positive correlation. The other way around, when the classifier always misclassifies (TP = TN = 0), we get a value of -1, representing perfect negative correlation. With a value of 0 the classifier is no better than random. The Matthews Correlation Coefficient is symmetric, which means that no class is more important than the other. Moreover it takes all four values in the confusion matrix into account. Having a high value means that both classes are predicted well, even if one class is underrepresented. The MCC has nice properties for a performance metric, though until here we have only defined it in the binary case.

(Gorodkin 2004) introduces the R_K score to compare "two K-category assignments by a K-category correlation coefficient". They derive the R_K score as a discrete solution to the Pearson correlation coefficient as multiclass equivalent for the Matthews correlation coefficient. In literature there is no clear name definition and $\mathbf{R_k}$, $\mathbf{PCC_{discrete}}$ and $\mathbf{MCC_{multi-class}}$ are used synonymously. For a kxk category confusion matrix the multiclass Matthews Correlation Coefficient is defined as

$$MCC_{multiclass} = \frac{\sum_k \sum_l \sum_m C_{kk}C_{lm} - C_{kl}C_{mk}}{\sqrt{\sum_k (\sum_l C_{kl})(\sum_{k'|k' \neq k} \sum_{l'} C_{k'l'})} \sqrt{\sum_k (\sum_l C_{lk})(\sum_{k'|k' \neq k} \sum_{l'} C_{l'k'})}} \tag{13}$$

Note that the multiclass form $MCC_{multiclass}$ will no longer range between -1 and +1 as MCC defined

in eq.12. Instead the minimum value will be between -1 and 0 depending on the true distribution. The maximum value is always +1.

3.4 Generalization

Having defined a performance metric we can evaluate the classification performance by calculating the error between label, i.e. actual or true class and prediction (Alpaydin 2014, chpt.2.1). Computing this error on the training set yields the **training error**. The **test error** or **generalization error** measures how well the model predicts on the test set. "The ability to perform well on previously unobserved inputs is called **generalization**"(Goodfellow, Bengio, and Courville 2016, chpt. 5.2). **Overfitting** occurs when the gap between training error and generalization error is too large. It can occur when the hypothesis is overcomplex so that we may not only learn the underlying function but also the noise in the data and may make a bad fit. **Underfitting** occurs when the model is not able to obtain a sufficiently low training error. This means that our degree of model complexity is less than the actual function. One way to control how well a model fits is to control a model's **capacity**, i.e. the complexity of the hypothesis, for instance by adding higher order functions. "Machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with. Models with insufficient capacity are unable to solve complex tasks. Models with high capacity can solve complex tasks, but when their capacity is higher than needed to solve the present task they may overfit."(Goodfellow, Bengio, and Courville 2016, chpt. 5.2)

3.5 Hierarchical Classification

The field of data science has an inherent dissonance: while the human mind perceives the world around it in hierarchical structures, the models we build receive input that is flat. Or in other words, they receive input that is statistically independent of each other, without information clustering. This observation is also true for taxonomic classification. What's the best way to bridge that gap? As is usually the case, there is no one clear answer, but there are a few ways to tackle that challenge.
(Silla and Freitas 2011) suggest several concepts to engage hierarchical classification. Suppose we want to classify a taxonomy of pets as shown in figure 7. The simplest approach would be to use *flat classification*, i.e. to only classify the leaf nodes with no respect to taxonomy at all. Though fairly easy, this approach obviously has a major drawback as it looses a lot of information content in not

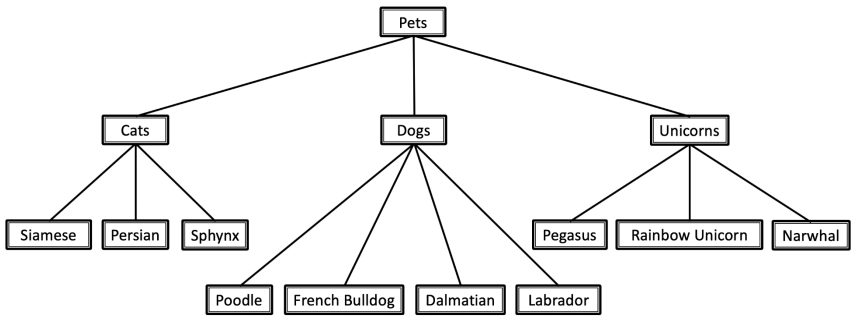


Figure 7: Taxonomy of pets. (Illustration by (Weiss 2019))

modeling the taxonomic hierarchy.
Throughout this chapter I am going to set the foundation I of the three models I am implementing, revisiting each in the successive chapter to add additional traits.

3.5.1 Separate Local Classifiers (SL)

Separated Local Classifiers come in two versions, as *local per-node classifiers* or as **local per-level classifiers**. Following the local node-classifier approach each parent node consists of one separate multiclass classifier, for example one multiclass classifier for "pets", one multiclass classifier for "cats" and so on. But the local node classifiers could also be implemented as separate binary classifiers for every child node, for instance a "cat" classifier, a "Persian" classifier and so forth.
Another approach is to work on each single taxonomic rank as a whole using separate local multiclass level-classifiers, Fig.8. Those model a taxonomic relationship within a given rank. By divide and conquer local node classifiers might additionally address the problem of class imbalance. The wider scoped view of local level-classifiers on the other hand might include additional information for a classification rule.
This **local per-level classifiers (SL)** is the simplest classifier I am implementing. Each taxonomic rank has it's own classifier, i.e. there are six classifiers, one for phylum, one for class, etc.. Information is not passed on between ranks, such that hierarchical information content is lost. Moreover each classifier as it's own separate loss function.

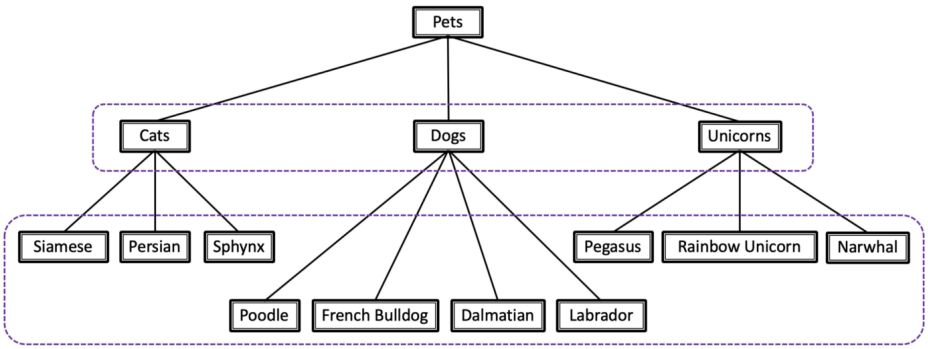


Figure 8: Separated local multiclass classifiers for each taxonomic rank. (Illustration by (Weiss 2019))

3.5.2 Global Classifier - Big Bang Approach (ML)

A global classifier or big-bang approach considers it all at once. There is one "single, relatively complex model, which considers the entire class hierarchy as a whole, during a single run"(Silla and Freitas 2011), Fig.9. Global classifiers can go very different directions. Some may use clustering methods, some are modified version on existing algorithms or as in this work, use a **multi-label multiclass** approach. A global classifier obviously comes with the advantage of having a wide scope, i.e. seeing the hierarchy and possibly learning from its information. Deep neural networks (chpt.4) as classifiers make an excellent choice for learning hierarchies, however their design as black box models do not offer any options for task specification in a generic, pre-designed network. Thus, unless a deep neural network is designed individually to model this very specific task, it's labels remain independent of each other, so that weird predictions such as "Poodle-Unicorns" might be made.

I designed the big bang classifier as a **multi-label multiclass (ML)** model with six output units (see chpt.4.2.1), one for each taxonomic level. However the inner structure of the network remains untouched, i.e. is not specifically designed to this task. Hence a hierarchy might be learned by the network on it's own by the labels provided, but strange combinations might occur. The classifier's learning is driven according to the unweighted summation over each output unit's loss function $\sum_i J_i$, with J_i being the loss in taxonomix rank i.

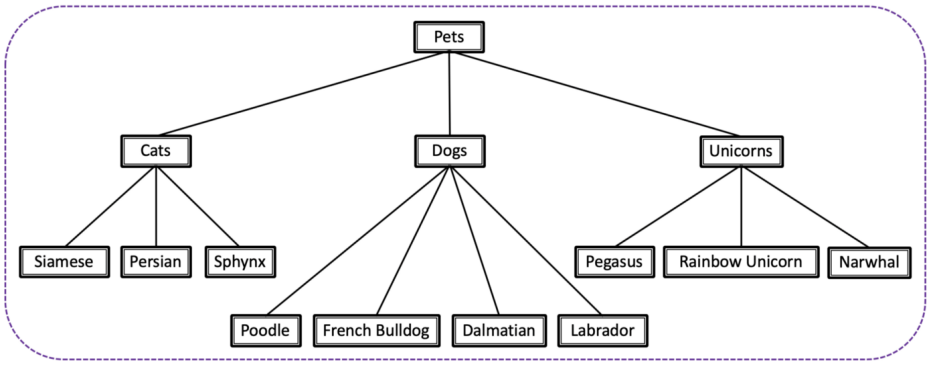


Figure 9: Global Big Bang classifier. (Illustration by (Weiss 2019))

3.5.3 Chained Local Classifiers (HC)

Last but not least local classifiers can be chained. Given a taxonomic rank a deep neural network classifier learns to discriminate taxons and it's learned parameters are passed on to the successive rank classifier, making use of the transfer learning approach described in chpt. 4.5. *Chained local node classifiers* are the go to approach as they make use of the taxonomic hierarchy and avoid misalignments of taxonomic ranks such as "Poodle-Unicorns". However do they pose a challenge on implementation as with increasing number of taxons the number of classifiers increases linearly. Moreover chained local node classifiers require a sufficiently large availability of data for each node to train on, which makes this task infeasible considering class imbalance.
Hence **chained local per-level classifiers (HC)** are an attractive alternative despite allowing for weird combinations. This model consists of six hierarchically stacked classifiers $C_{Phylum} < C_{Class} < C_{Order} < C_{Family} < C_{Genus} < C_{Species}$. Each classifier C has it's own nested loss function J_{Rank} with respect to hierarchy, s.t. $J_{Species} = J_{Species}(J_{Genus}(J_{Family}(J_{Order}(J_{Class}(J_{Phylum})))))$. The implementation of this hierarchical approach using deep transfer learning seems a natural and elegant way to model the hierarchical structure of taxonomy, given the assumption that diversification also expresses in visual traits.

4 Deep Neural Networks

Deep neural networks, also called **feedforward networks** or **multilayer perceptrons** approximate some function f^*. In case of classification $y = f^*(x)$ maps an input x to a category y. A feedforward network defines a mapping $y = f(x;\theta)$ and learns a parameter θ to result in the best function approximation. For taxonomic rank classification x might be image data that are mapped to a taxon y given a taxonomic rank. These networks are called **feedforward** because information flows through this function and in case of the image classifier I am about to define there is no **feedback** connection. They are called networks because they are composed of many different functions.

4.1 Layers

The model is connected with an **directed acyclic graph** describing how the functions are composed together. For example the three functions f^1, f^2 and f^3 might be chained to $f(x) = f^3(f^2(f^1(x)))$, with f^1 being called the **first layer**, f^2 the **second layer**, and so on. The **depth** of the model is defined by this chain. The last layer is called the **output layer**. During training $f(x)$ is driven to match $f^*(x)$. As the output is only shown in the last layer, we call the intermediate layers **hidden layers** and perceive the model as a black box model. These networks are called neural because they are loosely inspired by neuroscience. Each hidden layer is typically vector valued and the dimensionality determines the width of the model. We can think of each layer as consisting of many units that act in parallel, each representing a vector-to-scalar function. Hence each unit resembles a neuron in the sense that it receives input from many other units to compute it's own activation. The idea of using many layers of vector valued representation is drawn from neuroscience. A unit's **activation** is determined by an **activation function** which computes the weighted sum plus bias over all inputs to that unit and transforming it. "They are differentiable operators to transform input signals to outputs, while most of them add non-linearity", (Zhang, Lipton, Li, and Smola 2020, chpt.4.1.2). The most common ones are hyperbolic tangent, sigmoid and last but not least the default standard (leaky) rectified linear unit (ReLU).
A simple multilayer perceptron can model linear regression. Linear regression models perform well if the dataset is linearly separable, they are easy to implement and offer dimensionality reduction, regularization or cross validation techniques to combat overfitting. But their main limitation is the assumption of linearity between the variables. To overcome their limitations and "to extend linear

models to represent nonlinear functions of $\mathbf{x}$, we can apply the linear model not to x itself but to a transformed input $\phi(x)$, where ϕ is a nonlinear transformation", (Goodfellow, Bengio, and Courville 2016, chpt.6.2). Our model is now given by $y = f(x; \theta, \phi) = \phi(x; \theta)^\intercal \omega$. "We now have parameters θ that we use to learn ϕ from a broad class of functions, and parameter ω that map from $\phi(x)$ to the desired output". ϕ defines a hidden layer and we parametrize the representation as $\phi(x; \theta)$ and use an optimization algorithm to find θ that corresponds to a good representation by minimizing a cost function.

4.2 Gradient Based Learning

The introduction of nonlinearity into the network causes the cost function to become non-convex. This means that we train the neural network by usage of an interative, gradient-based optimizer, such as **stochastic gradient descent** (SGD, chpt.3.2) or versions derived from it, that merely drives the cost function to a very low rate, rather than a linear equation solver used to train linear regression models or convex optimization algorithms with global convergence guarantee used to train logistic regression. Stochastic gradient descent applied to non-convex cost functions has no convergence guarantee and is sensitive to **parameter initialization**. Hence it is important to initialize all weights with small random values and the bias with zero or small positive values. For example the initial weights might be sampled from a uniform distribution like $U[-0.07, 0.07]$ or Gaussian distribution with $\sigma = 0.01$. This however can lead to numerical instability due to vanishing or exploding gradients, (Zhang, Lipton, Li, and Smola 2020, chpt.4.8). Addressing this problem I apply **Xavier Initialization** as suggested by (Glorot and Bengio 2010), which samples weights from a Gaussian distribution with zero mean and variance $\sigma^2 = \frac{2}{n_{in}+n_{out}}$, or using normalized initialization with sampling from a uniform distribution $U[-\sqrt{6/(n_{in} + n_{out})}, \sqrt{6/(n_{in} + n_{out})}]$, with n_{in} being the number of inputs and n_{out} being the number of outputs.

4.2.1 Softmax Output Units for Multinoulli Output Distributions

To apply gradient-based learning we must first choose a **cost function**, which choice is tightly coupled to the choice of output unit. As stated in chpt.3.1 we derive the cost function from the negative log likehood to measure the cross-entropy between data distribution and model distribution. As all models in this work are implemented in a multi-class manner the output is expected to follow

a multinoulli distribution.

Let's use the **softmax function** in the output layer to transform the values produced in the 2nd last (hidden) layer so that the sum of the probabilities of all classes can be equal to 1. In other words, that it represents the probability distribution over the discrete output variable with n possible values, i.e. n possible classes. For a discrete variable with n values, we need to produce a vector $\hat{y}$ with $\hat{y}_i = P(y = i \mid x)$. We require not only $\hat{y}_i$ to be between 0 and 1, but also that $\sum_i \hat{y}_i = 1$ to represent a valid probability distribution. First, with focus on the output layer, a linear layer predicts unnormalized log probabilities:

$$z = W^\top h + b, \tag{14}$$

where $z_i = logP(y = i \mid x)$, with W being the weights and b a bias term. As we focus on the last layer, we suppose that the feedforward network provides a set of features defined by $h = f(x; \theta)$. The softmax function is defined as

$$softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)}. \tag{15}$$

Training the softmax to output a target value y using maximum likelihood we want to maximize $logP(y = i \mid z) = log\ softmax(z)_i)$. The **softmax cross-entropy loss function** is then given by

$$log\ softmax(z)_i = z_i - log \sum_{j=1..n} exp(z_j). \tag{16}$$

As stated earlier the three models apply a different amount of output units. *Separate Local per-level Classifiers* utilize one softmax output unit. They operate locally for each taxonomic rank and hence the used output units differ as the amount of possible labels differs between ranks. Being a multi-label multi-class implementation the *Big Bang Classifier* uses six softmax output units, one for each rank. The output units however are independent of each other. There are six *Chained Local per-level Classifiers* that apply one output unit each. Those units are in the sense that the output of the preceding level directly influences the units of this level.

4.2.2 Computational Flow and Backpropagation

In a feedforward network that uses input x to produce output $\hat{y}$, information flows forward through the network. The inputs x provide the initial information that then propagates up to the hidden units at each layer and finally produce $\hat{y}$. This is called **forward propagation**. In **backpropagation** an algorithm allows information to flow backward through the network, in order to compute the gradient of the cost function with respect to the parameters θ, $\nabla_\theta J(\theta)$ (Goodfellow, Bengio, and Courville 2016, chpt.6.5). Backpropagation applies the chain rule of the calculus in a very efficient manner to compute the gradient. Stochastic gradient descent then performs learning using this gradient.

4.3 Regularization

One of the major aspects of training a machine learning model is to reduce the effects of overfitting, i.e. to enable a model not only to perform well on training but also on test data. Such strategies are collectively known as **regularization**. We can define them as any modification made to a learning algorithm intended to reduce it's generalization error. Generally speaking we want to take a model that learned to include the true underlying data generating process but also other possible data generating processes to exclusively match the true data generating process. This can be achieved by adding a parameter norm penalty $\Omega(\theta)$ as regularization term to the objective function, in order to limit the models capacity (Goodfellow, Bengio, and Courville 2016, chpt.7):

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta) \tag{17}$$

where $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term.

One such regularization is **L2 parameter regularization** also known as **weight decay**. It drives the weights closer to the origin, by adding a regularization term $\Omega(\theta) = \frac{1}{2}\|\omega\|_2^2$.

My models train with weight decay $\alpha = 0.01$.

4.4 Convolutional Neural Networks

Convolutional neural networks (CNN) are specialized kind of neural network for processing data with a grid-like topology, like images. In at least one of their layers they employ a mathematical function called **convolution**. Most often they additionally employ an operation called **pooling**.

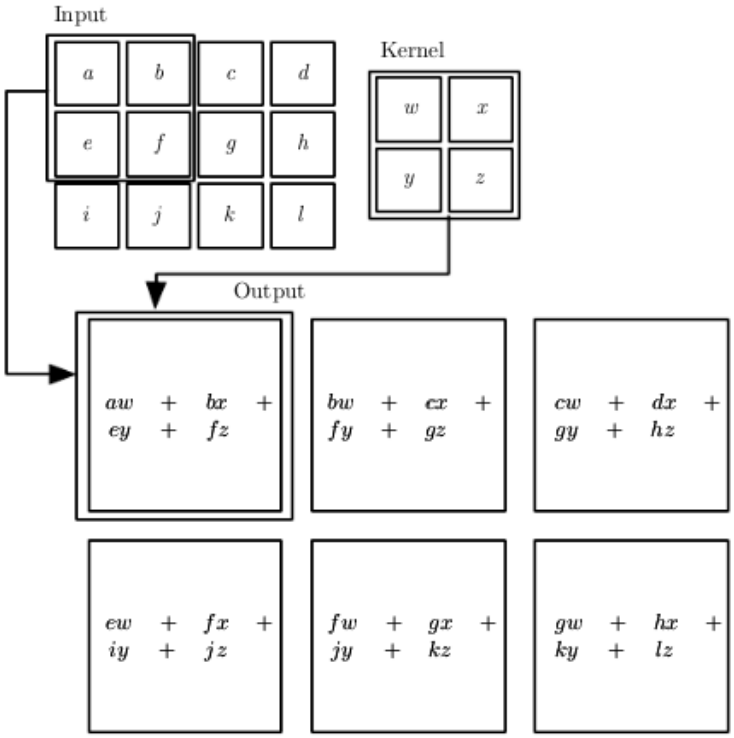


Figure 10: An example of 2-D convolution (Illustration by (Goodfellow, Bengio, and Courville 2016))

4.4.1 Convolution

Let's call an operation on two function a convolution denoted by an asterisk:

$$s(t) = (x * w)(t). \tag{18}$$

We refer to the first argument as **input** and to the second as **kernel**. The output we call a **feature map**. In chapter 9.1 (Goodfellow, Bengio, and Courville 2016) define convolution for an two-dimensional image I and a two-dimensional kernel K as:

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{19}$$

Convolution leverages three important ideas: **sparse interactions**, **parameter sharing** and **equivariant representations**. In contrast to traditional neural networks, where every output unit interacts with every input unit, convolutional networks introduce sparce interactions by making the kernel smaller than the input. This reduces the amount of parameters to store as well as the required operations. Storage is even further reduced by parameter sharing, in which parameters are

no longer learned for every location but rather for a "stage" or block, see figure 11. A function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$, hence convolutional creates a 2-D map of where certain features appear in the input.

4.4.2 Pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For instance **max pooling** reports the maximum output within a rectangular neighborhood. Other pooling functions applied to a rectangular neighborhood are for example L^2 norm or the average taken across that neighborhood. Pooling helps to make the representation approximately **invariant to translations** of the input, which means that most output values do not change for small changes in the input. This could translate to the example that rather than knowing the exact location of an eye in facial recognition we might just want to know the side it is on. "Pooling over spatial regions produces invariance to translation, but if we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to"(Goodfellow, Bengio, and Courville 2016, chpt.9.3).

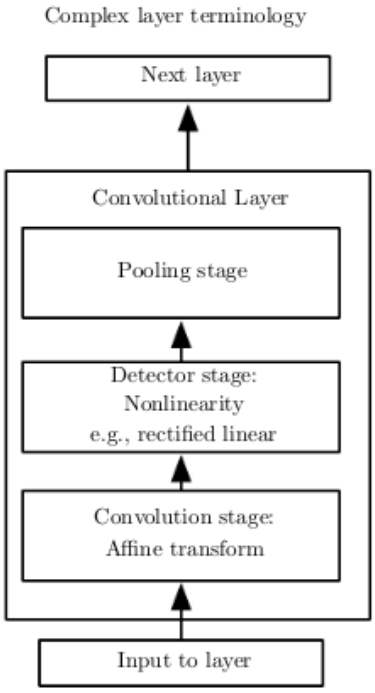


Figure 11: The components of a typical convolutional neural network layer. (Illustration by (Goodfellow, Bengio, and Courville 2016))

4.4.3 Architectures

We can devide convolutional neural networks for image classification in two parts, a sparsely connected feature learning part followed by a fully connected classification part, specified by the output units 4.2.1. A typical layer of the feature learning part consists of three stages, see Fig.11. In the first stage the layer performs several convolutions in parallel to produce a set of linear activations. In the second, also called **detector** stage, the activation is transformed by a nonlinear function such as rectified linear unit (ReLU) or hyperbolic tangent. The third stage is determined by a pooling function.

Densely Connected Convolutional Networks (DenseNet) Since AlexNet's ((Krizhevsky, Sutskever, and Hinton 2012)) breakthrough in image classification in 2012 many convolutional neural network architectures were developed. One such architecture is **DenseNet-169** as proposed by (Huang, Liu, van der Maaten, and Weinberger 2017). DenseNet "connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer — [DenseNet] has L(L+1)/2 direct connections. For each layer, the feature maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameter." Each model applies DenseNet-169 as core unit.

4.5 Transfer Learning

Transfer learning is a form of representation learning in which a learner learns a distribution P_1 over a target domain, which is than used to improve generalization when learning another distribution P_2 over a different yet related domain. Transfer learning makes use of the characteristic that "many visual categories share low-level notions of edges and visual shapes, the effects of geometric changes, changes of lighting, etc.", (Goodfellow, Bengio, and Courville 2016, chpt.15.2). For that transfer learning needs significantly more data in the first distribution that may help to learn representations that are useful to quickly generalize from only very few examples drawn from P_2. The concept of transfer learning however is also applicable when one is not interested in the semantics of the input but in the semantics of the output.

I am going to apply transfer learning to all models because of the limited size of the dataset and for reasons of efficiency. All models use DenseNet-169 pretrained on *ImageNet*. They are **finetuned** to learn this specific classification task in over 20 epochs. Moreover for the chained separate local per-level classifiers I apply transfer learning to pass the parameters learned in a higher taxonomic rank model on to the successive lower rank model.

5 Model Agnostic Explanation with LIME

Explaining why a blackbox model made a prediction is important. An untrustworthy model could make predictions based on the handwritten time stamp on a Petri dish for example. One way to explain deep learning models is to use model agnostic explanations such as LIME. **Local interpretable model-agnostic explanations (LIME)** are local surrogate models that are interpretable and are able to explain individual predictions, (Molnar 2019, chpt.5.7). They are trained to approximate the predictions of the underlying black-box model. Hence they are suited well to offer explanations for deep learning experiments. The key concept works as follows. We have a trained black-box model that we can probe as often as we want with permutations of a selected sample that we want to explain. Lime studies the effects of perturbation of this sample on the prediction. This is done by creating a new dataset consisting of such perturbed data and their predictions. On that new dataset a simpler, easier interpretable model is trained and weighted according to the distance to the original sample. Here LIME utilizes a linear model. The explanation model has local fidelity, i.e. it ought to be a good approximation locally but not globally.
(Ribeiro, Singh, and Guestrin 2016) suggest to express the local surrogate model $\xi(x)$ with interpretability constraint as:

$$\xi(x) = \underset{g \in G}{\arg\min}\, \mathcal{L}(f, g, \pi_x) + \Omega(g). \tag{20}$$

The explanation model for instance x is the model g from all possible explanations G that minimizes a loss function $\mathcal{L}$, such that the error between explanation and prediction of the original model is minimized and the model complexity $\Omega(g)$ is kept low. Model complexity translates as the number of features selected. The proximity measure $\pi_(x)$ defines the size of the neighborhood around instance x.
There are several implementations of LIME for differing data types. They especially differ in terms of the construction of the neighborhood. For image data LIME first applies an external segmentation algorithm, such that image x is segmented into **superpixels**. Superpixel are perceptually clustered areas of an image that carry more information than a pixel. They are the result of image oversegmentation and "are better aligned with intensity edges than a rectangular patch", (Neubert 2015). Image segmentation is carried out by the *quickshift* algorithm, (Vedaldi and Soatto 2008). The parameters $kernelsize = 6$, $max\,distance = 50$ and $ratio = 0.5$ provide best results and are

found by try out. Ratio is a trade-off between distance in color-space and distance in image-space. Once image x is segmented, superpixels can randomly be switched on or off, i.e. grayed out, to construct the perturbed dataset according to the neighborhood size $\pi_x = 1000$ including 100 features. The new images are than weighted according to their proximity to the original image and a weighted linear model is trained. The prediction is than explained by interpreting the local model.

Part IV

Results

The three models built - the Separate Local per-Level Classifiers (SL), Global Classifier - Big Bang Classifier (ML) and Chained Local per-Level Classifiers (HC) - are tested in three experiments. First the individual model's performance is assessed and compared to each other with respect to occurring class imbalance. The model's stability over successive runs is tested in the stability analysis. Lastly the model's performance is explained for test samples with high score using the model agnostic approach LIME.
All models are built in *python* 3.6.9 using Apaches *MXNET gluon* framework with GPU support CUDA-10.1. All deep learning computations are run on *google colab*. LIME 0.2.0.1 is run locally on a workstation in an Ubuntu Bionic Beaver environment.

Figure 12: **Overview chart for Experiments** conducted. Performance Analysis is conducted for all three models the separated local level classifier (SL), big bang classifier (ML) and hierarchical classifier (HC). Models used in Performance Analysis are explained in LIME. Stability is only tested for ML and SL.

6 Performance and Stability Analysis

Each model is trained in 20 epochs of finetuning on the three different datasets, *a)* original data, *b)* naive random oversampled dataset and *c)* transform random oversampled set. For performance analysis data is provided in a train/test split according to a ratio of 7:3. Training proceeded using

Xavier Initialization, a learning rate of 0.001, momentum of 0.8 and weight decay of 0.01. The models are trained and their classification performance on each taxonomic rank is assessed using MCC and accuracy metrics. For performance and stability plots the upper curves show the training score and the lower curves the test score. The upper three sub figures show a model performance according to Matthews Correlation Coefficient (MCC) while the lower sub figures show the accuracy value (ACC) for the same model. For each dataset and classifier type the best models according to the results of MCC on the test set are compared. A confusion matrix shows the model's performance with regard to taxonomic rank, for some selected models.
Stability of separate local level classifiers (SL) and big bang classifiers (ML) is tested using 5-fold cross validation in 20 epochs of finetuning. This experiment is performed on the original data as well as naive and transform random oversampled sets. The same hyperparameters are used as provided for performance analysis. In 5-fold cross validation the data is split into five parts of equal size. In a fold one part is declared test set, while the other parts compose the combined train set. The order of this assignment is changed in successive folds. For each one fold the models are trained and their performance recorded using MCC and accuracy metrics. Across such five trials the mean performance is to be evaluated as well as the standard error of the mean.

6.1 Separate Local per-Level Classifiers (SL)

Figure 13 shows the performance of the Separate Local per-Level Classifiers (SL). On the original dataset the classifier starts off at almost random performance and takes about 15 epochs to saturate on the training data for MCC. Highest accuracy values are achieved for phylum, class and order. Such encouraging classification scores are however misleading as this metric does not include true negatives. More realistic results are provided by the Matthews correlation coefficient. We can observe a classification score that is centered around 0.1, the best values are achieved for class, order and family. Learning on the training set happens a lot faster on the oversampled datasets. The curves are smoother especially for training. The classifier performs slightly better on the naive oversampled data, meaning that the training happens even faster and better results are achieved for phylum, class and order. All models overfit and lack generalization. Figure 14 and 15 show the confusion matrices for phylum, class and order. I only focus on the first three levels as they have the highest prediction score and moreover higher rank confusion matrices become extremely packed. For phylum the majority class is predicted most correctly. However the model fails to generalize

enough to discern other taxons. Hence most other samples are predicted as Ascomycota. But at least four Martierellomycota, one Mucoromycota and two Basidiomycota are predicted correctly. The trend of predicting the most majority class holds on for class as well as for order. Focusing on the majority classes the classifier for class level learns to predict Sordariomycetes well and discerns Eurotiomycetes by around two third, but mostly mis-classifies Mortierellomycetes. In the rank order Hypocrealis is the most dominant taxon, which is predicted well, however the model lacks to generalize to predict other majority classes well. Chaetothyriales and Eurotiales are correctly predicted to some extend.

The stability is shown in figure 16. The curves span the area in which also the curves of the performance plots lie. The standard error of the mean (SEM) becomes very small for the training curves on the oversampled datasets. In contrast to the SEM for the training curves on the original set, which remains high, this will be the result of the increased samples size due to oversampling. The standard error of the mean is bigger for the test scores, for both MCC and accuracy, which underlines again the generalization error.

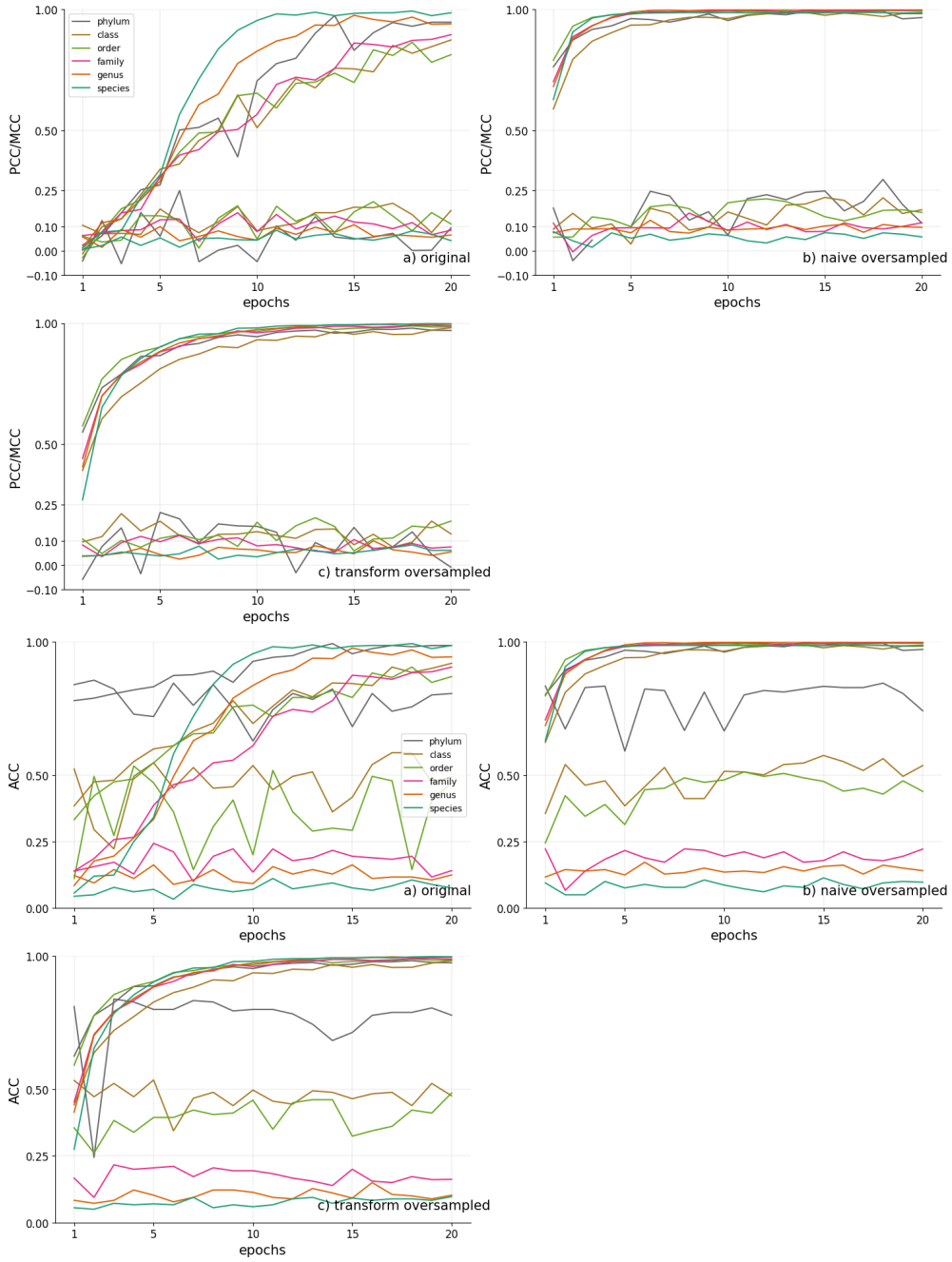


Figure 13: **Performance of Separate Local per-Level Classifier (SL)** finetuned in 20 epochs on *a)* original, *b)* naive oversampled and *c)* transform oversampled dataset. The upper curves are training scores, the lower ones are test scores. The upper three figures show the results using MCC metrics and the lower three accuracy metric for the same model.

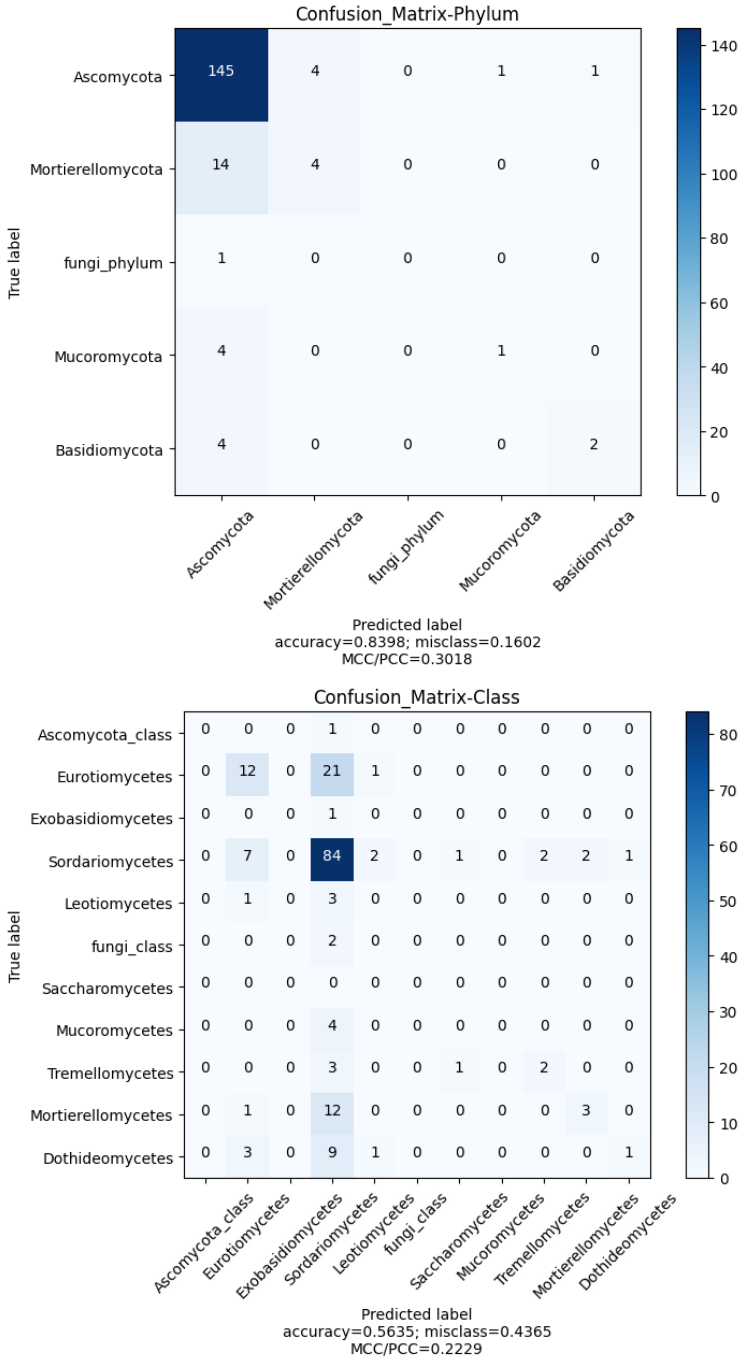


Figure 14: **Confusion matrix** for phylum and class level, prediction on test data of **SL** classifier trained on naive oversampled dataset, epoch $e_p = 17$ and $e_c = 14$.

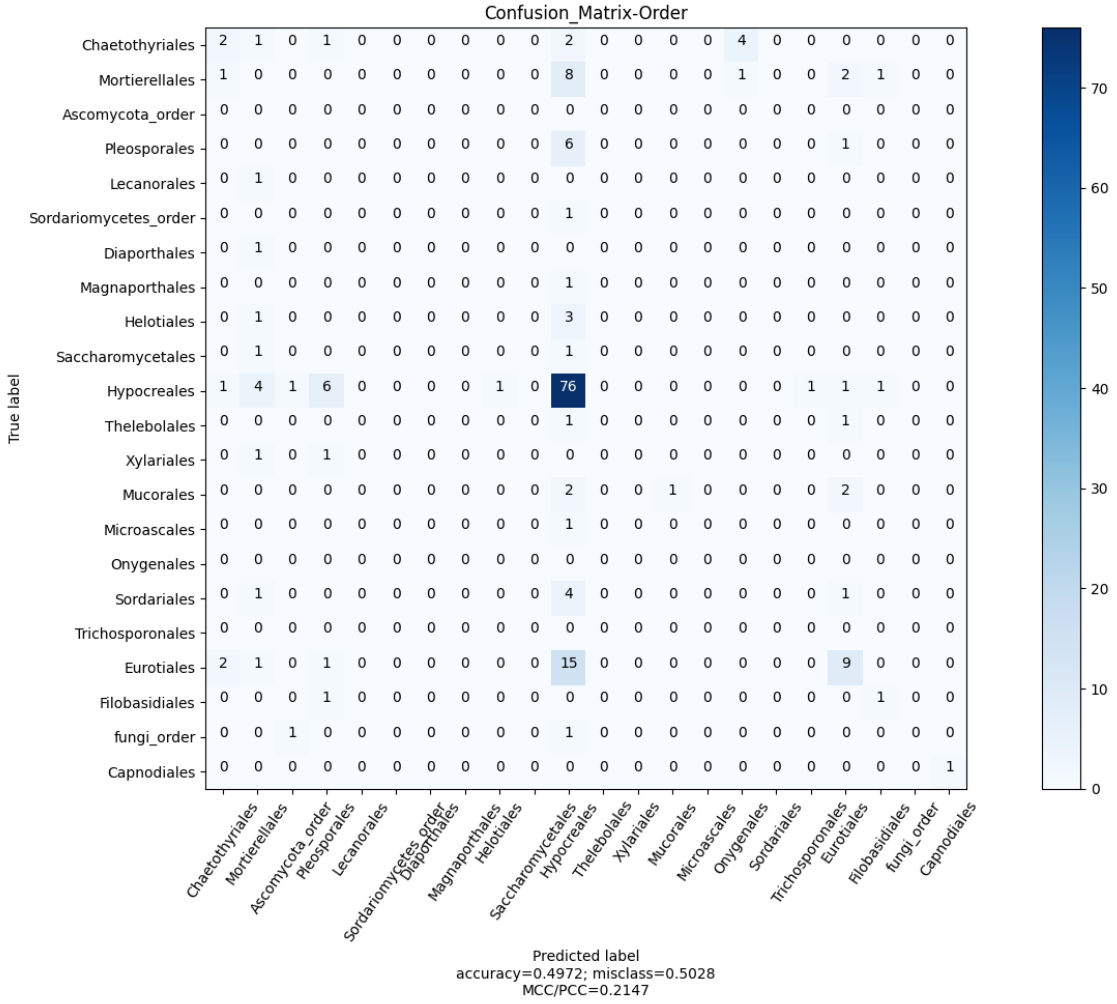


Figure 15: **Confusion matrix** for order level, prediction on test data of **SL** classifier trained on naive oversampled dataset, epoch $e_o = 11$.

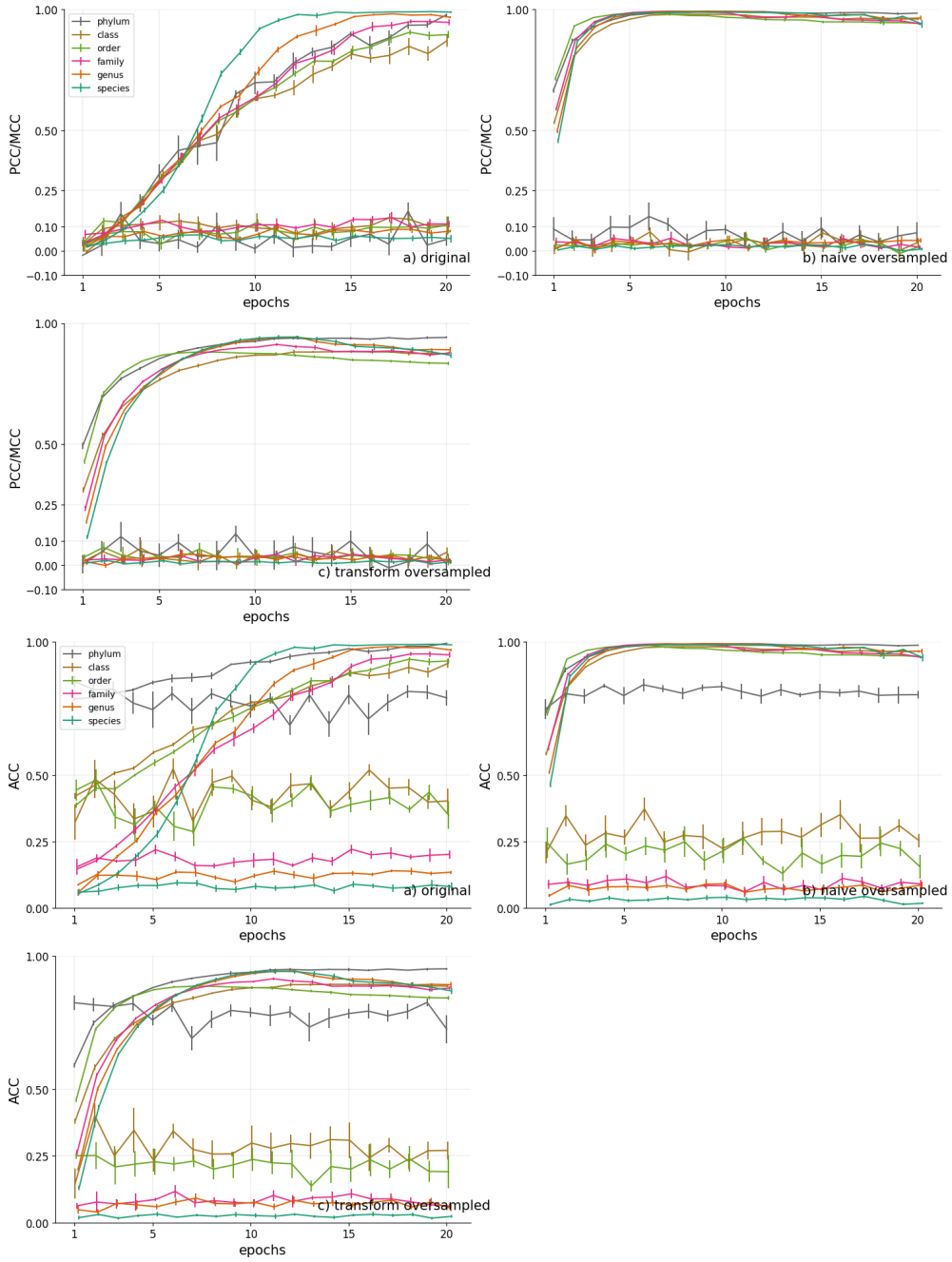


Figure 16: **Stability Plot of Separate Local per-Level Classifier (SL)** finetuned in 20 epochs on *a)* original, *b)* naive oversampled and *c)* transform oversampled dataset using 5-fold crossvalidation. The upper curves are average training scores, the lower ones are average test scores across five folds. The upper three figures show the results using MCC metrics and the lower three accuracy metric for the same model. The error bars are standard error of the mean (SEM).

6.2 Global Classifier - Big Bang Classifier (ML)

The performance of the multi-label, multi-class model is shown in figure 17. Being a multi-label model it learns to make predictions according to all taxonomic ranks at the same time and might possibly learn interactions between the taxonomic ranks. The model performs generally bad on the original data. It trains slowly, it's test scores are erratic and especially in the early stages they include $NaNs$. The model trains faster on the oversampled sets, from which fastest learning happens on the naive oversampled data. The model overfits on all datasets. On the original data it underfits as well. It produces the best test scores on the naive oversampled set for phylum, class and order and even has non-zero test scores for family and genus using accuracy and MCC. It is interesting to observe that phylum MCC is negative for some epochs. That indicates that the features learned to make a prediction based on class or order are inverse for a prediction on phylum. However this happens long after overfitting occurred and could thus also be an effect of overfitting. The observation of negative test scores for phylum is also found in the stability analysis in figure 20. Further investigations into this might yield interesting findings. An explanation behind why features leading to a positive prediction on for example class level leads to an inverse prediction on phylum level could indicate antagonistic phenotypic traits. This however might be difficult to explain.
Figures 18 and 19 show the confusion matrices for phylum, class and order rank. In phylum the influence of the majority class seems even more prevalent, as the model shows more difficulties to discern between the minority classes and more often predicts them as Ascomycota. The same effect holds true for class and order level. Majority classes are predicted to some extend.

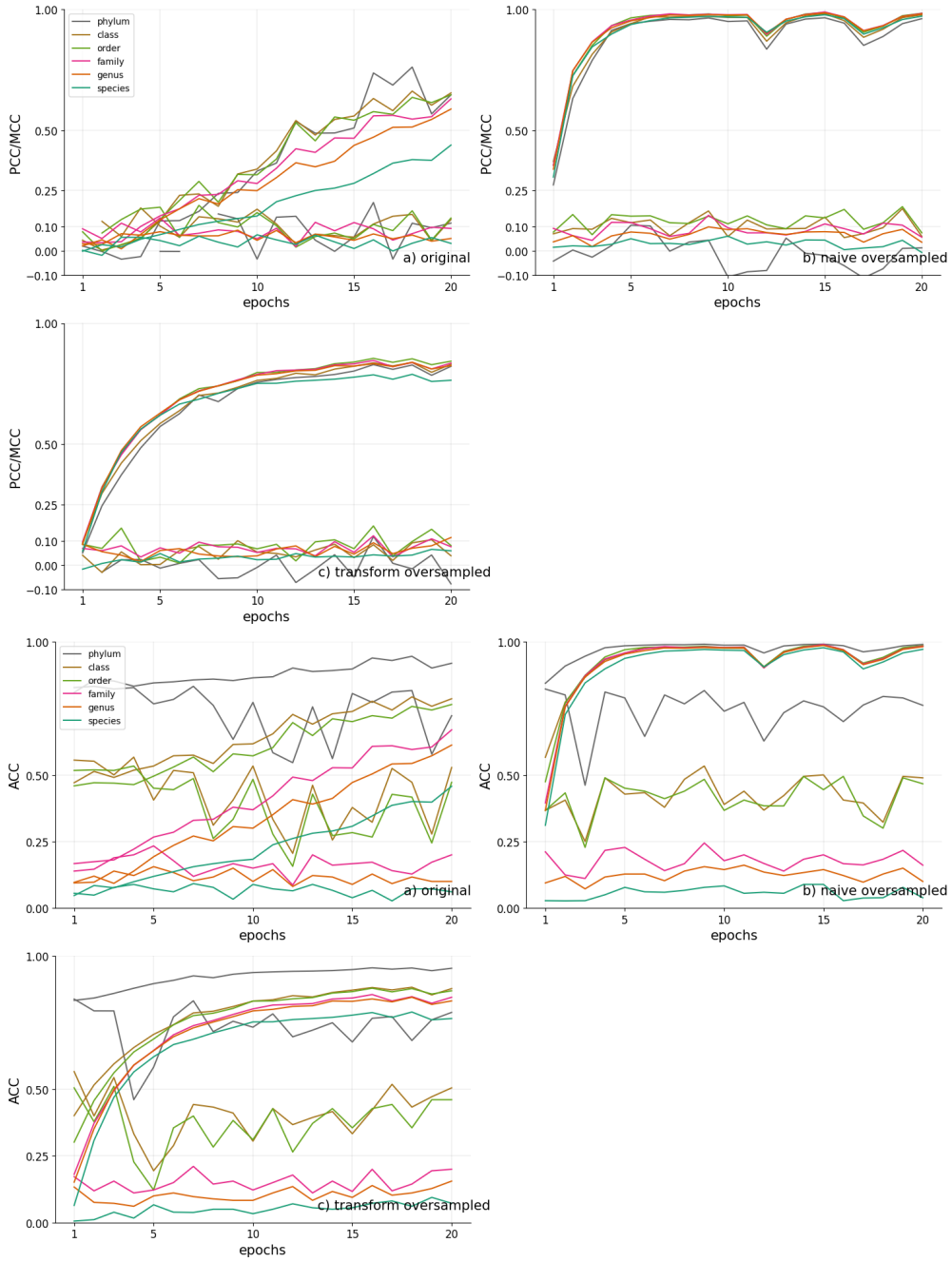


Figure 17: **Performance Plot of Global Classifier - Big Bang Classifier (ML)** finetuned in 20 epochs on *a)* original, *b)* naive oversampled and *c)* transform oversampled dataset. The upper curves are training scores, the lower ones are the test scores. The upper three figures show the results using MCC metrics and the lower three accuracy metric for the same model.

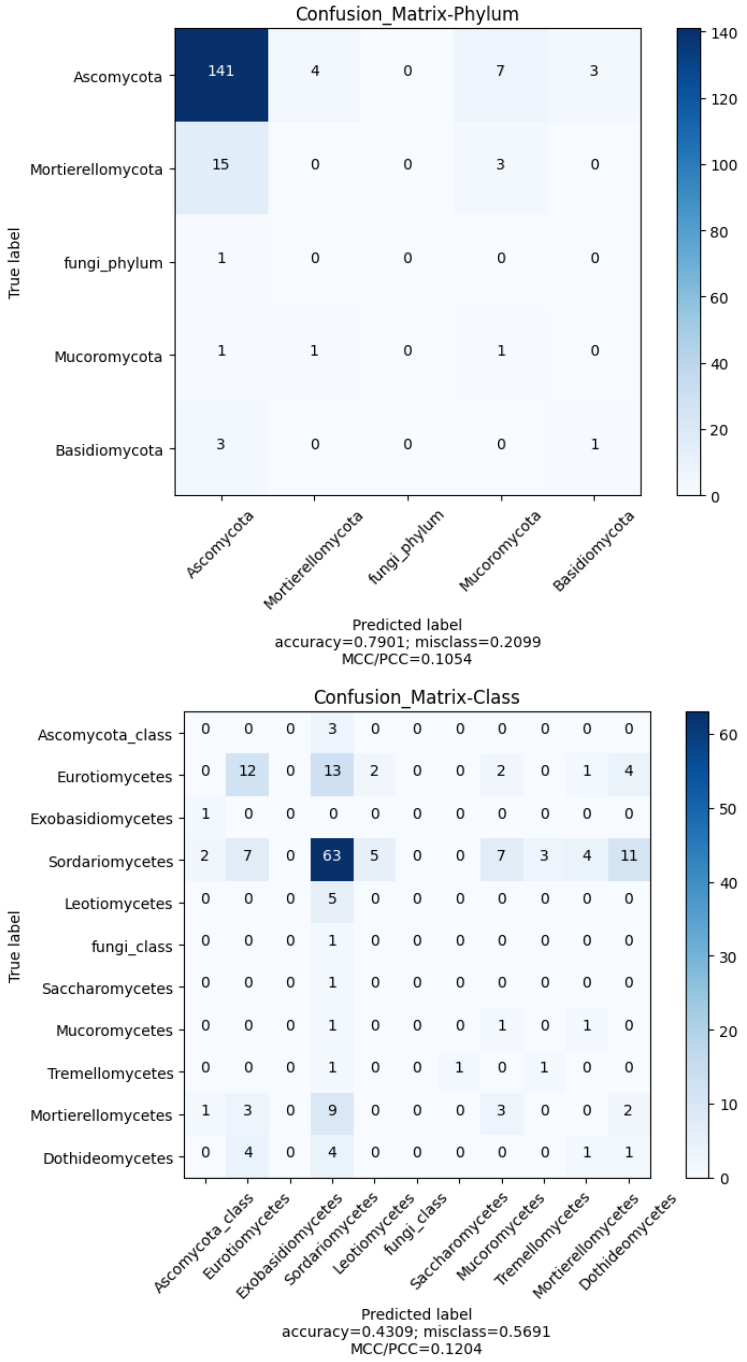


Figure 18: **Confusion matrix** for phylum and class level, prediction on test data of **ML** classifier trained on naive oversampled dataset, epoch $e = 5$

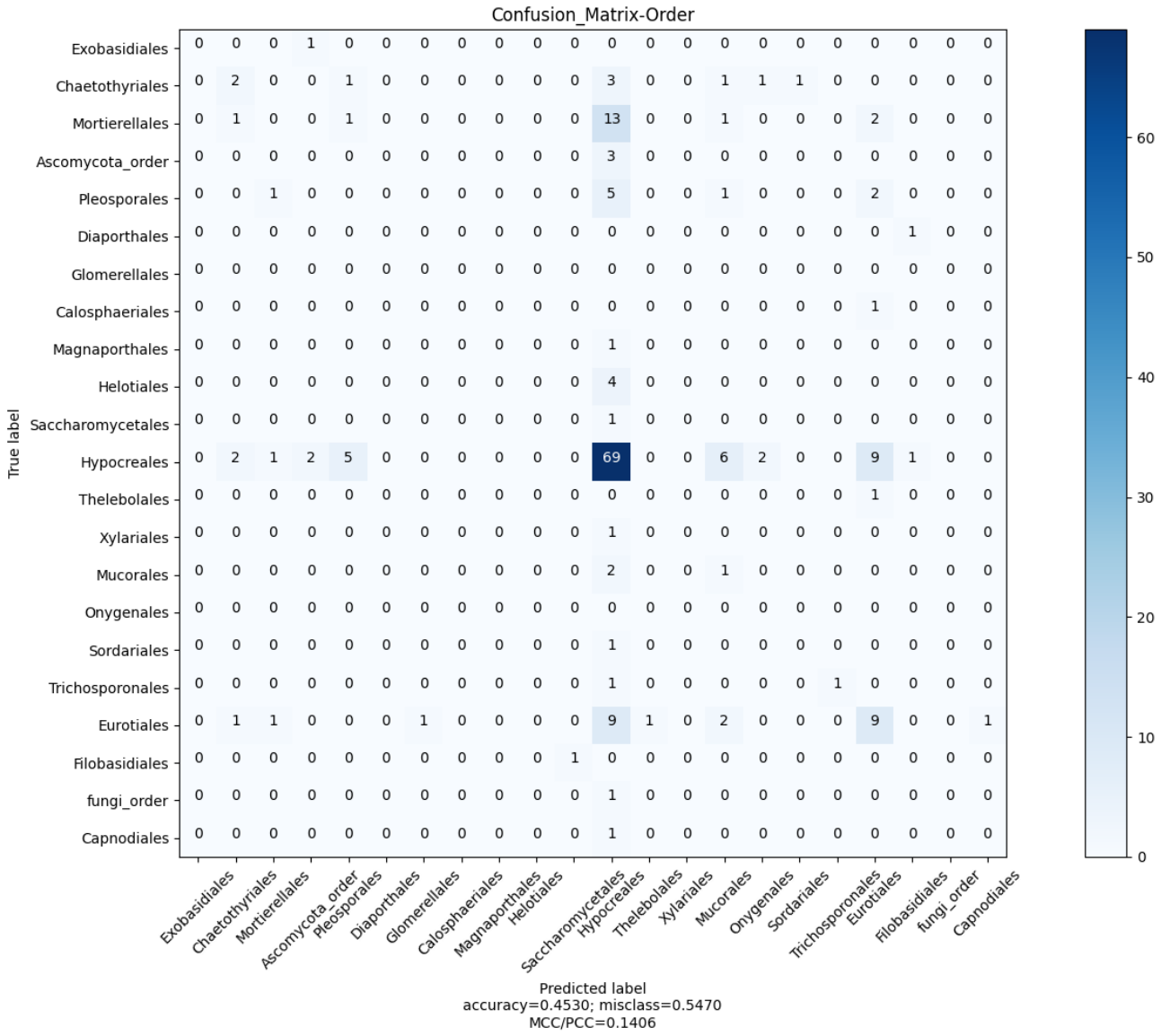


Figure 19: **Confusion matrix** for order level, prediction on test data of **ML** classifier trained on naive oversampled dataset, epoch $e = 5$.

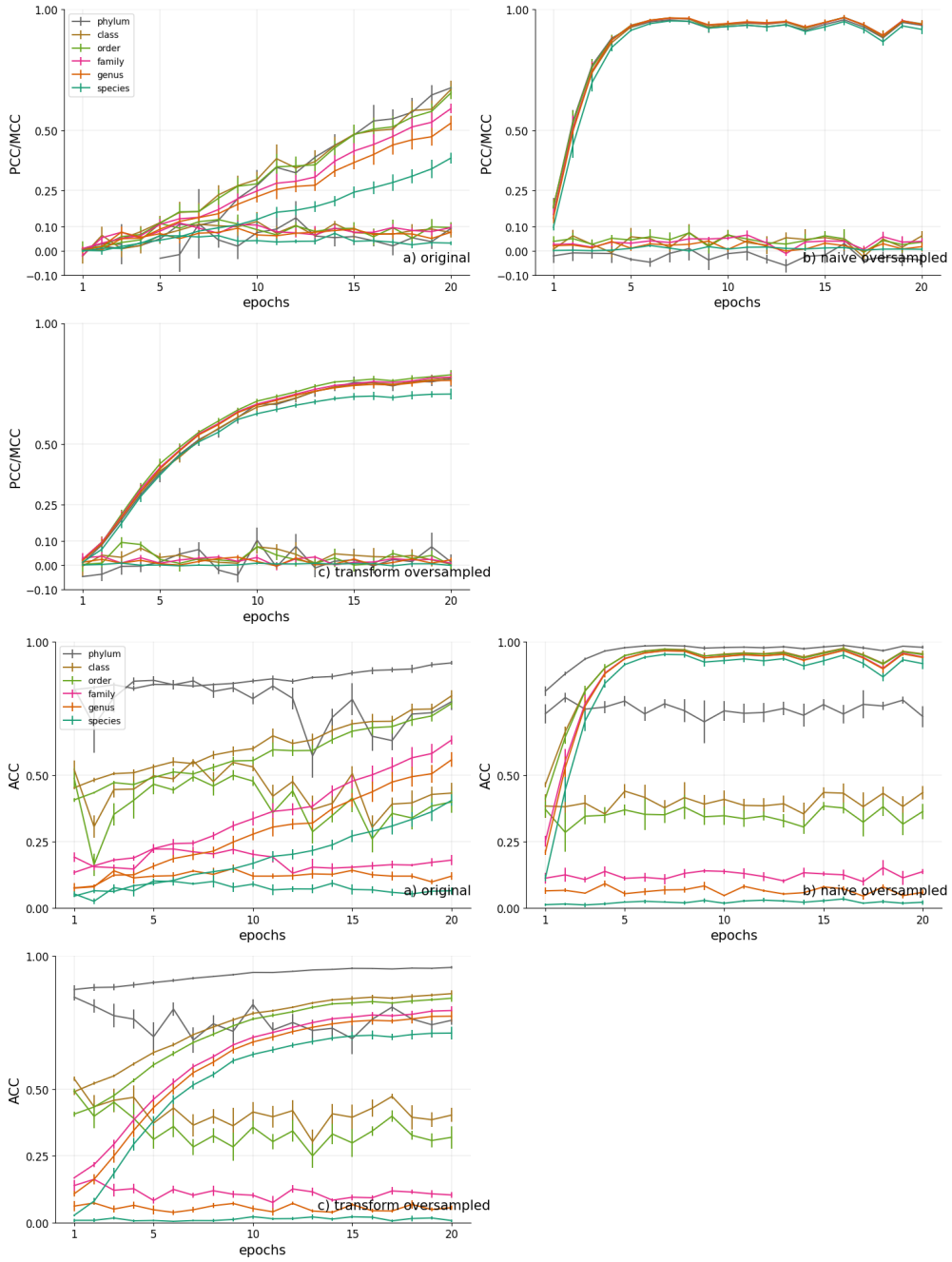


Figure 20: **Stability Plot of Global Classifier - Big Bang Classifier (ML)** finetuned in 20 epochs on *a)* original, *b)* naive oversampled and *c)* transform oversampled dataset in 5-fold crossvalidation. The upper curves are average training scores, the lower are the average test scores across five folds. The upper three figures show the results using MCC metrics and the lower three accuracy metric for the same model.

6.3 Chained Local per-Level Classifiers (HC)

The HC classifier has a slow learning on the original data as well as the transform oversampled data in MCC. Moreover it's train scores remain quite low. Using accuracy the model's performance on all three datasets seems quite similar. Using MCC however reveals that on the two before mentioned datasets the model overfits and underfits at the same time. Underfitting is stronger for lower taxonomic ranks. Especially species training scores underfit, which is also to be found in training on the naive oversampled set.
The test score for phylum computed on the naive oversampled set is lower in comparison to the other models. As the model is a hierarchical one the generalization error is also propagated to successive layers. Having a low test score for phylum, the confusion matrices in figures 22 and 23 show that the model struggles to predict even the majority classes.
A stability test is not provided for this classifier.

6.4 Classifier Comparison

Having a single number makes it easier to compare models. Usually such number is the model's test score of a certain metric just before overfitting. Given the diverse appearance of mycorrhizal fungi and the exploratory nature of this study, I however use highest performance and accept the risk of introducing false positives.
Figure 24 shows the best test scores that each model achieved on each dataset according to MCC and in the lower sub figure the accuracy for the same selection. Best performance on all three datasets was achieved by the separate local per-level classifiers. Though for the SL classifiers each taxonomic rank is connected through lines for better readability it is important to remark that each taxonomic rank has it's own separate classifier. Hence the models will have lower complexity, which will undoubtedly benefit their performance. Lacking this hierarchical information, this classifiers might also not be able to use hierarchical feature connections. The chained local per-level classifiers have a below average performance. Their performance on the original data is quite high though, which can be explained with the erratic prediction behaviour and the usage of maximum values in this figure. Having a look at the confusion matrix reveals that this classifiers performance is portrait overly optimistically. The big bang classifier is found in the middle, with the best performance on the naive oversampled set. Having a look at the average test scores (Fig.25) across epoch $[e_5 : e_{20}]$ confirms those findings.

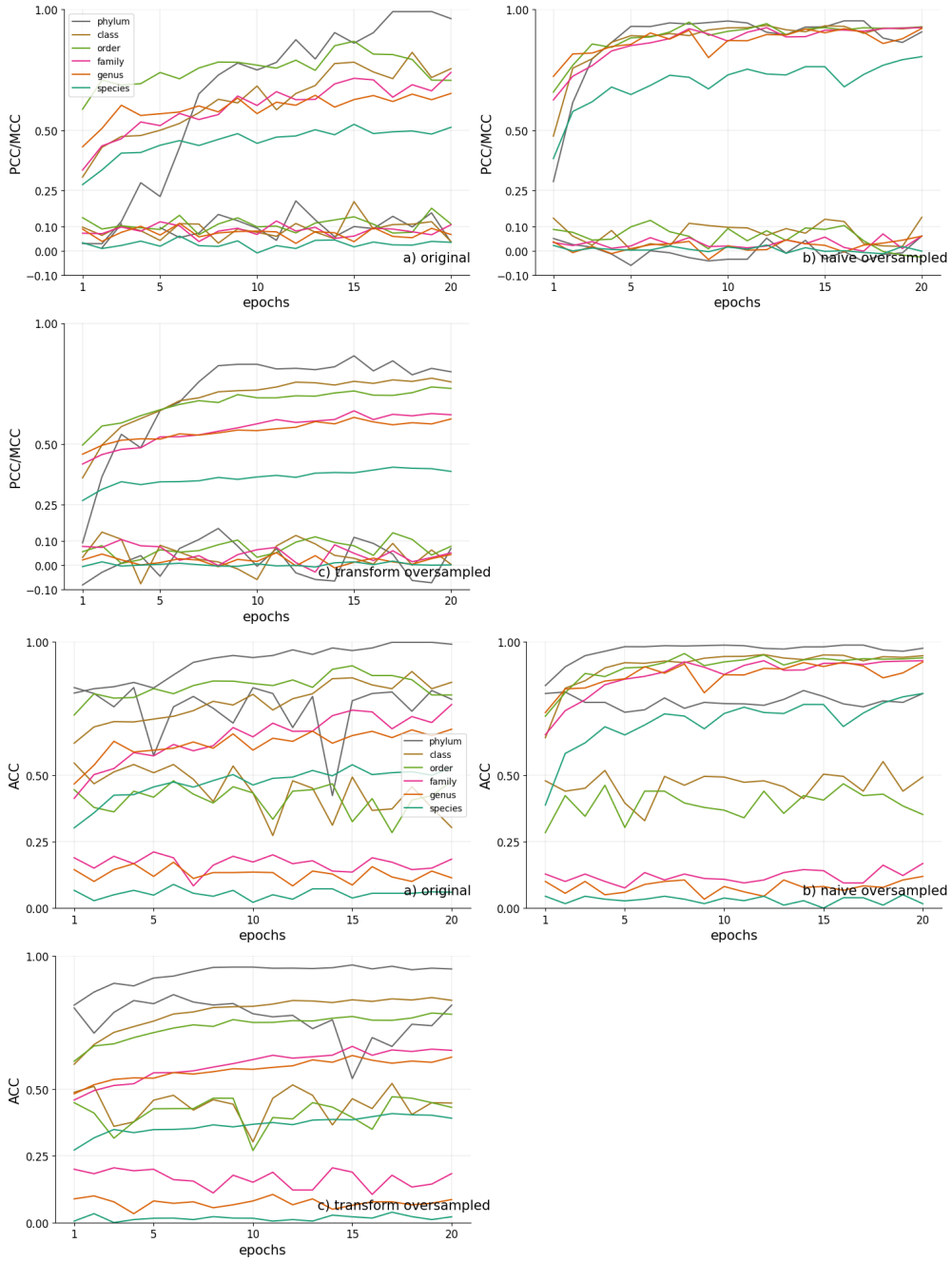


Figure 21: **Performance Plot of Chained Local per-Level Classifiers (HC)** finetuned in 20 epochs on *a)* original, *b)* naive oversampled and *c)* transform oversampled dataset. The upper curves are training scores, the lower test scores. The upper three figures show the results using MCC metrics and the lower three accuracy metric for the same model.

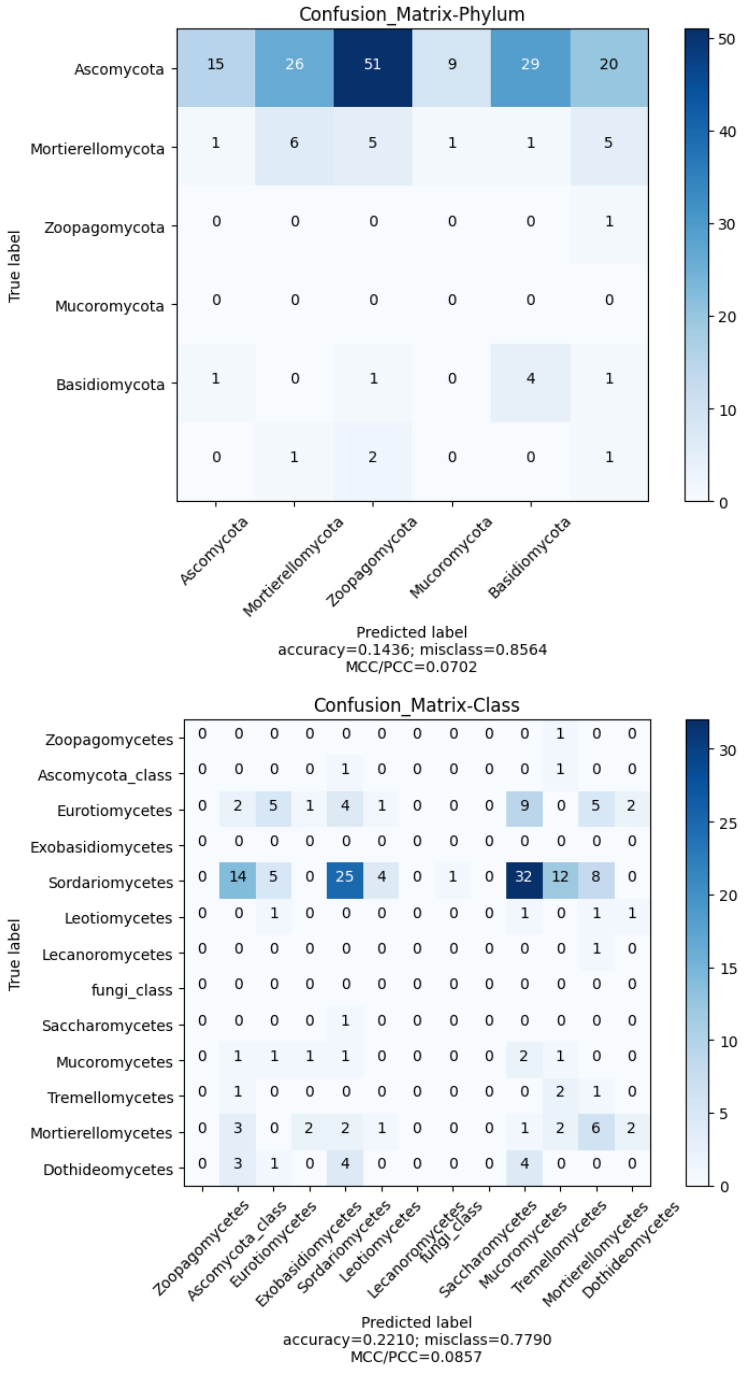


Figure 22: **Confusion matrix** for phylum and class level, prediction on test data of **HC** classifier trained on naive oversampled dataset, epoch $e_p = 20$ and $e_c = 20$

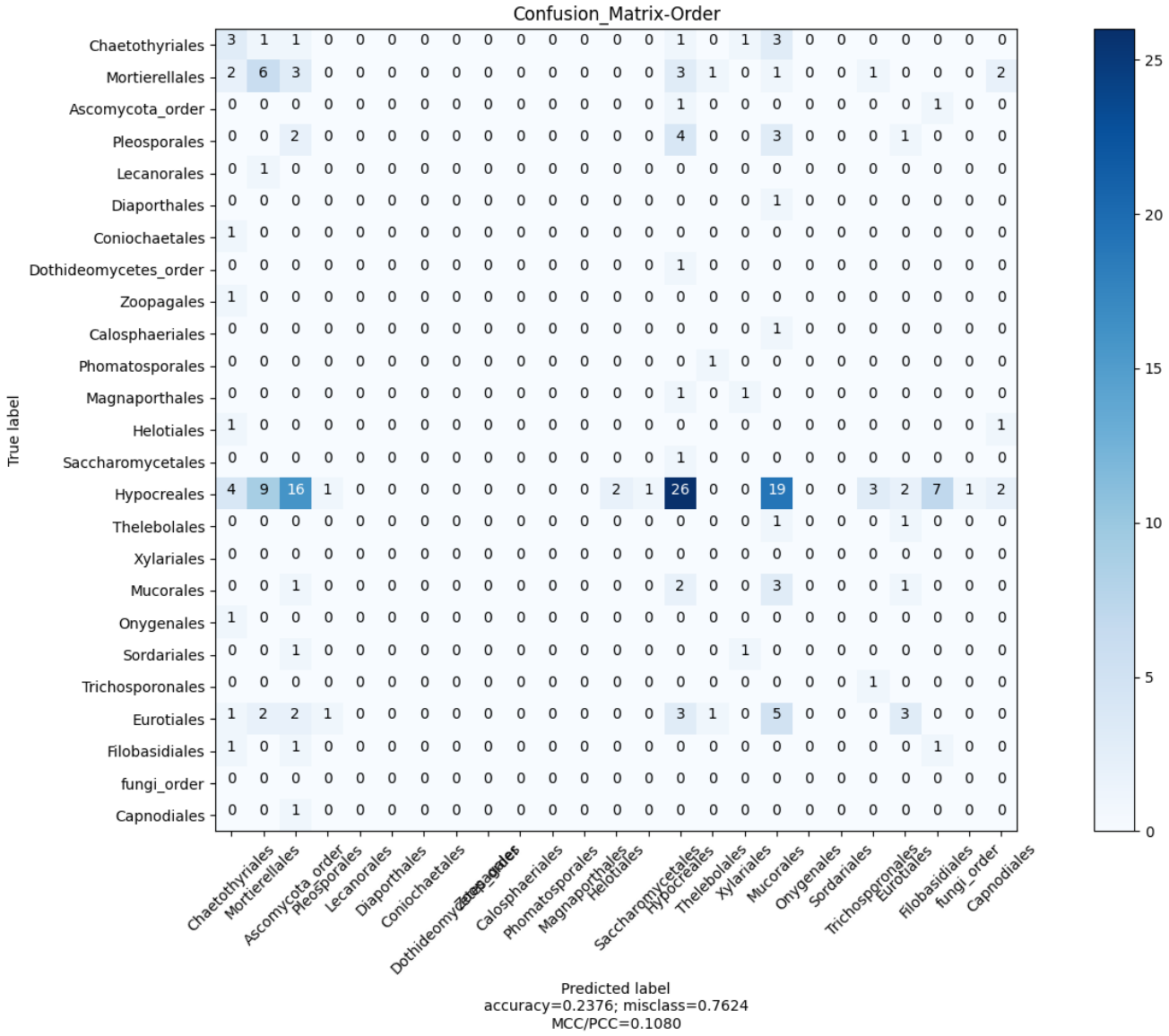


Figure 23: **Confusion matrix** for order level, prediction on test data of **HC** classifier trained on naive oversampled dataset, epoch $e_o = 6$.

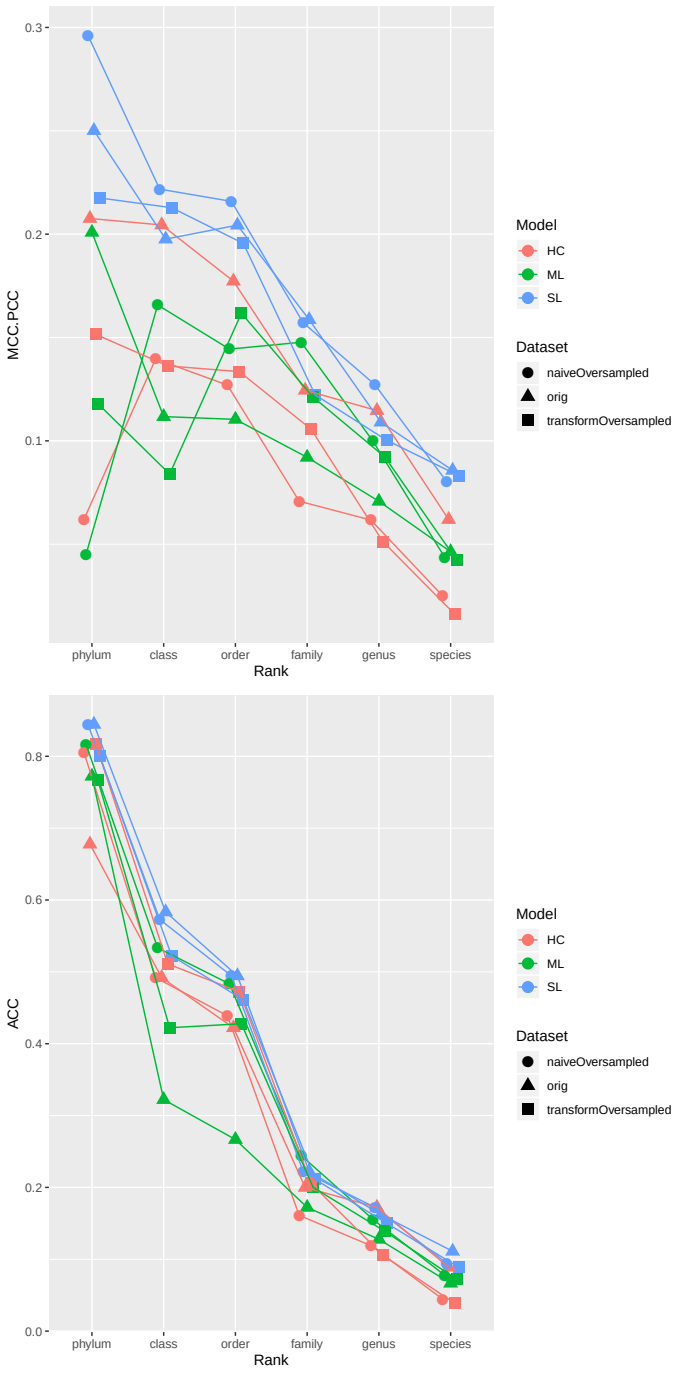


Figure 24: **Comparison Plot** shows **best test scores** that each classifier achieved according to MCC on each dataset. The lower shows the performance of that same classifier using accuracy metrics.

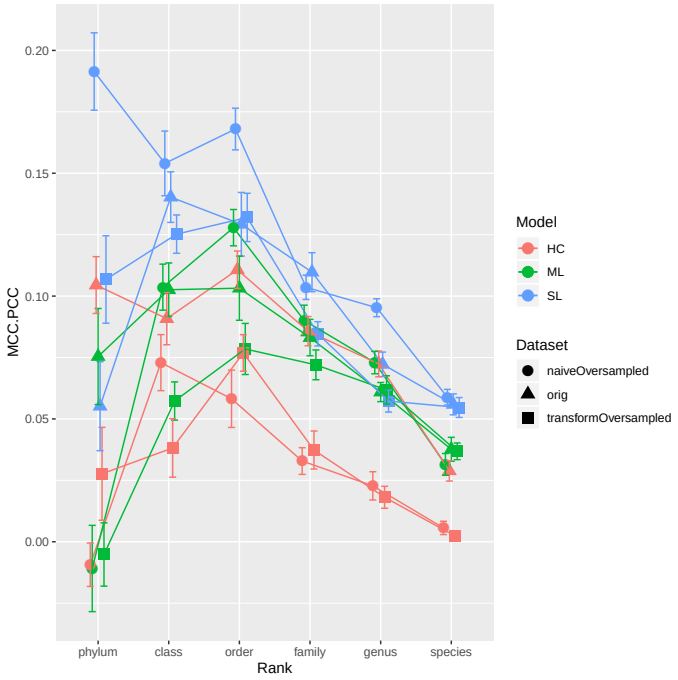


Figure 25: **Comparison Plot** of **average test scores** from epoch $[e_5 : e_{20}]$ and standard error of the mean that each classifier achieved according to MCC on each dataset.

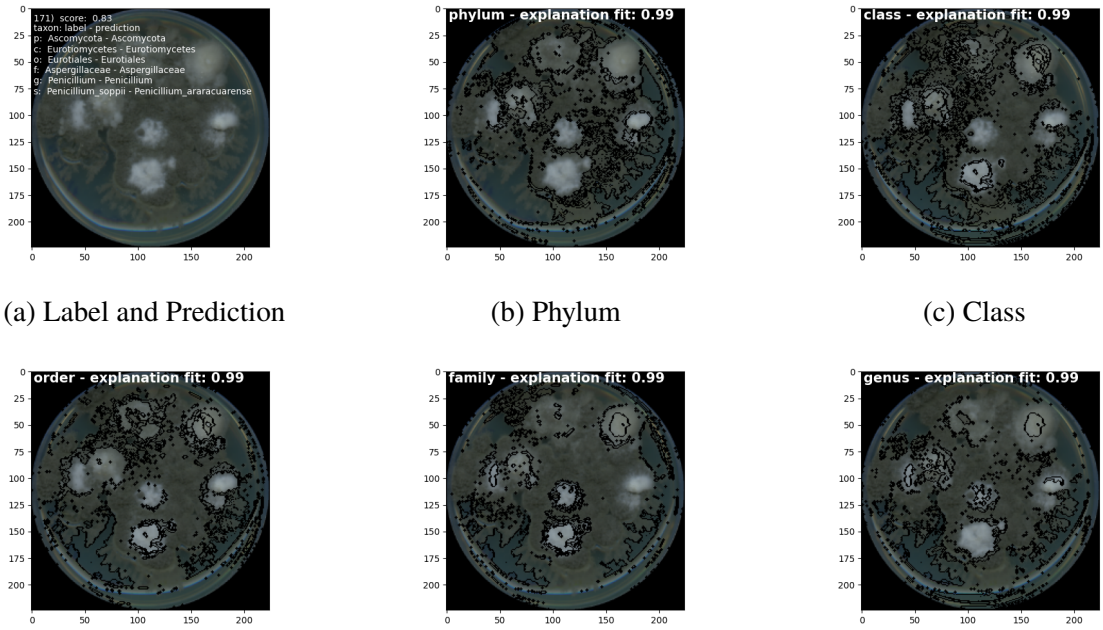
7 Behind the Scenes - Model Prediction Explanation with LIME

In the case of image explanation, LIME are local surrogate models that explain individual predictions by segmenting an image into superpixels and creating a perturbed set around the instance that we wish to explain. Explaining single instances, i.e. single images, has the advantage that a meaningful explanation for a correct prediction can still be made even if the overall performance of the classifier ain't best.

Coming back to fungi, (Lehmann, Zheng, Ryo, Soutschek, Roy, Rongstock, Maaß, and Rillig 2020) suggest that the hyphal growth speed and the complexity of the hyphal structure are phylogenetically conserved and observed at least at phylum level. A model explanation might confirm such findings, possibly even to a lower taxonomic rank or reveal new once. Hence it is important that the model takes possible taxonomic interactions into account. The big bang classifier at epoch e_4 on the naive random oversampled data is thus the **classifier of choice** in addition to it's shorter training time. For that model, I define a prediction score to identify **samples of interest** in the test set (for example Fig.27a, 26a, 28a, 29a), that indicates the percent of correct prediction, s.t. zero correct out of six predictions have a score of 0.0 whereas six out of six correct predictions have a score of 1.0. Samples with a score > 0.8 are classified correctly with at least five out of six taxons, which I will focus on for explanation.

Figure 27 shows the explanation for *Apiotrichum dulcitum*. It's taxons are predicted correctly by the model to species level. In the explanation figures 27b-27g black marks and surrounded areas indicate important regions for prediction. *Explanation fit* can be seen as a correlation score between LIME explanation and the original model. We can observe that the explanation highlights a full outline for phylum and class. Given a surface cover after a certain time of fungi growth, a full outline can be interpreted as hyphal growth. In lower taxonomic ranks the outline thins. Moreover certain structures in the fungal surface are highlighted. *Penicillium araracuarense* (Fig.26) is identified until genus rank. It's surface is a lot more complex. Marks appear especially in the areas surrounding the white parts. From class level on until genus the outline is more clearly marked. The explanation however also includes visual artifacts from the Petri dish lip. *Trichoderma asperellum* in figure 28 provides an interesting contrast as it covers the whole dish. Having values between 0.6 and 0.7 the explanations have a relatively low fit. It is interesting to observe that for phylum and class the explanation seems to focus on structures, whereas for higher taxonomic ranks also areas are included. The explanation of *Exophiala oligosperma* is a mixed story of success. Most importantly

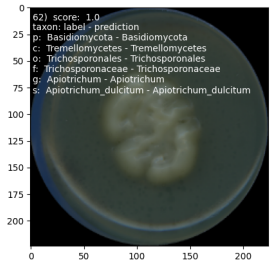
it shows the necessity of explanations, as all but one include a handwritten date on the Petri dish. Moreover regions just in front of the fungi are marked, let's say the shore, as well as inner structures.



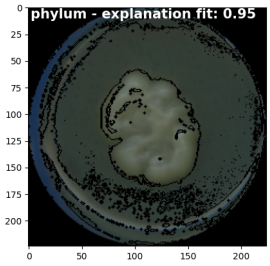
(a) Label and Prediction (b) Phylum (c) Class

(d) Order (e) Family (f) Genus

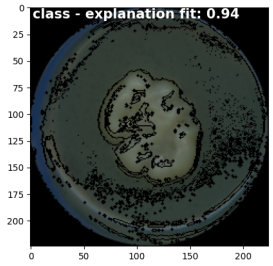
Figure 26: **LIME explanation** of *Penicillium araracuarense* made by ML e_4 with neighborhood size $\pi_x = 1000$ and 100 superpixels. Segmentation is performed by quickshift algorithm with $kernelsize = 6$, $max\,distance = 50$ and $ratio = 0.5$. Blacked areas are areas of importance for prediction.



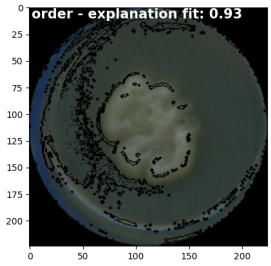
(a) Label and Prediction



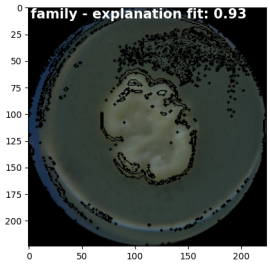
(b) Phylum



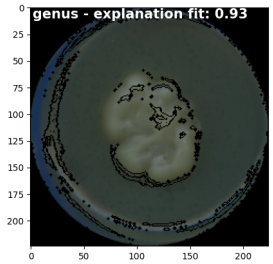
(c) Class



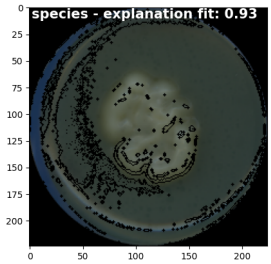
(d) Order



(e) Family

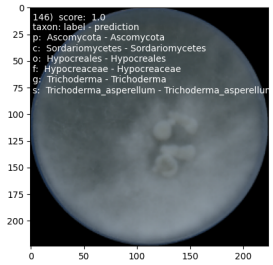


(f) Genus

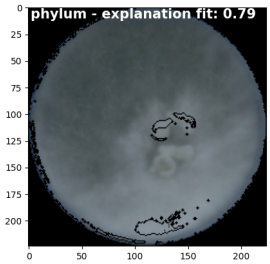


(g) Species

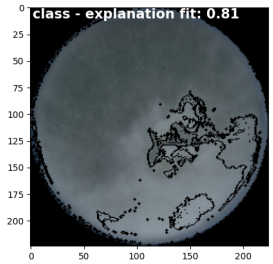
Figure 27: **LIME explanation** of *Apiotrichum dulcitum* made by ML e_4 with neighborhood size $\pi_x = 1000$ and 100 superpixels. Segmentation is performed by quickshift algorithm with $kernelsize = 6$, $max\,distance = 50$ and $ratio = 0.5$. Blacked areas are areas of importance for prediction.



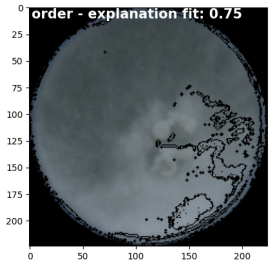
(a) Label and Prediction



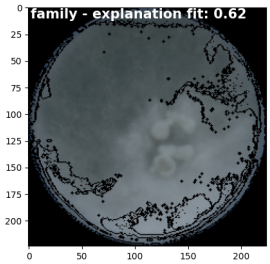
(b) Phylum



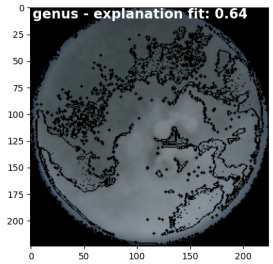
(c) Class



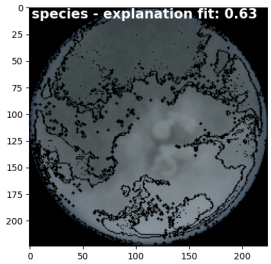
(d) Order



(e) Family

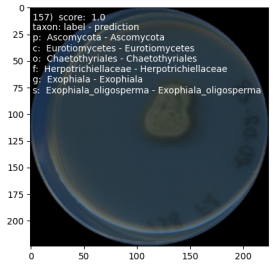


(f) Genus

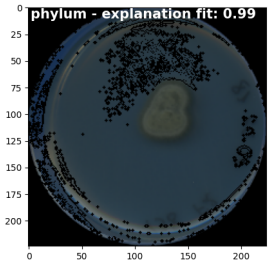


(g) Species

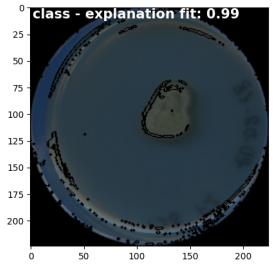
Figure 28: **LIME explanation** of *Trichoderma asperellum* made by ML e_4 with neighborhood size $\pi_x = 1000$ and 100 superpixels. Segmentation is performed by quickshift algorithm with $kernelsize = 6$, $max\,distance = 50$ and $ratio = 0.5$. Blacked areas are areas of importance for prediction.



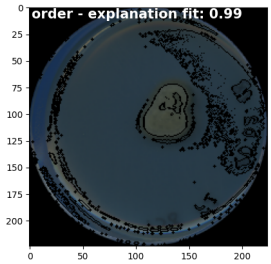
(a) Label and Prediction



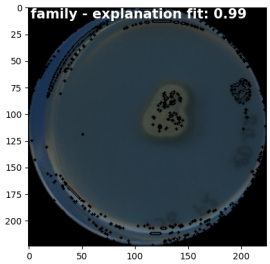
(b) Phylum



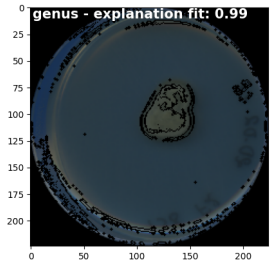
(c) Class



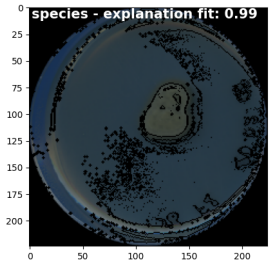
(d) Order



(e) Family



(f) Genus



(g) Species

Figure 29: **LIME explanation** of *Exophiala oligosperma* made by ML e_4 with neighborhood size $\pi_x = 1000$ and 100 superpixels. Segmentation is performed by quickshift algorithm with $kernelsize = 6$, $max\,distance = 50$ and $ratio = 0.5$. Blacked areas are areas of importance for prediction.

Part V
Discussion

8 Discussion

In this work I study the application of deep learning on taxonomically labeled image data of mycorrhizal fungi, find explanations behind classification and link them to recent findings in literature. I started off with an extensive data preparation, that are data extraction and missing value imputation. After class imbalance assessment I create three datasets, a naive random oversampled dataset with random flips, a transform random oversampled dataset with color and lighting augmentation and the original data. The hierarchical structure of taxonomic data I engage by building three types of multi-class classifiers, separate local per-level classifiers, a multi-label multi-class classifier (big bang) and hierarchically chained local per-level classifiers. Each model's performance is tested on each dataset. For the two first mentioned models stability is tested in 5-fold cross validation. I find that all models perform best on the naive oversampled dataset in terms of performance and stability. Hierarchically chained classifiers have the worst performance. Separate local per-level classifiers reach a slightly higher performance score than all other models. However they do not include hierarchical information, which is why big bang should be given preference. Test samples with the highest prediction score I explain using the model agnostic approach LIME. I find that with sufficient data preparation deep learning offers a way to classify mycorrhizal fungi to taxonomic rank of phylum, class and order at least to some extend, given the diverse nature of their appearance and the difficulties provided with the dataset. Explanations on test images with high prediction score mark outlines of the fungi and inner regions, which might be interpreted as indicator for hyphal growth and hyphal structures. Having higher prediction scores for phylum, class and order as well as the prevalence of such explanations seems to confirm phylogenetic conservation of such traits in lower taxonomic ranks as suggested by (Lehmann, Zheng, Ryo, Soutschek, Roy, Rongstock, Maaß, and Rillig 2020).

Tackling class imbalance, missing values and the small size of the dataset provide a challenge. I engage this on the side of training by random oversampling and the usage of matthews correlation coefficient as a metric. I find that naive random oversampling has the most positive effect, it accelerates learning and seems to boost test scores for phylum, class and order, while also reducing

noise influences of the set. Transform oversampled data also accelerate learning, yet at a slower pace. However the general classification score for this set on the test data drops dramatically, which might indicate that color or lighting are important features for taxonomic identification. Future works might consider the application of *Synthetic Minority Over-sampling Technique (SMOTE)*, (Chawla, Bowyer, Hall, and Kegelmeyer 2002), as it offers an elegant way of the synthetic creation of new samples in similarity to the existing ones. Matthews correlation coefficient proves to be a robust metric when facing class imbalance. As it includes true negatives, it's score is much more realistic than accuracy. Moreover it is able to show negative correlation.

Despite producing slightly better test scores, separate local per-level classifiers fail to include hierarchical information content. If there is no need to include such information, separate local per-node classifiers could increase test score even more. However they would need a larger and more balanced set to train on. Hierarchically chained local per-level classifiers in their implementation performed below expectation. I hoped that passing parameters from one classifier to it's successor as an implementation of transfer learning would increase the performance and would model the taxonomic hierarchy. However it's performance remained utterly low. Moreover such classifiers are at risk of error propagation. Here again performance could be improved by utilization of per-node classifiers if a larger and more balanced set to learn from is given.

A compromise between performance and a possible self-learned inclusion of hierarchy offers the big bang classifier. Being a single, relatively fast learning model I choose them as the go to approach for deep learning applications in mycorrhizal fungi classification.

References

2020. "MXNET Incubator." https://mxnet.incubator.apache.org/api/python/docs/api/.

2020. "open cv."

Alpaydin, Ethem. 2014. *Introduction to Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA: MIT Press, 3 edition.

Batista, Gustavo, Ana Bazzan, and Maria-Carolina Monard. 2003. "Balancing Training Data for Automated Annotation of Keywords: a Case Study." pp. 10–18.

Chawla, Nitesh, Kevin Bowyer, Lawrence Hall, and W. Kegelmeyer. 2002. "SMOTE: Synthetic Minority Over-sampling Technique." *J. Artif. Intell. Res. (JAIR)* 16:321–357.

Chicco, Jurman. 2020. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation." *BMC Genomics 21, 6* .

Frac, M., Silja Hannula, Marta Belka, and Malgorzata Jedryczka. 2018. "Fungal Biodiversity and Their Role in Soil Health." *Frontiers in Microbiology* 9.

Glorot, Xavier and Y. Bengio. 2010. "Understanding the difficulty of training deep feedforward neural networks." *Journal of Machine Learning Research - Proceedings Track* 9:249–256.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Gorodkin, J. 2004. "Comparing two K-category assignment by a K-category correlation coefficient." *Computational biology and chemistry* 28:367–74.

Huang, Gao, Zhuang Liu, Laurens van der Maaten, and Kilian Weinberger. 2017. "Densely Connected Convolutional Networks."

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." *Neural Information Processing Systems* 25.

Lehmann, Anika, Weishuang Zheng, Masahiro Ryo, Katharina Soutschek, Julien Roy, Rebecca Rongstock, Stefanie Maaß, and Matthias C. Rillig. 2020. "Fungal Traits Important for Soil Aggregation." *Frontiers in Microbiology* 10:2904.

Mathews, W. B. 1975. "Comparison of the predicted and observed secondary structure of T4 phage lysozyme." *Biochimica Et Biophysica Acta (bba) - Protein Structure* pp. 442–451.

Molnar, Christoph. 2019. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/.

Neubert, Peer. 2015. *Superpixels and their Application for Visual Place Recognition in Changing Environments*. Ph.D. thesis.

Ortigosa-Hernández, Jonathan, Iñaki Inza, and José Antonio Lozano. 2017. "Measuring the class-imbalance extent of multi-class problems." *Pattern Recognit. Lett.* 98:32–38.

Polyak, Boris. 1964. "Some methods of speeding up the convergence of iteration methods." *Ussr Computational Mathematics and Mathematical Physics* 4:1–17.

Ribeiro, Marco, Sameer Singh, and Carlos Guestrin. 2016. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier." pp. 1135–1144.

Shmueli, Boaz. 2019. "Multi-Class Metrics Made Simple, Part I: Precision and Recall." https://towardsdatascience.com/the-best-classification-metric-youve-never-heard-of-the-matthews-correlation-coe

Silla, Carlos N. Jr. and Ale xA. Freitas. 2011. "A survey of hierarchical classification across different application domains." *Data Mining and Knowledge Discovery* 22:31–72.

Stefan Hempel, Juan F. Dueñas. 2020. "FunTrait Project." https://www.biodiversity-exploratories.de/en/projects/current-projects/microorganisms-fungi/funtrait/.

Tomislav Petković, Sven Lončarić. 2015. "An Extension to Hough TransformBased on Gradient Orientation." *arxiv* .

Turland, Nicholas, John Wiersema, Fred Barrie, Werner Greuter, David Hawksworth, Patrick Herendeen, Sandra Knapp, Wolf-Henning Kusber, De-Zhu Li, Karol Marhold, Tom May, John Mcneill, Anna Monro, Jefferson Prado, Michelle Price, and Gideon Smith. 2018. *International Code of Nomenclature for algae, fungi, and plants (Shenzhen Code) adopted by the Nineteenth International Botanical Congress Shenzhen, China, July 2017*, volume 159.

Vedaldi, Andrea and Stefano Soatto. 2008. "Quick shift and kernel methods for mode seeking." In *In European Conference on Computer Vision, volume IV*, pp. 705–718.

Weiss, Noa. 2019. "The Hitchhiker's Guide to Hierarchical Classification." https://towardsdatascience.com/https-medium-com-noa-weiss-the-hitchhikers-guide-to-hierarchical-classification-

Zhang, Aston, Zachary C. Lipton, Mu Li, and Alexander J. Smola. 2020. *Dive into Deep Learning*. https://d2l.ai.

Zhu, Rui, Ziyu Wang, Zhanyu Ma, Guijin Wang, and Jing-Hao Xue. 2018. "LRID: A new metric of multi-class imbalance degree based on likelihood-ratio test." *Pattern Recognit. Lett.* 116:36–42.