Automated classification of syllable types in vocal sequences of pups of the greater sac-winged bat Saccopteryx bilineata

Master Thesis
Faculty of Science, University of Bern

handed in by

Simon Hiller

2021

Supervisors

Prof. Dr. Daniel Wegmann, University of Fribourg, Supervisor
Dr. Andreas Fischer, University of Fribourg, Co-Supervisor

Abstract

The main goal of this project is to evaluate the application of deep learning (DL) models for the classification of syllable types in recordings from the pups of the neotropical greater sac-winged bat (*Saccopteryx bilineata*). To realise this, two experiments were conducted, with the first one demonstrating how it is feasible and the second one presenting an actual realization of a prototype of an automated syllable type classification pipeline.

We aimed to classify the different syllable types of the adult territorial song and one variable juvenile syllable within variable vocal practice sequences. This was worked out in a supervised machine learning framework, where we investigated the effect of different preprocessing variants in combination with the features as spectrograms or histogram of oriented gradients (HOG). For the models, we considered different types and different constellations of artificial neural networks (recurrent, convolution and densely connected).

Our findings are promising and show that it is feasible to create an automated syllable type classification application in the given objective, although the syllables of the juveniles show a high degree of intra- and interindividudal variability because they mirror developmental and learning processes throughout ontogeny.

Acknowledgements

I would like to express my gratitude to Ahana Aurora Fernandez for providing the labelled dataset and the hours of discussion and more that formed the basis for conducting these experiments. For their support and patience with me, I sincerely thank my supervisors Andreas Fischer and Daniel Wegman. Thanks also to Petar Veličković, as many of the schemes on neural networks are modified versions of his publicly available work, his TiKZ GitHub repository was a little treasure for me. And for providing a graphics processing unit (GPU) cluster, I would like to thanks the HPC team of UBELIX (http://www.id.unibe.ch/hpc), the high-performance computing cluster at the University of Bern, Switzerland. And finally, I would like to thanks my friends Anna, Bettina and Sophie who all proofread my thesis.

Contents

1 Introduction

A promising approach to deepen our knowledge about human language capacity is to investigate if and to what extent key components - such as vocal imitation - have evolved in other species [VW20]. These studies yield a large amount of data and the analysis of this data is time-consuming if vocalisations cannot be detected automatically. Furthermore, with complex vocal sequences consisting of several different syllable types, automatic classification of syllable types is a desirable feature. The neotropical greatersac-winged bat species *Saccopteryx bilineata* is a promising candidate for biolinguistic studies, as this species is capable of vocal imitation [Knö14]. As the study of this specific trait in *S. bilineata* progresses, an increasing amount of research is devoted to the manual analysis of large amounts of bioacoustic data. More about the difficulty of manual analysis of bioacoustic data, especially syllable classification, can be found in Section 2.2 and Section 2.4.

For human speech, the understanding, parsing and encoding process is a highly studied procedure and is now used by a wide range of applications belonging to the broad field of natural language processing (NLP). Developments in this field have made it possible to convert speech directly into written form or translate it into another language in real time and many other astonishing feats [Kan+20]. A big step for this language application came in the form of deep learning (DL), a subdivision of machine learning (ML), which emerged as a promising field. DL involves the processes of "learning" information and then applying that learnt information to perform similar tasks by training deep and complex models.

DL tools have already been used in the field of bioacoustics. One of these applications is species identification, which uses the initial approaches of human speech decoding applications based on ML by searching and classifying patterns in the acoustic recordings and trying to combine the results to meaningful information [Sto+19]. Depending on the animal monitored, one was interested in the recognition of a song or only a single syllable, with the interest of distinguishing between animals. In recent years, the number of biolinguistic studies in animals has continued to increase [VW20; Wir+19]. Especially in animals that are thought to be able to contribute to the study of human speech, such as the bat *S. bilineata*, which comprises a promising taxon for biolinguistic studies [Knö14]. This, combined with the ongoing development of computational power leveraging the analysis of more information [Amo+19], makes the interest in applying NLP to biolinguistics increasingly promising. However, at the time of writing, this area appears to be largely unexplored, and no evidence of software for fully automated classification of bat syllable types has been found.

This project aims to deepen the understanding of syllable type classification of the *S. bilineata* using supervised ML on the basis of the initial DL models used for species identification. The experiments are divided into two parts. In the first we focus on different feature preparation methods as well as insights about the performance of different ML models. The second part focuses on the building of a pipeline for

an automated syllable type classifier prototype based on a dynamic length audio, with the generation of accessible results for further processing in other applications. The intention behind the prototype is to construct a guide for building a more elevated classifier which uses the newly emerged techniques around NLP.

This thesis is structured as follows. Chapter 2 highlights the work which contributed to the existence of this thesis. It includes a brief overview of language analysis methods and the current state of the use of ML in this field. Chapter 3 introduces the data we used for learning (i.e. training, model evaluation and selection) and chapter 4 presents and discusses the methodology included in the experiments. The steps and configuration used for the results and their outcomes are described and discussed in chapter 5. Finally, the conclusions from the results are drawn in the first part of chapter 6, in the second we render the feature works.

2 Background

What belongs to biolinguistics, what do we know about the vocal capabilities of the bat *S. bilineata* and how is this connected to ML. This and many more insights are provided throughout this chapter.

2.1 Biolinguistics

More than anything else, language defines human nature. Language allows infinite expression with finite means [HCF02] enabling us to exchange ideas and transmit knowledge across generations, building the foundation of our technological advances and cultural heritage. However, the origins of language remain a puzzle, possibly the most challenging in science because language does not fossilize [Tec10]. This renders tracing the evolution of language extremely challenging. The current most promising approach to solve this puzzle is to comparatively investigate the biological foundations of language across species [Fit18].
The biolinguistic approach combines research methods from fields such as neurogenetics, animal behaviour, linguistics, developmental psychology, bioinformatics and mathematics to understand the underlying mechanisms and processes of human language. This make sense, as language itself is a complex system composed of several key factors (e.g. syntax, semantics, vocal imitation) and various cognitive abilities (e.g. social learning, theory of mind, recursive processing) [FHB10]. Although no non-human animal communication system equivalent to the human language exists, bioacoustic research demonstrated that non-human animal vocal communication can be complex in terms of vocal repertoire size (i.e. the number of distinct syllables) (e.g. bats: [BV04]; songbirds: [EPV89]; cetaceans: [PM71]), syntactical rules (i.e. the rules according to which the acoustic units – for example syllables - are arranged within vocal sequences; [Wei+14]) and vocal versatility (i.e. vocal plasticity allowing to modify existing signals and/or learning new signals from scratch based on social experience) [Wir+19]. Furthermore, non-human animals possess multiple cognitive skills which are required for language acquisition in the first place, such as associative learning, vocal imitation and joint attention [FHB10]. Therefore, the biolinguistic research in non-human animal vocal communication aims to understand if and to what extent key features and cognitive skills evolved compared to our own.

2.2 Vocalisation

Across many species, vocal communication is a common social behaviour. Bioacoustic research seeks to uncover the structure, mechanism, behaviour and reason for vocal communication. This includes how signals are generated and perceived and how information about identity, fitness, presence of danger, environmental

stress and possibly much more is transmitted. In the same way, the extent to which fitness and reproductive success are influenced by communication, is also of interest [Ker+16; MPS04]. This list is not exhaustive, thus as one can imagine, there is a large amount of unanswered questions in this field. Unfortunately, only a tiny fraction of vocal communication seems to have been studied in detail so far. Here, the challenging work of segmenting acoustics into temporally discrete units and classifying them correctly according to specific characterising properties plays a major time-consuming, yet crucial role.
When it comes to vocal learning in non-human animals, the research field seems to be quite limited with focus on birds [PJ12] and evidence in mammals centered around cetaceans [Jan14], pinnipeds [RC14], elephants [Poo+05] and bats [Knö14]. In recent years, researchers are debating about understanding vocal learning as a continuum rather than a dichotomy (absent vs present), including not only vocal imitation but also vocal plasticity, including the modification of innate calls based on social experience. Martins et al. discuss the concept of a wider vocal learning continuum including more vocal abilities which are now being uncovered in other species. Thus engaging a broader comparative study about vocal learning, as the current view seems to be too narrow and excludes vocal abilities [MB20].
Still, non-human mammalian species which possess the ability of vocal imitation (i.e. learn a vocalisation from scratch based on the auditory input from a tutor) are especially promising candidates to study mechanisms and processes involved in language learning. Vocal imitation is a mandatory capacity for human babies and infants to acquire speech (i.e. oral output of language) [Vih14]. One of the most important first steps in human infant speech acquisition is the canonical babbling phase during which infants acquire the phonological repertoire of the maternal language [OLL80; Vih14]. Infants imitate the basic speech sound subunits (i.e. consonant-vowel syllables) and refine their oral output and control over speech articulators by rehearsing syllables in sequences (e.g. "mamama", "bababa") [KM96].

2.3 Vocal communication of the greater sac-winged bat

Figure 2.1: Greater sac-winged bats (*Saccopteryx bilineata*) roosting in their day roost, the young one on the right is vocalising. The smaller bats with the dark fur are juveniles and the others with lighter fur are their mothers. (© Michael Stifter)

A highly promising candidate for comparative biolinguistic studies is the neotropical bat species *Saccopteryx bilineata*. This bat species is a vocal production learner (i.e. capable of vocal imitation) [Knö+10] and possesses a large vocal repertoire including two song types produced by adult males [KBV06]. During ontogeny, pups produce long vocal sequences composed of precursors of syllable types of the adult repertoire

mixed with pup-specific vocalisations. While babbling, pups acquire a part of the adult vocal repertoire through vocal imitation, namely the syllables of the territorial song [KBV06]. For several weeks, the precursors of the territorial song syllables gradually converge towards the territorial song of tutor males, irrespective of relatedness and pup sex [Knö+10]. Studying the vocal practice behavior of pups offers the opportunity to study the underlying mechanisms crucial for vocal imitation because babbling represents the oral output of ongoing learning processes. Hence, investigating acquisition of learned territorial song syllable types could serve to unravel shared mechanisms and key factors across mammalian vocal learners including humans.

2.4 The value of syllable type classification

To gain insight about potential information encoded in acoustic vocalizations, besides behavioural observations accompanying social vocalizations to infer the context, acoustic measurements are used. Acoustic measurements can be taken over the entire vocalization (e.g. over an entire multisyllabic call) and from acoustic units such as syllables. In general, syllables are defined as continuous sound surrounded by silence. To measure syllables it is necessary, to detect, label and sometimes classify them. Currently, measuring and analysing syllables in vocal sequences is an extremely time-consuming job worked out mostly manually with the assistance of audio analysis tools. Additionally, the missing knowledge about how animals perceive vocalizations (e.g. what is the smallest meaningful acoustic unit) makes the acoustic analysis challenging. Usually, syllabic units are defined based on information of the oscillogram and spectrogram. Then, possible information encoded in vocalizations can be inferred from accompanying behavioural observations.
One aspect of understanding vocal learning processes in *S. bilineata* involves analyzing the acoustic parameter change of territorial song syllables across ontogeny. This begins with the selection of these particular syllable types within babbling sequences from different individual pups at different time-points during ontogeny. So far, this selection process is extremely time-consuming, mostly because the syllable types within vocal sequences need to be manually selected and classified based on spectral similarity to syllables from the adult vocal repertoire (i.e. babbling contains also precursors of adult syllables other than territorial song syllables), and syllable on-and offset need to be manually determined. Therefore, the automatic detection of syllable on-and offset and the classification of distinct syllable types would remarkably facilitate the analysis of acoustic datasets by significantly reducing the time dedicated to syllable selection in acoustic recordings prior to acoustic measurements. This would result in increased sample sizes, allowing a higher number of focal individuals to be included in the analysis and increasing the frequency of sampling across ontogeny within an individual (e.g. analyse more bouts per day/week/month). This would enable future analysis of large datasets (big data) with temporal high resolution.
As automated classification seems to be favorable, the acoustic analysis of babbling is challenging: the gradual convergence of precursor vocalisations towards mature adult vocalisations and the high number of distinct syllable types within a vocal sequence renders automatic classification extremely difficult. Especially the gradual convergence of syllable precursors is challenging because the precursors are not exact copies of adult syllable types but vary in amplitude, duration and spectral parameters, both within and across individuals. Hence, conventional acoustic classification programs such as the template scan in Avisoft SasLab Pro quickly reach their limits.

2.5 Machine learning

During the last decade, the field of ML has evolved at a rapid pace, reducing complexity and simplifying accessibility to the technology. Yielding more and more interesting projects in the field biology research, which contribute again to better accessibility and growing interest for this technique in the field. This leads to many applications in species identification. But a quite untouched area is the syllable type classification of animal acoustics for analyzing their syllable type repertoire and combinations.
According to the author's knowledge, there seems to exist no work addressing ML on automatic classification of syllable types for any bat species. However, there exists one study where the syllable structure of the

greater horseshoe bats were analyzed and ML was employed for automated call sequence classification [Zha+19]. The syllable mining was conducted manually with the help of audio analysis software, without any ML applied at this step.
Most vocalisation studies in animals are generally used for species classification, monitoring the health condition of the environment or the animal itself. In a much smaller scale ML methods are used for analyzing vocalisation, the most of these, so far, have been applied to birds [Qia+17; Tch+04], and mammals (dogs: [Mol+08]; rodents: [CMN19]; and primates: [PGG10; Tur+16]). This reflects the current state of research, where the characterization and abstraction of vocalisation remains both an art and a science requiring typically expert knowledge. A new promising technique seems to exists in unsupervised latent models, trained on compressed structures across diverse animal vocal repertoire, and leveraged by their quantitative high throughput and ability to reveal statistical patterns in complex data of ML methods [STG20].
Although echolocation classification could be seen as syllable type classification, we perceive this as a different task, because the interest is in distinguishing the echolocation between different species (i.e. species identification) and not between different echolocation syllables from one individual. But as some problems overlap, the advances in this field could be a possible source of ideas. DL models have been tested for detection of echolocation in audio samples (they outperformed random forest algorithms) [Mac+18] and for classifying tropical bats based on their echolocation call [Che+20]. Both propose a specialized version of a convolutional neural network (CNN), where the later also uses residual paths like the ResNet [He+16a].
For our project we stick to the already implemented artificial neural network (ANN) models provided by the work around classification of the Eurasian woodcock based on calls or call peaks with DL from Gilles Waeber [Wae19]. One of the applied DL models in our work is similar to CNN_{FULL} from the bat detective work [Mac+18], with some more layers. Unfortunately, the second related work was not known to the author at the time the experiments were set up and carried out, so the findings could not be included in this work.
A promising CNN is the DenseNet [Hua+17], originating from the field of image classification. Huang et al. proposed densely connected convolution neural layers with reducing network parameters by reusing features. As this type of DL model is included in the TensorFlow framework, we included an adapted DensNet-121 in our experiments.

3 Dataset

We provide an overview of the data used and give a short statistical analysis.

The dataset used in the experiments belongs to the Free University of Berlin and is kindly provided by A. A. Fernandez from the Natural History Museum of Berlin in cooperation with M. Knörnschild, affiliated with Natural History Museum of Berlin and Free University of Berlin. It consists of acoustic recordings of bat pup vocalizations from *S. bilineata* in wav-file format. The acoustic recordings were obtained over two consecutive field seasons from two populations in Panama and Costa Rica. At each location, they were able to record the target pup in a distance of 2 to 4 meters with a high quality ultrasonic sound ($500\,\mathrm{kHz}$ sampling rate, 16-bit depth resolution). The set-up consisted of a microphone (Avisoft UltraSoundGate 116Hm, with condenser microphone CM16, frequency range 1-$200\,\mathrm{kHz}$ + 3dB) connected to a laptop (Lenovo S21e) running the software Avisoft RECORDER (v4.2.05 R. Specht, Avisoft Bioacoustics, Glienicke, Germany). The facts that the recordings could be conducted during daytime (i.e. most of the relevant social interactions of this species occur across the day in the dayroost) with excellent visibility during the day and that the bats clearly open their mouths when vocalising, allowed individual on-axis recordings. This, in combination with the directional characteristic of the microphone used and considering good recording conditions such as the avoidance of reverberation caused by foliage, enabled to obtain high quality recordings.
The postprocessing and labeling of the syllables were conducted with the software Avisoft SASLab Pro (v.5.2.09; R. Specht, Avisoft Bioacoustics, Glienicke, Germany). Start and end of syllables were determined manually by researchers who have over 15 years of expertise in assigning different syllable types in the vocalizations of the focal bat species. Assigning syllable types manually is a common procedure; for instance, babbled utterances from children are coded by human listeners without a clustering method. As the pup syllable types show a high spectro-temporal similarity with adult syllable types from first appearance onwards, the entire adult vocal repertoire was used for visual classification of syllable types in babbling bouts. A subset of the visually classified syllables was statistically verified by performing discriminant function analyses (personal communication AA. Fernandez).

3.1 Simple call dataset

As the syllables are manually labeled we decided for economical reasons to restrict our analyses to the syllable types of a territorial song only. This song is composed of several syllable types of which the majority are acquired through vocal imitation by the pups [Knö+10]. To be specific, they are B2, B3, B4 and VS as

well as VSV. In addition, we included the relatively variable syllable type UPS.

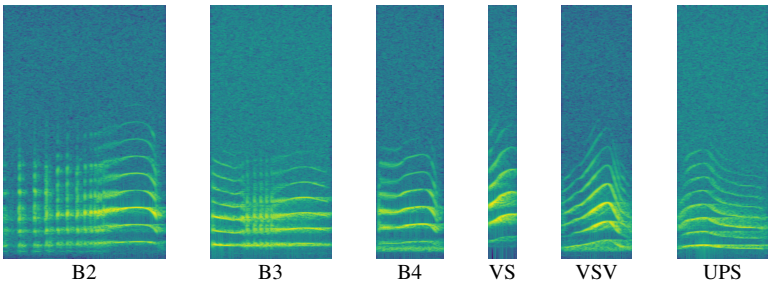

Figure 3.1: Spectrograms of all six syllables which were the target of our automated classification.

3.2 Descriptive statistics

Our restricted dataset consists of 1276 syllables. The k-fold cross-validation (CV) bins contain 11 syllables, because the smallest syllable type "B4" counts only 91 samples (see Figure 3.2). For a brief analysis of the syllable duration, see Table 3.1, the syllable length exhibits a high variability, with a ratio between the shortest and longest syllable of about 1:39. To account for the differences in the counts between the syllables classes one could consider to use data augmentation methods in the data pipeline. We decided to not apply any data augmentation, mainly as it is too complex to work on sound without extensive knowledge about it physical and biological aspects in respect to the bats.

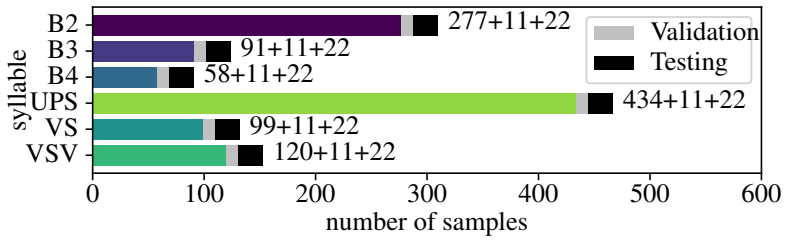

Figure 3.2: The distribution of the syllables from the simple call data set. Over all syllables we used 11 samples for validation and 22 for testing. The dataset is not balanced in the number of training samples, with a ratio of about 1 : 7 between the least and most represented syllable.

syllable type	mean/ms	median/ms	min/ms	max/ms	count
VS	13.48 ± 3.78	13.05	5.20	24.70	132
VSV	18.62 ± 5.85	16.93	10.47	39.33	153
B4	41.19 ± 15.15	39.06	16.85	73.19	91
UPS	45.12 ± 11.44	44.03	18.50	85.59	467
B2	59.17 ± 32.16	49.89	12.98	200.40	310
B3	54.67 ± 17.76	53.53	26.19	120.70	124

Table 3.1: Descriptive statistics of the syllable durations in the simple call dataset.

3.3 Terminology of syllable

We follow the definition of previous bat and bird literature for defining a "syllable" as one continuous vocal emission surround by periods of silence [Kan+94], [BV04]. They are the smallest independent acoustic units emitted from bats [BV04], [Lat+19].

3.4 Territorial Song

The territorial song (TS) is a multisyllabic vocalisation produced by all adult males to vocally defend their territories against potential intruders ([BKV09]). The territorial song encodes information about the males' quality, individual identity and social group membership ([Beh+06]; [EK13]).
It is hypothesized, that females include the evaluation of the territorial song into their mating decision. As the males produce this song mainly at dawn and dusk[EK13], the females should hear them sufficiently. So far, however, this has not been proven.
The TS is learned by the pups during their ontogeny through vocal imitation. This is achieved by the pups imitating the male tutors of their natal group [Knö+10]. This is one of the main reasons for the continuing research interest in this song.

4 Method

From the beginning it was clear that we want to apply convolution neural networks to spectrogram input. How this could be achieved and which techniques were applied is covered in this chapter.

4.1 Machine learning

ML is a fast evolving software art based on statistics as well as data and computer science. It is a subdivision of artificial intelligence (AI) where computer systems run statistical algorithmic models to effectively perform specific tasks without using explicit instructions by a human expert. Instead, they are recognize and learn patterns from the data they see. More recently, ML models were able to master tasks that were thought to be too complex for machines, combined with the ability to perform a given task consistently and tirelessly, they could speed up the experimental process enormously and enable much broader searches for patterns. Even more exciting is the finding that in some cases, computers seem to be able to learn patters that are beyond human perception. ML is now used in many fields and plays a central role in tasks such as speech recognition and translation between languages, autonomous navigation of vehicles and article recommendations.
Adapting ML to our problem results in applying a ML algorithm to a set of data (our recordings) and some knowledge about this data (the syllable type with their time boundaries). Through learning from the training data, the algorithm can apply what it has learned to make a prediction about the type of the syllable exhibited in a given spectrogram. The algorithm is considered to have learnt the task, when more test samples are diagnosed as the correct syllable type as a result of optimizing its parameters. As soon as the parameter change in a learning cycle does not result in a significant impact in performance, the ML algorithm can be seen as stable.

4.2 Pipeline

In the context of ML, the pipeline is a set of components describing the fate of the data. It starts at the creation of the data and ends in analysing the performance of the ML algorithm.

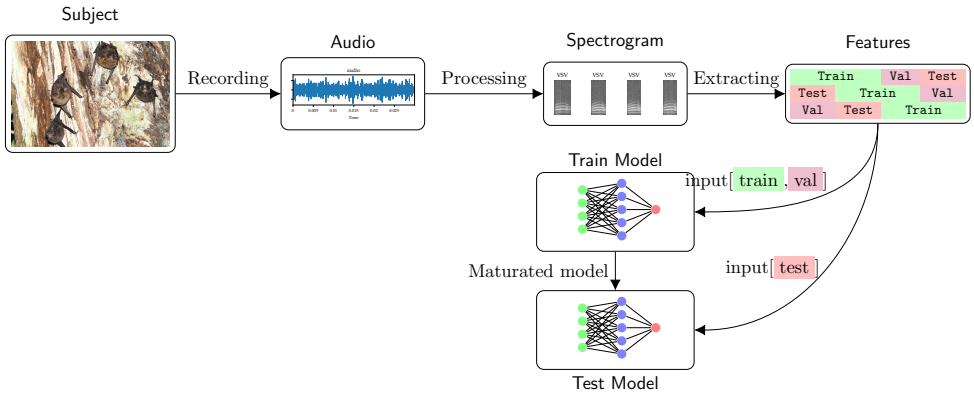

Figure 4.1: Flow diagram of the data pipeline adapted for this project. It shows the different formats into which the data is transformed, finally it is transformed in such a way that the ML model can understand and process it.

4.3 Data preprocessing

Data preprocessing defines all the processing and transformations which are applied to the data before it reaches its final form that can be digested by the ML algorithm. Certain concepts used for data preprocessing in this project are explained below.

4.3.1 Silent detection

For noise reduction and padding of the extracted audio parts containing the syllable, we had to extract background noise from the audios.
Adapted from the librosa.effect.split [McF+20], we detect silent parts based on the root mean squared (RMS) energy of the audio. We set the threshold to the average of the RMS energy of the audio without the syllable parts in it, as these are known foreground parts. This leads to a conservative definition of the background audio, as this method was originally intended only for padding and thus a low-noise background audio was the goal. The entire algorithm unfolds as follows:

$$silent = \begin{cases} 1 & mse < mean(mse \text{ without label}) + 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

— As shown in 4.1, the samples belonging to silent are marked with 1 in the first step and then successive samples are composed into silent slices.

— The silent slices are then shrunk on both sides by $5\,\mathrm{ms}$ and filtered against a minimum length of 10 samples. The idea behind this (which is not proven to be true for every case) is to remove the samples in the slice, where the energy becomes stronger and could lead in concatenated slices to small peaks at the beginning and end of a slice.

— Finally, all silent slices that overlap with a label are filtered out.

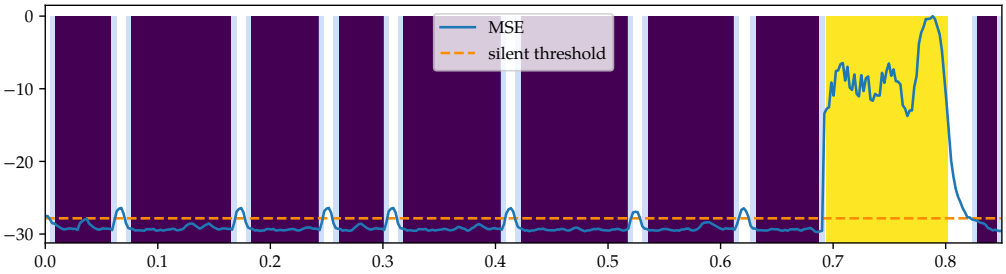

Figure 4.2: Plot of the result of the silence detection algorithm. The blackcurrant rectangles are the silent parts, on both sides the piece of $5\,\mathrm{ms}$ by which the silent parts were reduced is shown in transparent light blue. The yellow rectangle is a labelling part.

4.3.2 Audio splitting algorithm

For the second experiment, with the goal of finding and classifying syllable types in an audio recording of the bats, we additionally train the network to distinguish between syllables and background. In a comprehensive approach, we would train the network to learn the beginning and end of syllables, similar to the problems in human handwriting recognition. Since we do not have enough knowledge in this area for this work, as well as too little capacity to learn this, we have opted for a simple version where we include background patterns in the dataset and nothing more. For the training dataset, therefore, a preprocessing step is to split the audio and label the resulting chunks as background or as the corresponding syllable contained in the chunk. This is achieved by applying a moving window to the audio source and assigning the resulting audio slices to either the corresponding syllable type or the background. The audio slice is assigned to the corresponding syllable type as soon as a defined percentage or more of a slice is covered by a syllable or the boundaries of the syllable lie within the window, see Figure 4.3.

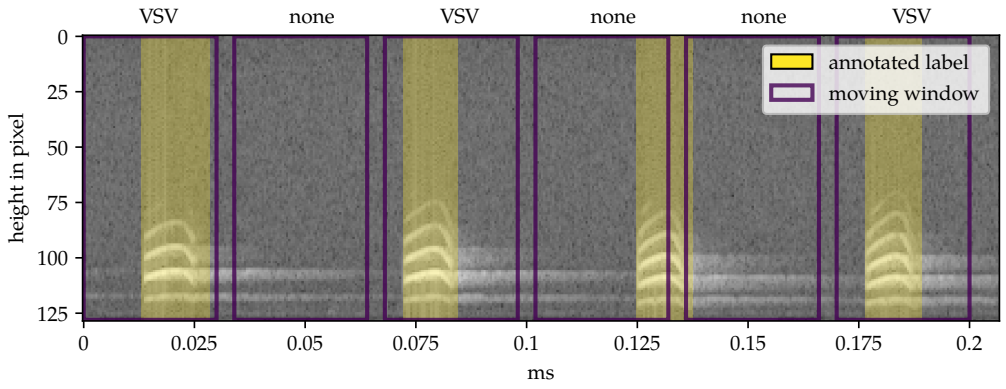

Figure 4.3: Visualisation of the audio splitting algorithm with a spectrogram as background from a labelled audio file. The rectangle with the purple borders represents the moving window with the assigned labels on the upper x-axes. The yellow rectangle covers the boundaries of the annotated syllable. For visualisation reasons, we used parameters ($30\,\mathrm{ms}$ window length and $34\,\mathrm{ms}$ strides) that do not result in overlapping windows.

For the training dataset, in order to reduce the number of slices marked as background, we discarded them after successive background slices reached three times the window length. Ideally, this should also prevent unlabelled syllables from being marked as background, as this concept ignores slices which are a bit further away from the labels. In the hope that there are no unlabelled syllables near labelled syllables.

4.3.3 Spectrogram

For audio analysis, the Fourier transform is one of the main tools to depict the audio and understand its inner parts. In our digital world the discrete Fourier transform (DFT) powers all the daily used data transport with any type of signal wave, especially electromagnetic waves. The short-time Fourier transform is a multi stage process applicable on audio, this and many derivations of it allow us to generate images from signal waves in which we are able to see patterns, these images are called spectrograms. The short-time Fourier transform determines the sinusodial frequency and phase content of local sections of a signal as it changes over time. The sampled data of a signal is divided into shorter segments of equal length and then each segment is subsequently Fourier transformed and the magnitude of the frequency spectrum of each segment is derived. The spectrum vectors are subsequently stuck together over the time axis to form the spectrogram image.
In our experiments we use a Fast Fourier transform which computes the DFT with a sufficiently good time complexity. The DFT for a fixed N is the linear operator $\mathcal{F}\colon x^N \to X^N$ on $\mathbb{C}^N$ defined by:

$$X_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn}, \quad k = 0, ..., N-1 \in \mathbb{Z}$$

For computational efficiency and not relying on the unitary transform, many implementations use the simpler scaling $\frac{1}{N}$. A unitary transform assumes that the energy in the physical domain is the same as the energy in the Fourier domain, i.e., satisfies Parseval's theorem.
Due to the temporal segmentation of the signal, spectral leakage occurs in the Fourier transform, i.e. signal waves which are not zero at the beginning and end of the window are slightly distorted. Since the curve of the signal at the beginning goes from 0 to the starting amplitude value (the opposite is true for the final segment), which leads to additional frequencies that are captured by the Fourier transform. Therefore, to smooth this behaviour, we apply a tapering (window) function to the segmented signal. We use the Hamming window in this project, it tends to fade the signal in and out at the beginning and end of the segment, resulting in an artificial periodisation of the signal within the time window length. This increases the dynamic range, but unfortunately leads to a trade-off in lower sensitivity. The Hamming window, like the Hann window, is one of the frequently used all-round windows. The loss in information at the edges (e.g. information about transients) of the window can be reduced by overlapping the windows by some percentage. The basic idea here is that one can average FFT results form overlapping frames and thus obtain a better frequency representation of our time-domain signal. The actual frequency resolution is still the same as without overlapping.
The resulting spectogram image thus shows us the amplitude of frequency bins over time bins. With the parameters width and height we can control the time and frequency resolution, i.e. across how many frequencies and seconds the amplitude in a pixel is averaged. But, a signal in time and frequency domain cannot have finite support unless it is identically zero, known as the Garbor limit, based on the Heisenberg uncertainty principle, the rise in temporal resolution would give loss in frequency resolution and visa versa. A possible solution would be a multiresolution analysis, which can give a good temporal and time frequency resolution for high, and low frequency events.

4.4 Features

When we train a ML model, we need something to describe the subjects which the model has to learn. In the context of ML the features can be seen as the values of the subject descriptors. They can be in many different forms, depending on the task, source, ML and much more. In this project we use time series data in different forms extracted from an audio source. Even the image normally contains, spatial information, and the movie extends to time, images are also able to contain the time information, as example in spectrogram images where each column represents a feature vector at time x. As the resources are somehow limited and efficiency plays a role when a large amount of data is analyzed, one is interested in breaking down the dimensions of features to the ones which play a significant role in identifying the subjects. This can be done by the model or by an additional "hand crafted" high-level features extracting step in the data pipeline.

4.4.1 Audio features

When recording an audio, the sound is sampled at a certain rate and forms a bitstream. The sampling rate is derived from the highest frequency to be captured by the recording, as it must strictly be more than the Nquist rate, which is twice the highest frequency. Our recordings mostly have a sampling rate of $500\,\mathrm{kHz}$, so we are able to capture frequencies up to $250\,\mathrm{kHz}$. This is more than enough to fully record the social vocalisations of *S. bilineata* composed of fundamental frequencies from $7\,\mathrm{kHz}$ to a maximum of $80\,\mathrm{kHz}$. As there is currently some progress in using raw waveforms as input for ML models, we decided to transform the bitstream into some higher level features, such as the spectrogram or HOG, both of them showed good performance for different ML models.

4.4.2 Raw

In this project we call raw features a feature tensor built up from the resulting pixel values of the spectrogram image in cray scale (e.g. one channel), which are then digested by the ML model. This is not unique to this project, as already many researchers defined audio raw features primarily as features resulting from some Fourier transform-like transformations.

4.4.3 Histogram of oriented gradients

The histogram of oriented gradients (HOG) features are used to extract the time-frequency information from the spectrograms by retrieving the gradients present in it. This is used in a wide range of applications, such as acoustic scene classification [RG15; YK17], face detection in images [Sun+14; DT05] or videos [Sur+18]. We use a specialisation called Slit-Style HOGs [TT09], as this was already implemented in the framework we used and provided acceptable results in combination with the long short-term memory (LSTM) model in the previous experiments from Gilles Waeber [Wae19].

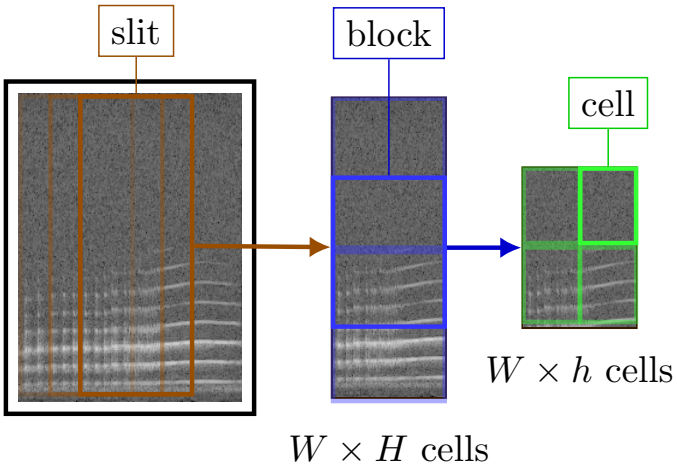

Figure 4.4: Visualisation of the different areas defined in the slit style HOG transformation and how they are encapsulated in each other. The slit contains 4×2 cells and a block is $h = 2$ cells high.

Summarized, the algorithm is described as follows: The spectrogram is windowed by a slit, a rectangle of size $x \times y$, the resulting slit images are then split up in $H \times W$ cells. These cells are then assigned to blocks, in such a way that the defined blocks overlap in the height by one cell, see Figure 4.4. Per cell the gradients are

extracted in π bins, where the gradient bins are evenly spaced over 0 - 360 ("signed gradient"). The resulting gradient vectors are concatenated together per block in a vector with a size of $h \times W \times \pi$. The block vectors are then normalized and concatenated in the slit vector resulting in a size of $((H - h + 1) \times h) \times W \times \pi$ [TT09].

4.5 Deep Learning

We give a dense overview about DL and the ANNs used in this project, for a full explanation see [GBC16; Nie15].

DL is a further subdivision of ML, this specific learning type involves the use of ANNs. Besides the existence of different learning strategies, we focused on supervised ML. It involves the manual labeling of data, which is then transformed into a dataset where the connection between subject and features is given. So the problem is to approximate a function (the target function) $f : X \to Y$ that maps the features from some domain X to the subject in some codomain Y. As our goal is to distinguish multiple syllable types defining our classes, the features X and the subjects Y are vectors with the subject information being "one-hot" encoded. The "one-hot" encoding defines encoding by assigning the class to a position in a 1d vector where the vector for a subject has the value of one at the corresponding position and the other values are set to zero, see Figure 4.5.

Labels
A
E
I
O
U

→

A	E	I	O	U
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Figure 4.5: The "one-hot" encoding, the labels are replaced by a sparse feature vector, only the value at the corresponding position is one.

In our case, f is the function that maps an ultrasound recording slice to a syllable class, with the ANN models defined as a approximate function $\hat{f}(x, \Theta)$ parameterized by a finite-dimensional vector $\Theta^n \in \mathbb{Z}$ with some fixed n. The learning objective can be expressed as searching the values of the parameters Θ in a way that leads to the best possible function approximation. The function $\hat{f}$ defines our ANN models, which is a parametric method, are bound by a fixed order or type of operations and only the values of the parameters Θ are variable.
For complex networks, learning the optimal values of the parameters is not trivial and is generally worked out by defining a cost function and stochastic optimizer algorithm. The cost function should be adapted to the type of y (and therefore to the output of the model too) and measure the extent to which the parameterized function $\hat{f}(x, \Theta)$ deviates from the target function f in regard to the features from the dataset. A suitable cost function over all datapoints is the crossentropy defined in (4.2) describing the crossentropy between the training data y^n and the model distribution, which is equivalent to the negative log likelihood.

$$C = -\frac{1}{n} \sum_{i}^{n} \left(y_i \ln \hat{y} + (1 - y_i) \ln (1 - \hat{y}_i) \right) \tag{4.2}$$

The convention is that $0 \ln 0 = 0$ but $\hat{y} \ln 0 = \infty$ for $\hat{y} \neq 0$ makes the loss function perfect for binary classification tasks. For our multi-class classification task we took the categorical crossentropy loss function, it is the sum over C classes from the crossentropy between the "one-hot" encoded training data and one model prediction of the same length, where the model prediction is normalized by a prepended softmax layer.

$$L(y, \hat{y}\_softmax) = -\sum_{c=1}^{C} y_c \ln \hat{y}\_softmax_c$$

The softmax function is introduced in the next section 4.5.1.
A basic stochastic optimizer used for DL is the stochastic gradient descent (SGD) algorithm. This algorithm alters the model parameters by specific proportions in respect to the gradient value of the cost.

$$\Theta := \Theta - \eta \nabla J(\Theta)$$

The proportion is defined by the learning rate η, this hyperparameter should be carefully tuned as it affects time and quality of the solution it converges to. A more sophisticated type of optimizer is the adaptive moment estimation (ADAM) algorithm, which is the one we use for our experiments. Compared to the SGD, it provides an adaptive learning rate per parameter and uses moments of the stochastic gradients to calculate the resulting values. The moments lead to smaller changes in direction. One can imagine a ball entering from one side at the top of a valley and rolling downwards. Due to the built-in moment calculation, the ball rolls straight down the valley after some zigzag movements, as if some force tries to keep the ball in the middle of the valley. How much the forces of the moments influence the direction is controlled by the hyperparametres beta1 and beta2, which are the exponential decay rate of the first and second moment estimates. Although ADAM is controlled by three hyperparameters: learning rate, beta1, beta2; it is often sufficient to set the learning rate and the other two can be left at the default value. As the learning objective has not changed, we used the same parameters as the framework provided.
These two algorithms are used in conjunction with the backpropagation algorithm. For explanation, let us first consider the forward pass algorithm, which describes the first part of the training. The forward pass of a simple feedforward network with three layers can be described in a mathematically simplified way with the chain rule $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$, this implies that each layer is described by a function $f^{(d)}$. The first derivative of this function can be written out using the chain rule as follows $\frac{\partial f}{\partial x} = f^{(3)'}(f^{(2)}(f^{(1)}(x)) f^{(2)'}(f^{(1)}(x)) f^{(1)'}(x)$. It can be seen that the gradient calculation involves some repetitive calculations, e.g. $f^{(1)}(x)$. Such results are calculated and cached during the forward pass, this concept of caching intermediate results is called dynamic programming. Then for "learning" the parameters, the backpropagation algorithm populates the error computed by the loss function back to the input layer and solves the partial derivative $\frac{\partial C}{\partial p}$ layer-wise, which results in the parameter gradient in respect to the error. Subsequently, leveraged by the stochastic optimisation algorithm, the corresponding gradient is applied to each parameter.

4.5.1 Artificial neural networks

The ANN functions by imitating the neural connections made in the biological brain and connected these in a network of nodes, forming multiple layers. In the multi layer perceptron (MLP) architecture nodes are called perceptrons or neurons and are the computational units. The parameterized scalar valued function takes a set of inputs $x_1, \ldots, x_n$ and transforms the values by some parameters to the output y. The basic parameters are the weights $w_1, \ldots, w_n$, one for every input and the bias value b, through them one controls what the neuron computes. The processed input is given to an activation function (transfer function) σ which is commonly a threshold function.

$$y = \sigma \left(b + \sum_{i=1}^{n} x_i w_i \right) \tag{4.3}$$

The math in a neuron is defined in (4.3), which is an affine transformation of the input fed into an activation function.

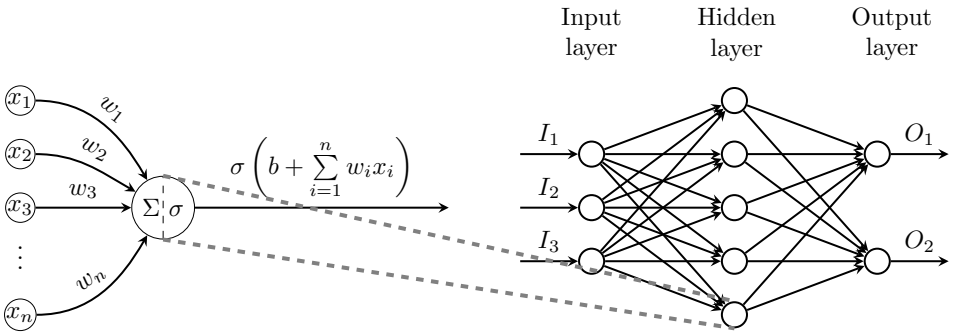

Figure 4.6: A diagram of a single perceptron, along with its position within a multilayer perceptron with fully connected layers.

The Figure 4.6 shows the structure of an MLP, the neurons are stacked in layers, with sparse to fully connected neurons between the layers. The number of neurons per layer can vary and activation functions can normally change per layer. Basically, called as the "feedforward ANN", the layers are connected in an order, starting with the input layer which digests the provided features. This is then followed by one or more hidden layers, hidden because their activation is hidden from the outside, and in the end there is finally the output layer (a.k.a. classification layer). The output of a neuron is digested by the neurons of the next layer sequentially, thus the activations are only passed on to the subsequent layers. This combination of neurons supports the aggregation of information from neurons specialised in the analysis of a single element to neurons at higher layers analysing more complex information.
The type of activation function and its respective abilities plays a crucial role for mapping complex functions, for example, a composition of linear functions is again a linear function, this implies that a MLP with only linear functions as activation functions will decay into a single layer network handling linear tasks. By selecting them consciously, they power the rise of the MLP networks by granting them the ability to learn non linear boundaries.

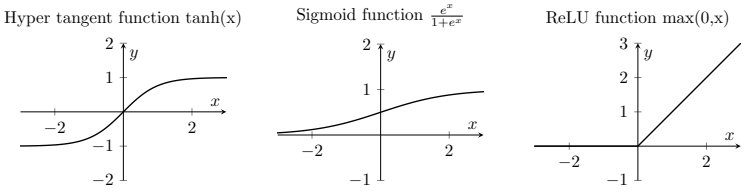

Figure 4.7: Graphs of some activation functions used in ANN.

One of the first class of functions commonly used were sigmoidal functions such as the logistic function $\sigma(x) = \frac{e^x}{1+e^x}$ and the hyperbolic tangent function tanh. In recent years the computationally cheaper rectifier function $\sigma(x) = max(0, x)$ called rectified linear unit (ReLU) has become the most used function for deep neural networks [LBH15]. This function has the advantage over the sigmoid function, that the weights for large values do not become insignificant e.g. changing them influences the input in a linear manner as long it is over the threshold. Because of this, the ReLU is seen as a possible approach to address the vanishing gradients problem when training deeper models. For multi-class problems, the softmax function is commonly used for the output layer. The normalising exponential function is defined as $\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$ for C classes, it is a generalization of the sigmoidal function for multiple dimensions. The calculated values can be seen as probabilities of how likely the network is to classify the input as one of the classes.

4.5.2 Long short term memory

LSTM architecture contains cells with a inherent state unit and the LSTM cells belong to the recurrent neural network (RNN).RNNs are specialized networks to analyze sequence data. The RNN cells hold a inner state, which influences the reception of the next value. Because the basic RNN cell does not control the alteration of the inner state, it tend to quickly "forget" information. The LSTM cell, as the name implies, exhibits memory gates that control the values that update the internal state, which increases the memorization ability.

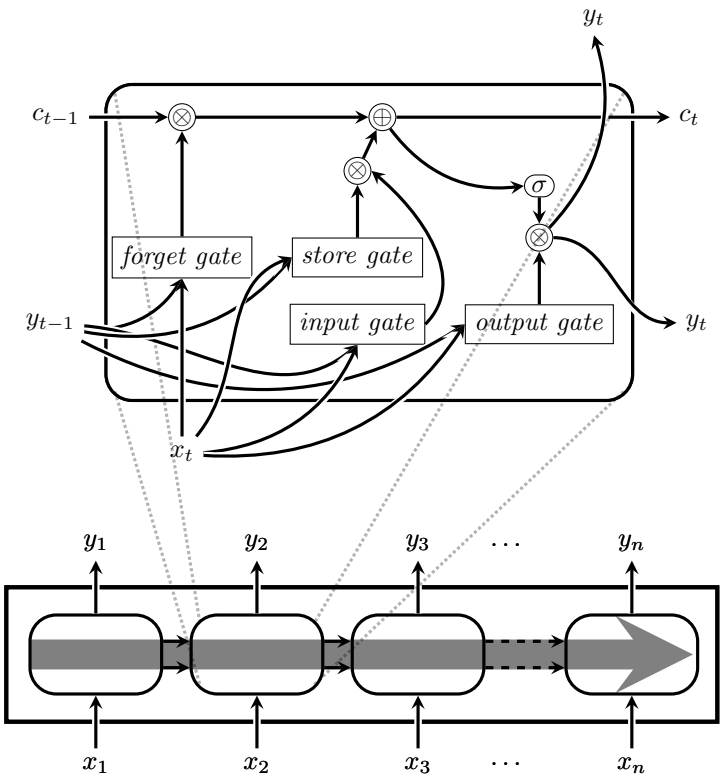

Figure 4.8: A diagram of a single LSTM cell and which values are recurrently used when processing a feature vector $x_1, \ldots, x_n$.

The upper diagram of the Figure 4.8 shows the inner parts of a typical LSTM cell. The previous output value and the new input value are both used by all gates to determine which part of the values should be changed. The gates function by squeezing the values between 0 and 1 (usually the transformation function is a sigmoidal one), the resulting output is then multiplied point-wise by the destination vector, resulting in a down-regulation of some elements in the destination vector. The forget gate, for example, controls what is overwritten (what should be forgotten) in the state value. The input gate is a special case because it is the same function as the one used by the activation function.
The lower diagram exibits how the cell processes an input vector $x_1, \ldots, x_n$, where the current state c_t and output y_t is used for analyzing the next element. An elaborated and easily understandable description of the LSTM cell is provided by Christofer Ola on his blog [Ola15].

4.5.3 Convolution neural networks

The CNNs are a kind of deep neural networks tailored for image-like data and inspired by the behaviour of the animal visual cortex. They consist of convolution layers, containing a tensor of filters, where the values of the filters are the trainable parameters. The filters are convolution kernels, a one or more dimensional matrix, which becomes convoluted over the input in a windowing manner. The output of the convolution layer is then sent through an activation function.

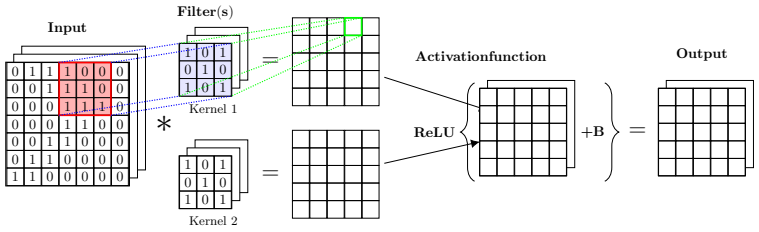

Figure 4.9: A diagram of a single convolution layer depicts the fate of the features through them.

Overall, a convolution layer is entirely defined by four hyperparameters; the number of kernels k, their spatial extent v, the strides s and padding p. Given an input of $[W_1, H_1, D_1]$, the corresponding convolution layer output $[W_2, H_2, D_2]$ relates to the input one as follows:

$$W_2 = \frac{W_1 - v - 2p}{s+1}; \quad H_2 = \frac{H_1 - v - 2p}{s+1}; \quad D_2 = k$$

In our experiments we stick to the already implemented variants of hyperparameters v=3, s=1 and p=1 which seems to be a common choice [Hua+17; SZ15].
Compared to a layer in the MLP, the neurons are quite similar, they are composed of weights, a bias and an activation function. The difference is, that the convolution neuron shares the activation function and only has a constrained set of weights per layer. This "constraint" allows the convolution layer to detect a characteristic, spatially local pattern in his receptive field, of the input data over the whole input at different places. Additionally, the stacking of convolution layers leads to non-linear filters that become responsive to larger regions of input space, so the networks creates representations of small parts (the receptive field) which are assembled to representations of larger areas.

4.5.4 DenseNet

The DenseNet takes advantage of the concept of additional connection by passing on layers, as other deep networks like ResNet. This idea stems from a problem which arises around deep networks; when continuously adding layers the performance saturates at some point. When adding more, e.g. increasing network depth beyond some point, it results in a significant drop in accuracy. This concept of additional skip connections allows networks to be employed with more than 100 layers in a efficient manner. In contrast to the skip connections from the ResNet, the DenseNet applies a dense connectivity pattern where the feature map of a layer l is passed to all $L - l$ subsequent layers. This is worked out block-wise by concatenating the output of all previous layers as input to the next layer. Where a layer is a combination of convolution layers and max pooling. As a direct consequence of the input concatenation, the feature maps learned by any of the DenseNet layers can be accessed by all subsequent layers per block. This encourages feature reuse throughout the network, and leads to more compact models [Hua+17]. Besides this, there are some evolving concepts incorporated, like bottleneck and full pre-activation [He+16b]. In addition, because of the skip connections, one can reduce the memory of the network with some nice tweaks like sharing the memory for outputs between the subsequent inputs in a dense block [Hua+].

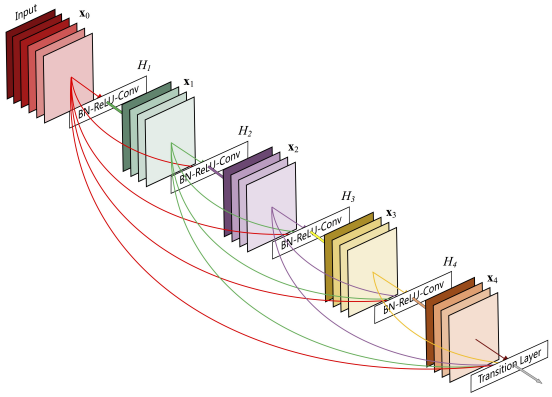

Figure 4.10: Densely connected convolutional network block with 4 weight layers and a growth rate of 4. (© [Hua+17])

4.5.5 Regularization

The ANN network architecture used in the experiments constitutes a complex computational model, which incorporates a high number of parameters. With the limited data we have, this makes the network prone to overfitting. Fortunately, the field of ANN has an arsenal of regularization techniques to prevent overfitting, we present the ones used in our models.
With Dropout, the ML model randomly drops units from the network during training, which corresponds to an activation of zero [Sri+14]. For every training example digested by the network the input or neurons are randomly dropped with probability p resulting in a thinned input and a thinned network. This forces the model to create alternate decision paths and hopefully simpler ones. The backpropagation of the training step then only concerns the parameters of the thinned network (respectively the retained neurons and inputs), which leads to the training of an ensemble of thinned networks. After a training step, when evaluating the model on the validation set, the full model without dropout is used. To account for the scaled weights of the neurons to which dropout is applied, the weights are multiplied by the probability $r = 1 - p$, i.e. the probability that they are retained in the network. This should grant that for any involved neurons the expected output at training time should be equally to the output at test time [Sri+14].
Max pooling has some regularization abilities, but this is not the main focus of this concept. The pooling layer is commonly combined with a convolution layer at the front and it subsamples its input tensors into a smaller output tensor. The subsampling is applied to every channel, therefore the resulting tensors only vary in width and height where the channel stays the same. The subsampling is done in a sliding window manner where the values of a window are sampled in respect to the pooling operation. Common pooling operations are max, average or L^2-norm and common hyperparameters are 2×2 for patch size and strides. One advantage of applying a pooling layer to a feature layer is that the features are invariant to small spatial shifts. This can be useful when detecting the presence of a pattern is more important than determining its exact position. Unfortunately, this nice feature leads to the loss of some information in the layer.

4.6 Evaluation

ANN can be difficult to understand, as their inner workings become, in a way, obfuscated by the large number of units. Compared to SVM or other similar ML models it could be near to impossible to explain the effect of a unit on the network, e.g. knowing which changes of the hyperparameters leads to which exact outcome. Therefore, we use tools for evaluating the models in a way that we can understand the inner working in a broader view. In this chapter we will introduce some methods to train and evaluate ANN models.

4.6.1 K-fold cross-validation

A model validation method is used for measuring the extent to which a model generalizes, e.g. how good it performs on unseen data. The generalisation can be assessed by comparing the predictions from the unseen data with the true output.

K-fold CV is a model validation method and is a general tool for selecting models on datasets with difficult-to-check assumptions on the data generating process [ZY15]. It is preferred over other methods when the dataset has limited samples, as it run the model on the whole data, with the trade-off that multiple runs are conducted per dataset and ML model. As the name of the method already implies, the experiment is executed k times. The data is evenly spread over k bins, with the shortest classes defining the bin size. The remaining samples are added to the training set. In every round the model is trained on $k - v - t$ bins plus the remaining samples, where v is the number of validation bins and t is the number of testing bins used. After each run, the bins are shifted by one, so that all subsets contain every bin at least once.

	Test	Validation	Training	
Experiment 1	1	2	3	4
Experiment 2	2	3	4	1
Experiment 3	3	4	1	2
Experiment 4	4	1	2	3

Figure 4.11: A visualisation of the distribution of bins in a k-fold CV with $k = 4$, using one bin each for validation and testing.

In the context of this method we speak about two different estimations made on unseen data. One is during model maturation where the unseen data belonging to the validation set is used for picking the model in the best state. The other (belonging to testing) provides a fair assessment on the maturated model, after the test, no changes on the (hyper)parameters should be made anymore.
An additional input about model generalisation is provide by Juan Gabriel Colonna et al. [CGN16]. They propose to include the species and specimen information at the split process in a way, that syllables of one specimen are in test or validation only. This enables the assessment of how well the model is generalized for detecting syllables of new specimen of the same species. As we are interested in the performance on different feature presentations, we omitted this kind of preparation. In a future work, the integration of species and specimen information would become interesting!
In our experiments, the samples were stratified by their labels and then distributed evenly to the bins depending on their input order, in contrast to the basic concept where the samples are distributed in a randomized manner. We chose this concept because it ensures that samples of the same audio are more likely in the same bin than distributed over all bins, which should result in a more accurate estimation on how the model could generalize as the validation and test bins are less likely to contain a sample of the same audio as a sample exiting in the training bins. However, due to this decision, we loose the nice feature of randomization, which helps to break preexisting order of datapoints and ensures a similar distribution as possible.

4.6.2 Learning curves

We already showed how the training is done in this project, but how do we assess this? And how is this used to select the interesting model, i.e. parameter set?

During the training phase of a model one can monitor the loss and accuracy of the ML model on the training set and the validation set. The resulting visualisation graphs of loss or accuracy over time belong to the learning curves, they are widely used as diagnostic tools for ANNs, as they learn from the data set in an incremental manner.
The training curves provide information about how well and quickly the model can learn from the data. If the model is too simple (e.g. small ANN) the model tends to struggle with the learning data, called underfitting. As the model is unable to capture the patterns in the training data, the accuracy plateaus at low magnitude and the loss remains relatively high.

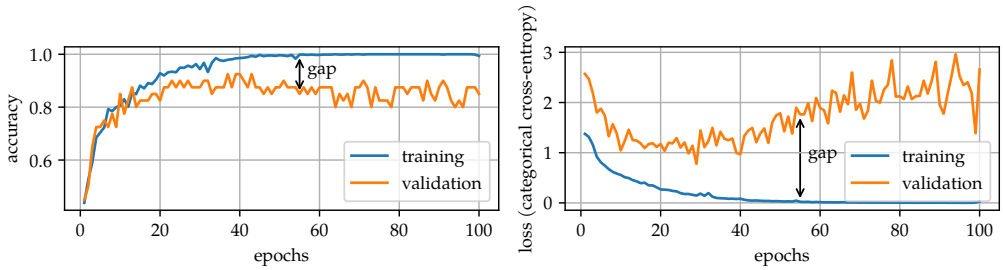

Figure 4.12: Visualisation of the training curves exibiting overfitting, with the accuracy and loss curves merged into one plot. The validation loss draws a U-shaped curve with a increasing gap to the training loss and the training accuracy is about 10% higher than the validation accuracy.

By considering all four graphs, one can extract information about overfitting and how the validation data is represented by the testing data and vice versa. Overfitting becomes visible by plotting both accuracy graphs together. Where over time the gap between the greater training accuracy and lower validation accuracy becomes visible. Commonly, this gap increases over time, as the model starts to over optimize more and more to fit to the training data. The overfitting behaviour is also visible when comparing the two loss curves, with the training loss curve getting steadily lower and the validation loss curve initially decreasing until it starts to increase, sometimes exponentially, resulting in the typical U-shaped curve.
Consider an accuracy and loss graph from the validation data, one option would be to select the epoch with the lowest loss value, but as we are more interested in the accuracy and the loss value is typically only used as a proxy for the accuracy, the better choice is to select the model where the validation accuracy stops improving. But when does accuracy stop improving? One can say this happens when the accuracy starts to plateau the first time or starts fluctuating around the same value for some epochs. It could happen that if we train over many more epochs after the accuracy starts to plateau, that it increases again, but the magnitude would likely be small and therefore be negligible [Nie15]. As this seems to be a valid point, there exists a different view; Goodfellow et al. suggest in the book deep learning [GBC16] to select the model with the lowest validation loss, as the generalization is then typically maximized.
Therefore, we tend to take one snapshot of the epoch with the lowest validation loss and one with the highest validation accuracy. Finally, in the results, we stick to the second metric because of our interest in comparing the accuracy.

4.6.3 Layer wise relevance propagation

For image-like features, there is a well-suited technique called layer-wise relevance propagation (LRP). After a forward pass, the algorithm searches neurons that contribute most to the result backwards through the

network. This is done by examining the weights and activations from the forward pass. The contribution of a neuron is significant if it has a high activation value and it contributes to a lot to relevant neurons in the deeper layers. The result can then be visualised as a heat map where the areas with high pixel values are the most important for getting a certain result. Unfortunately, the python framework ”innvestigate” used to explore the DL model was not fully supported by the TensorFlow version used. Therefore some changes were made with insufficient knowledge about the code of the framework in question, in the hope that they would have limited damage on the results. Thus, when interpreting the results they should be taken with a pinch of salt.

5 Experiments

At first we provide a brief intro about our setup, including the used software, preprocessing, DL models, and applied parameters. Secondly we cover the experiments, which are divided into two parts:

— **Test experiments**: We try to understand how in general the DL models could classify only the syllables, without additional background samples.

— **Seq experiments**: We investigate the DL models' performance on detecting syllables by providing the whole audio as slices in a moving window manner.

5.1 Main setup

This chapter contains the general experiment set-up. Specialisations are described in the additional subchapters in the individual experiment chapters if required.

5.1.1 Software

We adapted a piece of software created with the bachelor thesis "Bird Voice Deep Learning" [Wae19] kindly provided by Gilles Waeber. Apart from needing a similar pipeline and DL models to conduct our experiments, it provides some interesting features like interruption awareness and extended logging of the results in an accessible manner.

5.1.1.1 Changes

We removed the birdvoice_server section of the code and everything else needed to run the web application. The Docker files we kept back, but they would first have to be adapted to the new folder structure. Since UBELIX, the HPC cluster at the University of Bern, does not directly support docker, we did not further invest in it. Moreover, due to the possibility of installing our custom software on UBELIX, we no longer needed this or similar options.
Then we updated to TensorFlow 2.3, as with this version we could use the Ragged Tensors for variable length input data. This allowed us to train the LSTM model through the convenient training method from Keras-Model including batch and graphics processing unit (GPU) support. Therefore, we could ignore all the specialized parts of the software written for training the RNN models with variable length input.

For portability reasons, all file paths are made relative to the data path and converted to slash delimited paths which are written to reports or similar reused data files.
We used the jupyter notebook library for generating reports about the experiment testing results. The reports are generated by a template which can be customized as needed.

5.1.1.2 Dependencies

Over time and with the advent of new products or simply through convenience, the dependencies have changed a little. The changes are presented below:

— Python 3.8, although unfortunately we do not make direct use of new features such as the walrus operator.

— TensorFlow, this version enables faster LSTM training, as it fully supports the concept of Ragged Tensor.

— innvestigateupdates_towards_tf2.0, with some additional modifications, we could use the "innvestigate" library in conjunction with TensorFlow 2.3 for generating the LRP headmaps by the deeptalyor algo.

— Wavinfo 1.6.2, is used for extracting the label information produced by the avisoft software.

Below are the relevant python libraries:

— **TensorFlow GPU 2.3.1**, machine learning backend, Apache 2.0
`https://www.tensorflow.org/`
`https://github.com/tensorflow/tensorflow/blob/master/LICENSE`

— **Numpy 1.19.1**, fast numeric computation, BSD-3-Clause
`https://numpy.org/`
`https://github.com/numpy/numpy/blob/master/LICENSE.txt`

— **TQDM 4.50.2**, progress bars, MPL 2.0
`https://github.com/tqdm/tqdm`
`https://github.com/tqdm/tqdm/blob/master/LICENCE`

— **SciPy 1.5.3**, linear algebra, BSD-3-Clause
`https://scipy.org/`
`https://github.com/scipy/scipy/blob/master/LICENSE.txt`

— **Matplotlib 3.3.1**, 2D plotting, PSF
`https://matplotlib.org/`
`https://matplotlib.org/users/license.html`

— **FileLock 3.0.12**, file locking, Unlicense
`https://github.com/benediktschmitt/py-filelock`
`https://github.com/benediktschmitt/py-filelock/blob/master/LICENSE`

— **Jupyter Notebook 4.6.3**, interactive notebooks, BSD-3-Clause
`https://jupyter-notebook.readthedocs.io/`
`https://github.com/jupyter/notebook/blob/master/LICENSE`

— **Librosa 0.8.0**, manipulate audio, ISC License
`https://librosa.org/doc`
`https://github.com/librosa/librosa/blob/main/LICENSE.md`

— **Wavinfo 1.6.2**, read wav labels, MIT License
`https://wavinfo.readthedocs.io`
`https://github.com/iluvcapra/wavinfo/blob/master/LICENSE`

5.1.2 Spectrogram

— The spectrograms have been created with SoX using the audio recordings.

— Sampling rate is set to 500 kHz.

— Height is variable, no band-pass filtering (0-250 kHz).

— All other parameters vary per experiments.

5.1.3 Samples

— The models were trained by k-fold cross-validation with k = 8, validation bins 1 and testing bins 2.

— The samples are distributed among the bins so that each bin contains the same number of samples for each class. The remaining samples are always used for training and therefore do not change between runs.

5.1.4 Features

— Raw stands for pixel features from the spectrogram scaled between 0 and 1.

— For HOG features, the descriptors are extracted by HWRecog (without preprocessing) using the Slit-Style HOG variant.

— The HOG variant suffixed with 3D, the 12 bins of the HOG-descriptor are populated to 12 channels/layers of the image.

— The values are normalized for each sample between -1 and 1.

5.1.5 LSTM Model

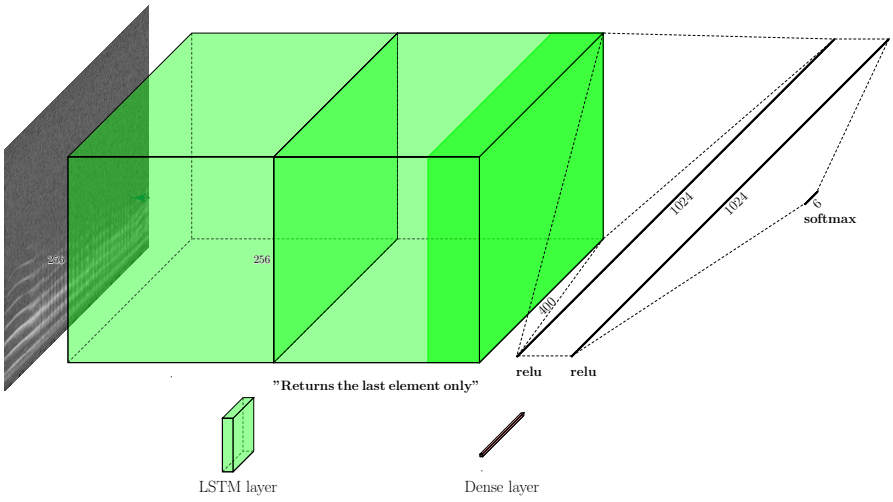

Figure 5.1: Visualisation of the LSTM architecture. The size of the elements does not correspond exactly to their real dimension numbers.

The Figure 5.1 represents the LSTM architecture used for the experiments. Form left to right:

— Input layer, it provides raw or HOG features in various sizes.

— First LSTM layer with 265 neurons, it transmits the whole output sequence to the next layer.

— Second LSTM layer with 265 neurons. Only the output of the last element is transmitted.

— Two fully connected layers with 1024 neurons and a dropout rate of 50%. Both use the ReLU activation function.

— The final classification layer is a fully connected layer with softmax activation.

5.1.6 CNN Model

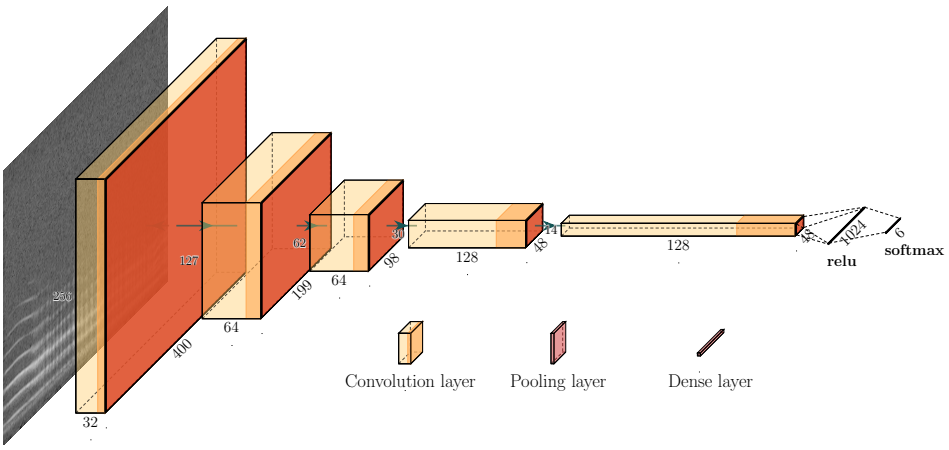

Figure 5.2: Visualisation of the CNN architecture. The size of the elements do not correspond exactly to their real dimension numbers.

The Figure 5.2 represents the CNN architecture used for the experiments. Where the CNN 1D model uses 1 dimensional convolution layers. From left to right:

— Input layer, it provides raw or HOG features in various sizes.

— The next five blocks consist of a convolution layer followed by a max pooling layer with ReLU activation function. The size and stride of the pooling layer are 2×2, resulting in dividing the size of the volume and the number of feature maps in half. From the second block onwards the next block is only created if the output shape of the previous block is big enough. The filter configurations of the convolution layer in the blocks are as follows:

1. $32 \times 3 \times 3$
2. $64 \times 3 \times 3$
3. $64 \times 3 \times 3$
4. $128 \times 3 \times 3$
5. $256 \times 3 \times 3$

— The output of the previous max pooling layer is flattened (sample dimension reduced to one) and transmitted to a fully connected layer with 1024 neurons, a dropout rate of 50% and a ReLU activation function.

— The final layer, also called the classification layer, is a fully connected layer with the same number of neurons as classes. The classification layer uses the softmax activation function.

5.1.7 DenseNet Model

We use the DenseNet-121 architecture as described here [Hua+17] followed by a a fully connected layer with 1024 neurons, a dropout rate of 50% and a ReLU activation function. The final classification layer uses the softmax activation function.

5.2 Test experiments

The general aim of these experiments is to understand how the models performs on different preprocessed syllable samples. Therefore, we only altered the preprocessing hyperparameters for the different tests. However, one should be aware that the glsdl models are automatically adjusted to the size of the input data, which leads to a change in the number of parameters of a glsdl model. Whenever possible, we tested the resulting features on all DL models.

5.2.1 Specialized setup

For all test experiments, the source of the data remained the same. Unfortunately, constraint by the spectrogram resolution and feature type applied, we had to exclude some syllable types in some experiments.

5.2.1.1 Cutting the syllables

The syllable parts are extracted from the audio source based on their start and end times, the SOX software was used for this.

5.2.1.2 Silent profile and noise reduction

We generated one silent file per audio file group, from this we created the noise profile and applied it to the extracted syllable audio file with the effect controlled by the sensitivity parameter called noise reduction sensitivity (nrs). How we defined the silent parts of an audio source is described in Section 4.3.1.
For the manual labelling, the original audio recording was cut into smaller pieces, whereby sub strings from the file name were changed or extended. We tried to reverse this process as it was sometimes difficult to extract background noise from the edited audio files at all. The audio file group is defined by a pattern in the file name of the audio files. The extracted pattern is used as the key for grouping the audio files and generating the silent audio file name.

5.2.2 Compressed

The goal for this experiment was to evaluate how well the DL models could learn to classify the syllables when they are compressed with respect to the time axes. Additionally, this should give us a sort of baseline of what we should achieve in future optimized runs.

5.2.2.1 Setup

The raw and HOG features were prepared as follows:

— Because the version of sox has a minimum fixed width length of 100 pixel, we had to filter out the syllables that were shorter than $22\,\mathrm{ms}$. This value is $2\,\mathrm{ms}$ more than should theoretically be possible with sox, since sox has a maximum resolution of 5000 pixels per second, which would allow a minimum length of $20\,\mathrm{ms}$. However, with samples below $22\,\mathrm{ms}$, sox sometimes fails to create the spectrograms in the correct size after noise reduction is applied.

— Additionally after filtering the dataset in length, we kept all syllable types with a minimum sample count of 50 samples. This resulted in a smaller dataset, where the ratio between the shortest and the longest syllable sample is around $1 : 9$.

— The syllables were extracted.

— A noise reduction filter with sensitivity of 6%, 12% and 21% or no noise reduction was applied to the extracted syllables.

— Finally, the samples were converted into 100×256 spectrogram images. Compared to the shortest syllable sample, the resulting compression ratio for the largest sample with a duration of $200.4\,\mathrm{ms}$ is about $1 : 9$. The resulting visible change can be seen in the Figure 5.3.

syllable type	mean/ms	median/ms	min/ms	max/ms	count
B4	43.56 ± 14.05	39.57	23.54	73.19	82
UPS	45.45 ± 11.13	44.07	23.04	85.59	461
B3	54.67 ± 17.76	53.53	26.19	120.70	124
B2	62.26 ± 31.28	54.50	22.19	200.40	288

Table 5.1: Descriptive statistic of the syllables used in the compressed experiment.

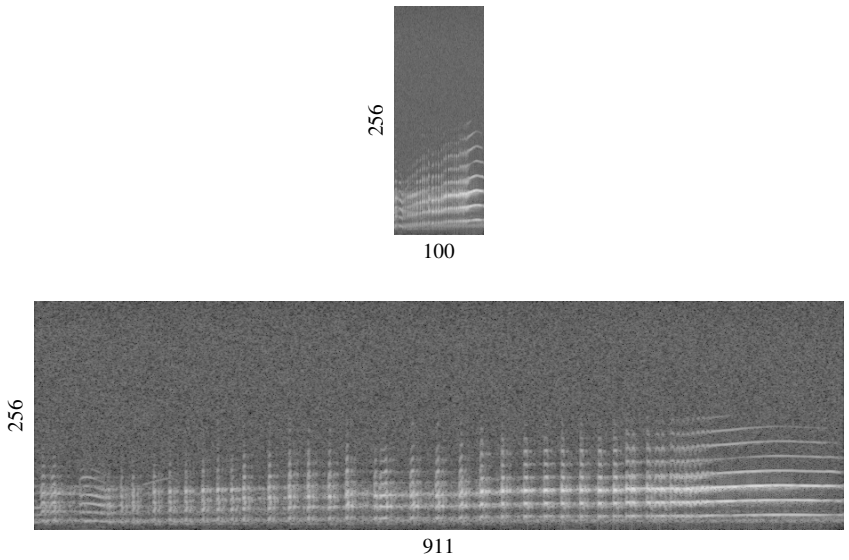

Figure 5.3: Visualisation of a compression ratio of 1:9 on the basis of the longest syllable sample.

As shown in the Figure 5.4, the bins contain 10 samples because the smallest set of syllables had only 82 samples. For the syllable type with the most samples, only around 17.35% of the samples are rotated in the k-fold process. The samples for the bins are taken from the beginning of the data set.

5.2.2.2 Results

According to the Table 5.2, the DenseNet and the CNN 2D models perform similar, however the first has a slightly better test accuracy. Interesting is, that the CNN models overcome the visual distortion through the noise reduction and achieve almost similar test accuracy than without. The LSTM model becomes even better on the highest nrs value when applied to raw features. However, it should be taken into account that the LSTM model performs much better on the HOG features. And therefore the result about the higher accuracy at high nrs in raw values is not further relevant for our goal.
Overall, the Figure A.1 shows that the CNN models perform better than the LSTM model in terms of raw features. The opposite is true for the HOG features, but the accuracy is generally lower there. The effect

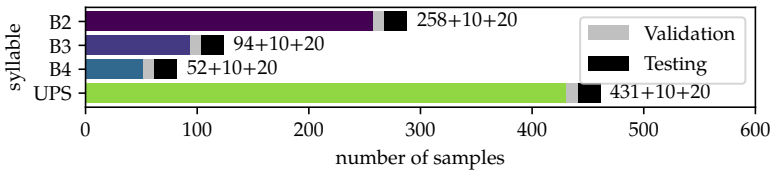

Figure 5.4: The distribution of the samples used for the compressed test experiment. For all syllables we used 10 samples for validation and 20 for testing. The dataset is not balanced with a ratio of about 1 : 8 in the number of training samples between the least and most represented syllable.

of noise reduction is opposite for the CNN models and the LSTM model at the raw features. However, the LSTM model with HOG features behaves similar to the CNN model with raw features in respect to noise reduction.

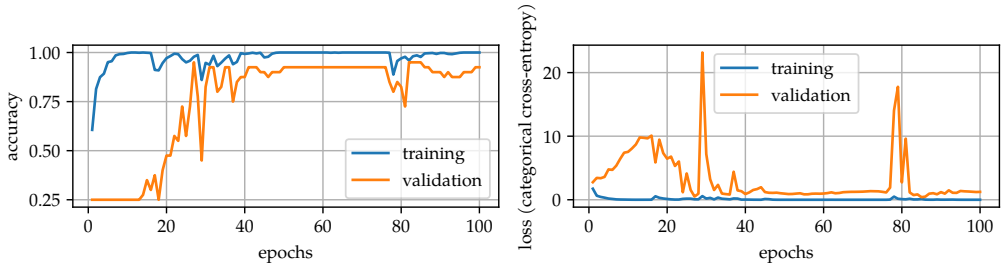

Figure 5.5: Training progression for DenseNet model [M1] on raw features.

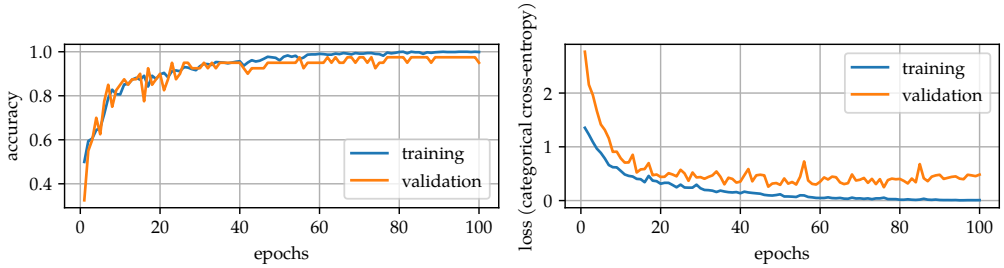

Figure 5.6: Training progression for CNN 2D model [M2] on raw features.

Gazing at the learning curves in Figure 5.5 and Figure 5.6, the best models show an acceptable learning curve with little to no overfitting. However it can be seen that the DL models have difficulty in learning patterns and selecting features that accurately describe the samples in all datasets and tend to overfit. This is revealed by the rather large difference between validation and test accuracy in the result Table 5.2. With the exception of the 2 best models, all others tend to overfit to a greater extent. This is exemplified in the Figure 5.8, where the validation loss curve shows a vague U shape and the cap between validation and test accuracy grows until the end.

Even though the DenseNet model performs quite well on the noise-reduced features compared to the other models, it clearly has difficulty learning general features of the syllables compared to the raw features without noise reduction. This is reflected in the continuous gap between the test and validation accuracy and the loss learning curves, see Figure 5.8.

The confusion matrices of the best models per type display difficulties in differentiating the syllables B2, B3

Model	Validate Accuracy (%)	Loss	Test Accuracy (%)	Loss
[M1] DenseNet, nrs: 0, raw	93.4 ± 4.2	1.23 ± 0.75	87.5 ± 2.8	0.65 ± 0.29
[M2] CNN 2D, nrs: 0, raw	92.2 ± 5.1	0.97 ± 0.60	87.0 ± 4.4	0.43 ± 0.18
[M3] CNN 2D, nrs: 12, raw	91.6 ± 5.2	0.80 ± 0.42	85.6 ± 6.0	0.41 ± 0.14
[M4] CNN 2D, nrs: 6, raw	91.2 ± 5.5	0.87 ± 0.43	84.5 ± 5.5	0.38 ± 0.14
[M5] DenseNet, nrs: 12, raw	90.9 ± 5.2	1.51 ± 1.26	84.4 ± 6.5	0.72 ± 0.42
[M6] CNN 2D, nrs: 24, raw	90.6 ± 8.0	0.82 ± 0.53	84.1 ± 5.7	0.46 ± 0.20
[M7] DenseNet, nrs: 24, raw	90.6 ± 5.0	1.13 ± 0.57	83.0 ± 4.5	0.71 ± 0.30
[M8] CNN 1D, nrs: 0, raw	90.6 ± 4.4	1.02 ± 0.36	82.7 ± 4.7	0.49 ± 0.08
[M9] DenseNet, nrs: 6, raw	91.9 ± 5.5	1.00 ± 0.54	81.2 ± 5.6	0.66 ± 0.31
[M10] CNN 1D, nrs: 24, raw	87.5 ± 5.7	1.18 ± 0.57	81.2 ± 6.1	0.63 ± 0.27
[M11] CNN 1D, nrs: 6, raw	87.5 ± 4.0	1.25 ± 0.48	79.2 ± 2.7	0.61 ± 0.11
[M12] CNN 2D, nrs: 0, HOG	86.6 ± 5.5	1.30 ± 1.03	79.2 ± 4.6	0.56 ± 0.14
[M13] CNN 1D, nrs: 12, raw	86.2 ± 5.2	1.45 ± 0.62	78.4 ± 4.3	0.68 ± 0.15
[M14] CNN 2D, nrs: 0, HOG 3D	83.8 ± 3.3	1.08 ± 0.22	77.7 ± 3.7	0.54 ± 0.05
[M15] CNN 1D, nrs: 0, HOG	85.6 ± 3.5	1.08 ± 0.27	76.9 ± 5.1	0.57 ± 0.06
[M16] LSTM, nrs: 0, HOG	89.7 ± 4.9	1.24 ± 0.39	76.2 ± 2.6	0.82 ± 0.34
[M17] DenseNet, nrs: 0, HOG	87.8 ± 2.5	1.69 ± 1.11	75.8 ± 4.4	1.20 ± 0.21
[M18] LSTM, nrs: 6, HOG	84.1 ± 7.3	1.78 ± 1.10	75.5 ± 4.9	0.84 ± 0.23
[M19] CNN 2D, nrs: 6, HOG	82.8 ± 2.1	1.61 ± 0.98	75.3 ± 6.0	0.81 ± 0.30
[M20] LSTM, nrs: 12, HOG	85.3 ± 5.7	1.61 ± 1.17	73.3 ± 6.0	0.94 ± 0.37
[M21] CNN 2D, nrs: 24, HOG	80.9 ± 6.1	1.50 ± 0.65	73.3 ± 9.7	0.76 ± 0.22
[M22] DenseNet, nrs: 6, HOG	83.1 ± 4.2	3.12 ± 2.26	72.8 ± 6.2	1.63 ± 0.62
[M23] CNN 2D, nrs: 12, HOG	82.5 ± 4.0	1.22 ± 0.42	72.3 ± 8.0	0.75 ± 0.18
[M24] CNN 1D, nrs: 12, HOG	78.4 ± 6.5	1.52 ± 0.58	72.2 ± 9.8	0.76 ± 0.24
[M25] CNN 2D, nrs: 12, HOG 3D	80.6 ± 5.8	1.30 ± 0.46	72.0 ± 6.9	0.71 ± 0.16
[M26] CNN 1D, nrs: 6, HOG	79.7 ± 8.3	1.40 ± 0.55	71.9 ± 7.1	0.77 ± 0.17
[M27] LSTM, nrs: 12, raw	79.1 ± 5.5	3.84 ± 2.41	71.9 ± 7.2	1.40 ± 0.55
[M28] LSTM, nrs: 24, HOG	82.8 ± 6.7	1.87 ± 1.25	71.7 ± 8.2	0.98 ± 0.27
[M29] DenseNet, nrs: 12, HOG	83.1 ± 5.1	3.62 ± 1.49	71.4 ± 4.5	1.90 ± 0.38
[M30] CNN 2D, nrs: 6, HOG 3D	79.7 ± 4.9	1.57 ± 0.48	71.1 ± 6.9	0.74 ± 0.18
[M31] LSTM, nrs: 24, raw	79.1 ± 6.4	2.76 ± 1.09	70.6 ± 5.6	1.22 ± 0.41
[M32] DenseNet, nrs: 24, HOG	77.2 ± 6.3	3.83 ± 1.77	68.9 ± 4.9	1.77 ± 0.28
[M33] LSTM, nrs: 6, raw	76.6 ± 5.2	3.43 ± 1.81	68.4 ± 6.9	1.26 ± 0.63
[M34] CNN 2D, nrs: 24, HOG 3D	78.1 ± 6.4	1.53 ± 0.49	68.0 ± 10.2	0.79 ± 0.22
[M35] CNN 1D, nrs: 24, HOG	78.1 ± 6.6	1.43 ± 0.51	68.0 ± 8.8	0.89 ± 0.24
[M36] LSTM, nrs: 0, raw	77.2 ± 4.5	2.72 ± 1.27	67.7 ± 2.3	1.23 ± 0.52

Table 5.2: Results of the compressed images experiment, sorted by test accuracy.

and B4, see Figure 5.9. The most common error is that the syllable B2 is falsely predicted as B3 or B4.

5.2.3 Variable length

In this experiment, our goal was to measure the performance of the DL models on spectrogram images which are variable in length, equal to the duration of the syllable. Since our CNN models are not able to process variable-length input, this experiment is conducted with the LSTM model only. A side interest was whether and how much the performance is influenced by changes in frequency and time resolution of the spectrogram.

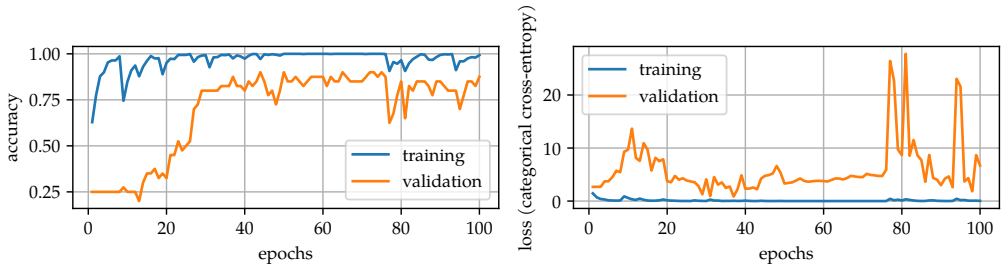

Figure 5.7: Training progression for DenseNet model [M5] on by 12% noise reduced raw features.

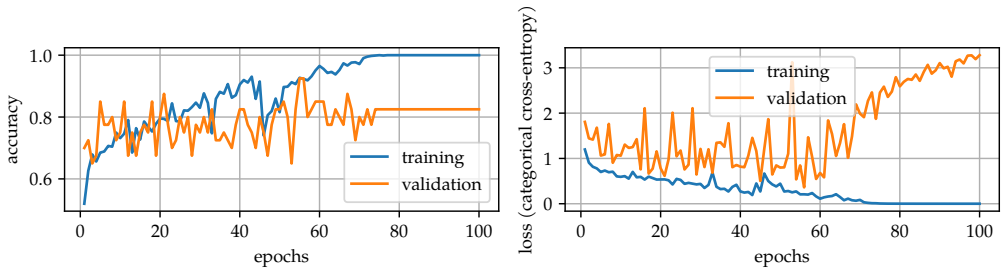

Figure 5.8: Training progression for CNN 2D model [M16] on raw features.

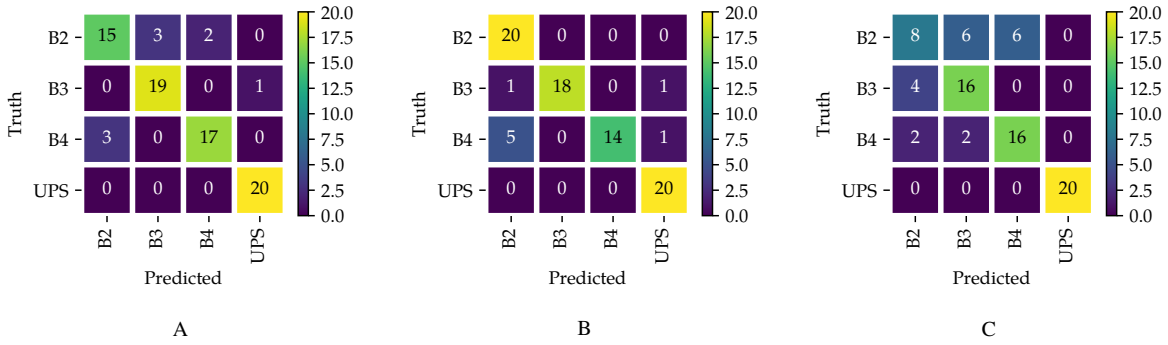

Figure 5.9: Confusion matrices from some models of the compressed test experiment performing on the same test set: (A) DenseNet model [M1], (B) CNN 2D model [M2], (C) LSTM model [M16].

5.2.3.1 Setup

The syllables were extracted and converted into spectrogram images with variable width. We varied the frequency resolution between 256, 300 and 512 pixels for frequencies up to 250'000Hz and the time resolution by 2000 and 5000 horizontal (x) pixels per second (xpps). In this experiment we used all syllables, information about the dataset is provided in Section 3.2.

5.2.3.2 Results

In the Table 5.3 we see that LSTM achieves much better results with the HOG features. Looking only at the results of the HOG features, the relatively high, resolution leads to only minor differences across all runs and does not show a clear pattern in any direction. Compared to the compressed experiment, the best model in this experiment [M38] yields slightly better test accuracy than the previously best CNN model

Model	Validate Accuracy (%)	Loss	Test Accuracy (%)	Loss
[M37] LSTM, xpps: 4K, height: 300, HOG	93.4 ± 2.1	0.66 ± 0.17	88.7 ± 2.4	0.44 ± 0.17
[M38] LSTM, xpps: 5K, height: 512, HOG	93.0 ± 2.9	0.59 ± 0.21	88.5 ± 1.8	0.40 ± 0.11
[M39] LSTM, xpps: 5K, height: 256, HOG	91.9 ± 1.4	0.70 ± 0.25	88.2 ± 1.7	0.42 ± 0.11
[M40] LSTM, xpps: 4K, height: 512, HOG	93.0 ± 2.7	0.51 ± 0.20	88.1 ± 3.4	0.42 ± 0.13
[M41] LSTM, xpps: 5K, height: 300, HOG	93.4 ± 1.8	0.63 ± 0.31	87.0 ± 2.4	0.46 ± 0.12
[M42] LSTM, xpps: 4K, height: 256, HOG	92.4 ± 2.6	0.72 ± 0.29	86.6 ± 2.9	0.52 ± 0.18
[M43] LSTM, xpps: 2K, height: 256, raw	78.2 ± 4.7	3.07 ± 2.36	70.5 ± 3.8	1.50 ± 0.53
[M44] LSTM, xpps: 4K, height: 300, raw	73.1 ± 3.2	3.73 ± 1.25	68.7 ± 1.9	1.68 ± 0.37
[M45] LSTM, xpps: 4K, height: 256, raw	72.2 ± 2.4	4.97 ± 1.97	67.2 ± 2.7	2.05 ± 0.22
[M46] LSTM, xpps: 5K, height: 300, raw	72.0 ± 2.8	4.11 ± 2.29	66.7 ± 1.6	1.76 ± 0.44
[M47] LSTM, xpps: 5K, height: 256, raw	72.0 ± 1.8	3.58 ± 1.91	65.2 ± 2.9	1.62 ± 0.45
[M48] LSTM, xpps: 4K, height: 512, raw	72.5 ± 3.0	3.63 ± 1.41	65.0 ± 3.7	1.82 ± 0.45
[M49] LSTM, xpps: 5K, height: 512, raw	69.7 ± 2.0	4.04 ± 1.91	64.5 ± 3.0	1.82 ± 0.76

Table 5.3: Results of the variable length experiment, sorted in descending order by test accuracy.

[M1]. On the contrary, for the raw features, the higher resolution leads to worsened results. The model with the highest frequency and time resolution [M49] yields a lower test accuracy than every model in the compressed experiment.

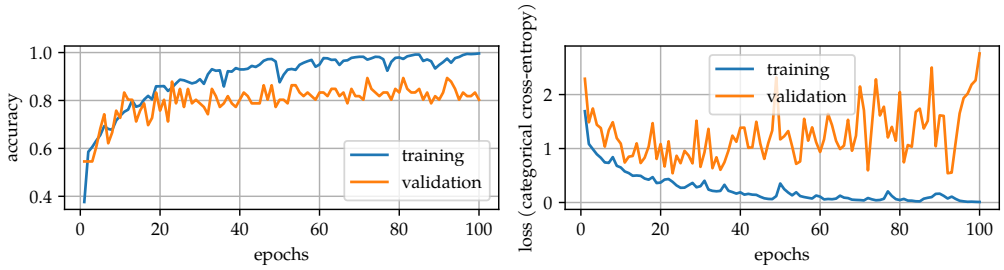

Figure 5.10: Training progression for LSTM on HOG features.

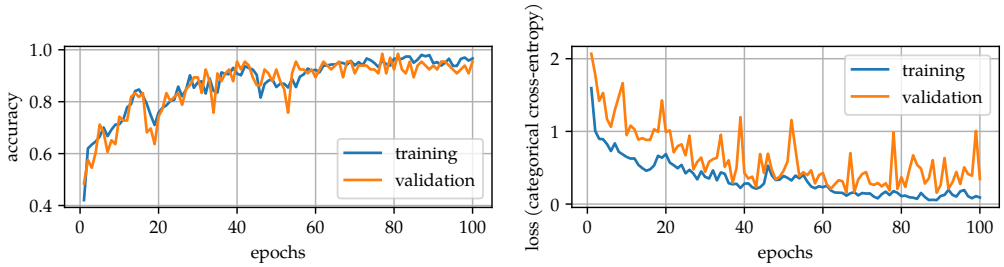

Figure 5.11: Training progression for LSTM on HOG features at high resolution.

In the Figure 5.10 we see that the model [M37] tends to overfit, as sometimes the gap between training and test accuracy is relatively large and the loss becomes too great. However, the model seems to be able to readjust the network in a fluctuating manner. For the higher resolution features, the model has a much lower tendency to overfit, but the learning curve is less steep, see Figure 5.11. Furthermore, it can be seen from the

accuracy curve, which rises slightly towards the end, that 100 epochs are too short to learn on this data. This indicates that higher resolution could lead to a better performance with more training time.
Similar to the compressed test experiment, the confusion matrix reveals the difficulties for the models in differentiating the B2, B3 and B4 syllables, see Figure 5.12. This time B2 and B4 are mostly misclassified. To a smaller degree the VS and VSV syllables are misclassified.

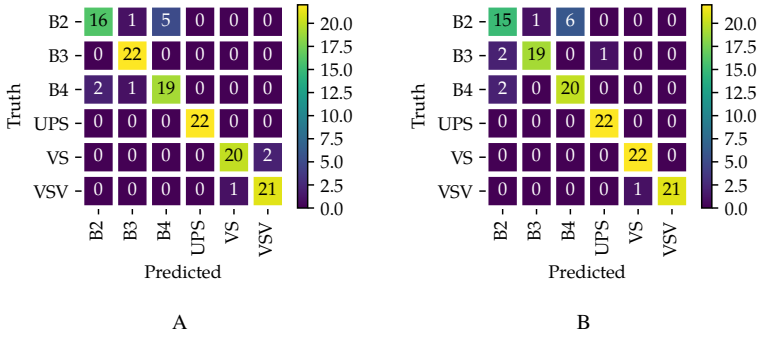

Figure 5.12: Confusion matrices of some models of the variable length test experiment performing on a test set: (A) best model [M37], (B) high resolution [M38].

5.2.4 Padded

This experiment is similar to the previous one, but the extracted syllables are left padded with silent parts. Therefore, we could test all our DL models on this setup. The main goal is to evaluate how well the DL models can focus on the syllable parts and not be disturbed by the padded part.

5.2.4.1 Setup

Following the preparation of the audio files, as described in the setup of the variable length experiment in Section 5.2.3.1, the audio files were left padded with silent parts of the same audio source during the syllable extraction process (how we generated the silent audio parts is described in Section 4.3.1). We applied either no noise reduction or noise reduction with sensitivity of 6%. The spectrogram images are then created in two resolutions: 2000xpps/256pixel and 5000xpps/512pixel.

5.2.4.2 Results

Model	Validate Accuracy (%)	Loss	Test Accuracy (%)	Loss
[M50] DenseNet, nrs: 0, xpps: 2K, height: 256, raw	93.4 ± 4.4	1.68 ± 2.82	88.4 ± 3.5	0.62 ± 0.31
[M51] LSTM, nrs: 0, xpps: 5K, height: 512, HOG	91.9 ± 1.6	0.76 ± 0.13	87.1 ± 2.1	0.45 ± 0.11
[M52] DenseNet, nrs: 6, xpps: 2K, height: 256, raw	91.5 ± 4.9	1.09 ± 0.84	87.1 ± 4.4	0.81 ± 0.43
[M53] DenseNet, nrs: 0, xpps: 2K, height: 256, HOG	89.8 ± 4.6	1.50 ± 0.91	85.0 ± 4.8	0.88 ± 0.41
[M54] LSTM, nrs: 6, xpps: 5K, height: 512, HOG	89.6 ± 3.7	0.87 ± 0.22	84.5 ± 4.6	0.59 ± 0.19
[M55] LSTM, nrs: 0, xpps: 2K, height: 256, HOG	89.2 ± 4.8	1.30 ± 0.78	83.8 ± 2.3	0.74 ± 0.19
[M56] CNN 2D, nrs: 6, xpps: 5K, height: 512, raw	86.2 ± 4.8	1.84 ± 1.22	80.8 ± 2.7	0.99 ± 0.44
[M57] LSTM, nrs: 6, xpps: 2K, height: 256, HOG	85.8 ± 4.0	1.17 ± 0.53	80.3 ± 3.9	0.72 ± 0.11
[M58] CNN 1D, nrs: 0, xpps: 5K, height: 512, raw	85.4 ± 3.9	1.42 ± 0.49	80.3 ± 3.7	0.73 ± 0.13
[M59] CNN 2D, nrs: 0, xpps: 5K, height: 512, raw	86.2 ± 4.9	2.31 ± 1.40	80.0 ± 3.2	1.06 ± 0.47
[M60] DenseNet, nrs: 6, xpps: 2K, height: 256, HOG	86.0 ± 4.4	1.93 ± 1.30	79.7 ± 3.4	0.99 ± 0.32
[M61] CNN 1D, nrs: 0, xpps: 2K, height: 256, raw	83.0 ± 3.0	1.19 ± 0.40	77.7 ± 5.4	0.63 ± 0.12
[M62] CNN 2D, nrs: 0, xpps: 2K, height: 256, raw	84.1 ± 5.1	1.48 ± 0.49	76.9 ± 5.4	0.89 ± 0.35
[M63] CNN 2D, nrs: 6, xpps: 5K, height: 512, HOG 3D	81.8 ± 3.1	2.19 ± 1.56	76.7 ± 4.2	1.01 ± 0.34
[M64] CNN 2D, nrs: 0, xpps: 2K, height: 256, HOG	82.8 ± 4.4	1.56 ± 0.49	76.5 ± 2.8	0.89 ± 0.31
[M65] CNN 2D, nrs: 6, xpps: 5K, height: 512, HOG	82.4 ± 3.3	1.81 ± 0.75	76.5 ± 3.3	0.95 ± 0.27
[M66] CNN 1D, nrs: 6, xpps: 2K, height: 256, raw	80.9 ± 4.5	1.30 ± 0.30	76.2 ± 3.8	0.71 ± 0.19
[M67] CNN 1D, nrs: 6, xpps: 5K, height: 512, raw	83.7 ± 2.9	1.55 ± 0.42	76.0 ± 3.2	0.80 ± 0.17
[M68] CNN 2D, nrs: 0, xpps: 5K, height: 512, HOG	83.5 ± 4.2	1.82 ± 0.40	75.8 ± 4.1	1.05 ± 0.37
[M69] CNN 2D, nrs: 6, xpps: 2K, height: 256, HOG	81.1 ± 4.3	1.28 ± 0.32	75.4 ± 5.3	0.74 ± 0.25
[M70] CNN 2D, nrs: 6, xpps: 2K, height: 256, raw	82.8 ± 4.3	1.32 ± 0.31	75.3 ± 2.3	0.81 ± 0.20
[M71] CNN 1D, nrs: 0, xpps: 2K, height: 256, HOG	82.6 ± 3.2	1.07 ± 0.22	75.1 ± 3.3	0.59 ± 0.09
[M72] CNN 1D, nrs: 6, xpps: 2K, height: 256, HOG	83.0 ± 2.8	1.09 ± 0.17	75.0 ± 7.6	0.68 ± 0.14
[M73] CNN 2D, nrs: 0, xpps: 5K, height: 512, HOG 3D	80.5 ± 4.2	1.44 ± 0.29	74.5 ± 3.8	0.79 ± 0.15
[M74] CNN 2D, nrs: 0, xpps: 2K, height: 256, HOG 3D	79.7 ± 3.8	1.81 ± 1.35	74.1 ± 2.2	0.82 ± 0.20
[M75] LSTM, nrs: 6, xpps: 2K, height: 256, raw	79.4 ± 3.2	3.20 ± 1.26	73.4 ± 3.5	1.58 ± 0.40
[M76] CNN 1D, nrs: 0, xpps: 5K, height: 512, HOG	79.9 ± 4.3	1.37 ± 0.49	73.2 ± 5.6	0.77 ± 0.13
[M77] CNN 1D, nrs: 6, xpps: 5K, height: 512, HOG	79.0 ± 4.5	1.82 ± 0.97	72.6 ± 6.9	0.92 ± 0.17
[M78] LSTM, nrs: 0, xpps: 2K, height: 256, raw	79.7 ± 3.0	3.00 ± 1.40	72.0 ± 3.4	1.40 ± 0.51
[M79] CNN 2D, nrs: 6, xpps: 2K, height: 256, HOG 3D	81.3 ± 4.7	1.52 ± 0.24	71.8 ± 4.2	0.87 ± 0.15
[M80] LSTM, nrs: 6, xpps: 5K, height: 512, raw	69.3 ± 5.0	4.46 ± 0.94	63.5 ± 1.5	2.11 ± 0.53
[M81] LSTM, nrs: 0, xpps: 5K, height: 512, raw	71.8 ± 1.6	4.11 ± 2.11	63.4 ± 3.9	1.73 ± 0.69

Table 5.4: Results of the padded experiment, sorted in descending order by test accuracy.

Due to resource constraints, we could not train the DensNet on the high resolution features, although it achieves the highest test accuracy. Surprisingly, the LSTM model with HOG features outperforms the other CNN models at all resolutions, to read more about this see the corresponding discussion 5.2.5. The CNN 2D models perform slightly worse overall in this experiment, with the best model [M56] having an accuracy of approximately 6% less in comparison to the accuracy of the best CNN 2D model [M2] in the compressed experiment.

Looking at the confusion matrices in Figure 5.13, the same difficulties are apparent as in the previous experiments, the main difficulty is the separation of the B2, B3 and B4 syllables. As already evident in the variable length experiment, the VS and VSV syllables are confused with each other to a small extent.

Looking at the LRP results on the Figure 5.14 yielded by the CNN 2D model [M56], one can see that the model has unfortunately learned to use some features from the background to classify the syllables. This can be seen in the LRP heatmap "VS" where there is a red stroke in the bottom centre of the background part of

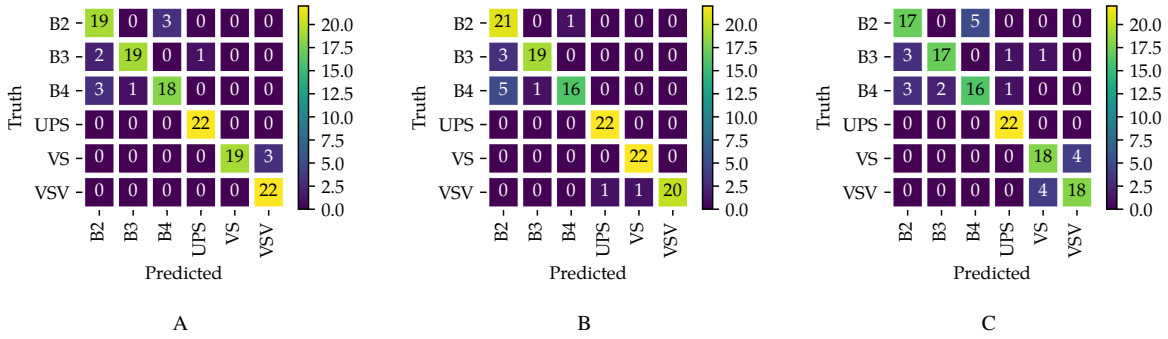

Figure 5.13: Confusion matrices of some models of the padded test experiment performing on a test set: (A) DenseNet model [M50], (B) LSTM model [M51], (C) CNN 2D model [M56].

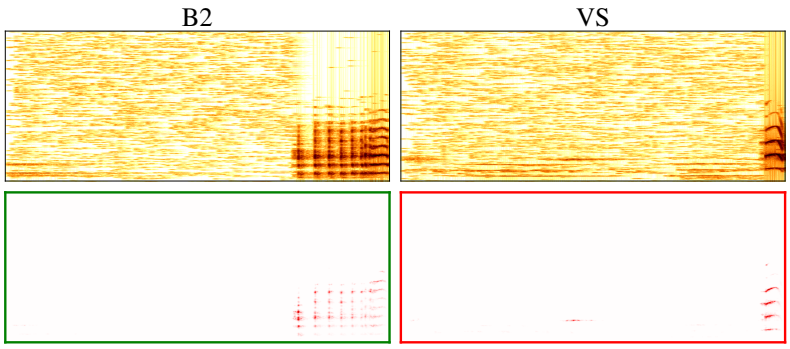

Figure 5.14: LRP of the CNN 2D model [M56] on raw features (red = wrong prediction).

the sample. On the other hand, the B2 syllable heatmap shows us, that the model has the ability to learn weights which target the syllable only.

5.2.5 Discussion

We show with these three test experiments that compression of the features leads to a drop in the performance and by providing simply the untouched syllables, resulting in variable length, the combination with HOG features and a LSTM network yields promising results. This is highlighted by the results of the variable length dataset with the compressed dataset in the LSTM model, we see a huge drop of around 12.5% in test accuracy between the best models [M37] and [M16]. This is supported by the comparison with the LSTM models of the padded experiment, here the LSTM models in the variable length experiment perform better with an around 1.6% higher test accuracy with quite similar variance compared to the best LSTM models from the padded experiment. For the DenseNet model we see similar behaviour, but with a less substantial difference that ultimately does not allow a clear conclusion.
The results of the CNN 2D model are opposite to the previous ones, it failed in the left padded experiment and achieved significantly better results in the compressed test experiment. However, we do not believe these results should be given much weight as we probably added or created patterns in the left padded experiment by padding with the silent audio parts, which leads the CNN networks to make false conclusions. That the model has found patterns in the padded area is visible in the LRP results, see Figure 5.14.
The padded experiment was initially right padded, there the LSTM model failed at at all levels and is outperformed by every CNN model feature combination. So the change to padding the samples on the left side in the preprocessing process leads to surprising results from the LSTM models mentioned in the padded experiment section. This could be explained by how the LSTM encounters the features from left to right,

Model	Validate Accuracy (%)	Loss	Test Accuracy (%)	Loss
[M37] LSTM, variable length, HOG	93.4 ± 2.1	0.66 ± 0.17	88.7 ± 2.4	0.44 ± 0.17
[M38] LSTM, variable length, HOG	93.0 ± 2.9	0.59 ± 0.21	88.5 ± 1.8	0.40 ± 0.11
[M50] DenseNet, left padded, raw	93.4 ± 4.4	1.68 ± 2.82	88.4 ± 3.5	0.62 ± 0.31
[M39] LSTM, variable length, HOG	91.9 ± 1.4	0.70 ± 0.25	88.2 ± 1.7	0.42 ± 0.11
[M40] LSTM, variable length, HOG	93.0 ± 2.7	0.51 ± 0.20	88.1 ± 3.4	0.42 ± 0.13
[M1] DenseNet, compressed, raw	93.4 ± 4.2	1.23 ± 0.75	87.5 ± 2.8	0.65 ± 0.29
[M52] DenseNet, left padded, raw	91.5 ± 4.9	1.09 ± 0.84	87.1 ± 4.4	0.81 ± 0.43
[M51] LSTM, left padded, HOG	91.9 ± 1.6	0.76 ± 0.13	87.1 ± 2.1	0.45 ± 0.11
[M2] CNN 2D, compressed, raw	92.2 ± 5.1	0.97 ± 0.60	87.0 ± 4.4	0.43 ± 0.18
[M41] LSTM, variable length, HOG	93.4 ± 1.8	0.63 ± 0.31	87.0 ± 2.4	0.46 ± 0.12
[M3] CNN 2D, compressed, raw	91.6 ± 5.2	0.80 ± 0.42	85.6 ± 6.0	0.41 ± 0.14
[M53] DenseNet, left padded, HOG	89.8 ± 4.6	1.50 ± 0.91	85.0 ± 4.8	0.88 ± 0.41
[M4] CNN 2D, compressed, raw	91.2 ± 5.5	0.87 ± 0.43	84.5 ± 5.5	0.38 ± 0.14
[M54] LSTM, left padded, HOG	89.6 ± 3.7	0.87 ± 0.22	84.5 ± 4.6	0.59 ± 0.19
[M5] DenseNet, compressed, raw	90.9 ± 5.2	1.51 ± 1.26	84.4 ± 6.5	0.72 ± 0.42

Table 5.5: Overview of the top five models in each test experiment, sorted in descending order by test accuracy.

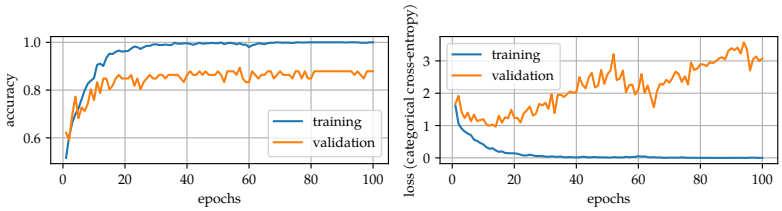

Figure 5.15: Training progression for CNN 2D model [M56] on raw features.

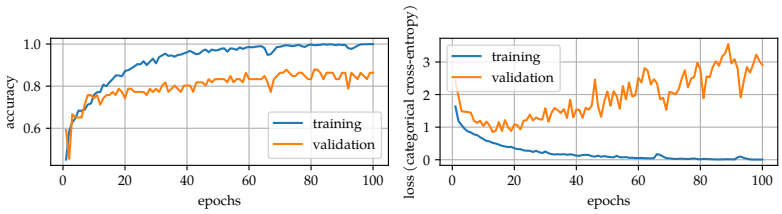

Figure 5.16: Training progression for CNN 1D model [M58] on raw features.

so the information about the syllable was vanished by the padding information. With the CNN models, no great difference is to be expected, yet the accuracy of the DenseNet increased by 1% and the simpler models performed about 1% worse. An explanation is given by the training process in Figure 5.15 and Figure 5.16, where it is obvious that the simpler CNN models have difficulties in learning and eventually always overfit, e.g. they are stuck in a local optima as already explained. For these models, it might be necessary to adapt the training process or add stronger regularization abilities.
Throughout all three test experiments, the models had difficulty in differentiating between the syllables B2, B3 and B4. To a lesser extent, they struggled with VS and VSV. Part of this can be explained by their similarities, which is to some degree visible and has been studied by comparing audio features (personal communication AA. Fernandez). As example, VS and VSV belong to the set named as "tonal syllables"

which included only a single frequency modulation.
Following previous work, we trained the CNN 2D models on the HOG features with the gradients comprising the third dimension, the gradients thus correspond to the channels of the tensor and therefore each gradient is analysed by a dedicated kernel. Overall, this approach showed poor results and does not seem worth mentioning. Nevertheless, we think this is an interesting idea and we find it important to also mention which ideas did not lead to good results, to help future researchers.
Ultimately, it is not clear over all models exactly which preprocessing leads to better results, but some trends are discernible, e.g. that noise reduction tends to worsen the performance of the models and higher resolution leads to better performance. Interestingly, in the latter case, the variance of all metrics is also often considerably lower. In the case of the features, it is clear that the HOG features in combination with the LSTM networks and the raw features with the CNN models are more advantageous than other combinations.

5.3 Seq experiment

As the main goal of this thesis was to evaluate an automated way to classify the syllable types from bat pups, we evaluate the performance of the DL models on windowed samples of the audios. So the goal for this experiment is to find the best parameters for the split algorithm 4.3.2 and then the best DL model, with which we then implement a naive automated syllable detection tool.

5.3.1 Setup

We applied the split algorithm 4.3.2 with different window length, strides and minimum covered area. In the final experiment we compared all combinations using the following settings:

— length of $30\,\mathrm{ms}$

— stride of $10\,\mathrm{ms}$ or $5\,\mathrm{ms}$

— min cover length (mcl) of 60% or 80% (minimum width of a slice that has to be covered by a syllable if the boundaries of the syllable are not within the slice.)

And all the spectrogram images are created with a time resolution of 4000 xpps and height of 512 pixels.

5.3.2 Results

Every model shows a more or less poor performance on the HOG features, this could be explained by the again reduced information provided by each slice when converted to HOG features, as the slices are already much smaller than the ones used in previous experiments. Therefore, we excluded the HOG features in the results. All results are in the appendix in Table A.1.
Gazing at the distribution of samples in the Figure 5.17, the splitting algorithm produces rather unbalanced datasets where the background samples are heavily overrepresented. The resulting datasets with an mcl of 80% exibit the highest ratio r between the number of training samples of the least represented syllable and background samples. The dataset where the DL models yield the best results ($5\,\mathrm{ms}$ stride and 60% mcl), possesses the smallest ratio r around 1:41. The number of samples generated increases almost twofold between the two stride values $10\,\mathrm{ms}$ to $5\,\mathrm{ms}$. Compared to the sampling distribution of the original dataset (see Figure 3.2), the generated datasets contain at least more than twice as many samples per classes.
Compared to all previous runs of this experiment, we get by far the best test accuracy of 93.65% for the [M82] model. Interestingly, the LSTM model [M84] performs quite well on the raw features and ranks right behind the two best results of the DenseNet. Even more surprisingly, the 1D CNN model outperforms the 2D CNN model the first time. Similarly to the LSTM model, the 1D CNN model is designated to find patterns in sequential inputs, as the kernel considers information in the width axis, which corresponds to the time axis of the spectrogram images.
The best results are obtained with a step size of $5\,\mathrm{ms}$, where at minimum 60% of the slice width is covered by one syllable or the syllable is entirely inside it. In this experiment, all models tend to overfit, see figures

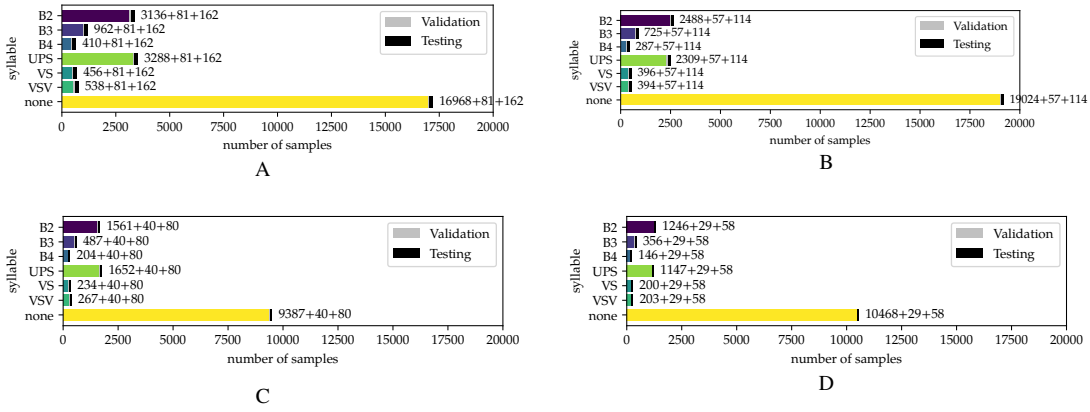

Figure 5.17: The generated distributions of the syllable and background samples of the simple call dataset. The datasets are not balanced with the ratio r expressing the number of training samples ratio between the least represented syllable and background samples. A) stride is $5\,\mathrm{ms}$, mcl is 60% and r is around $1:41$. B) stride is $5\,\mathrm{ms}$, mcl is 80% and r is around $1:66$. C) stride is $10\,\mathrm{ms}$, mcl is 60% and r is around $1:46$. D) stride is $10\,\mathrm{ms}$, mcl is 80% and r is around $1:72$.

Model	Validate Accuracy (%)	Loss	Test Accuracy (%)	Loss
[M82] DenseNet, stride: 0.005, mcl: 0.6, raw	95.9 ± 0.7	0.84 ± 0.23	93.7 ± 0.6	0.25 ± 0.04
[M83] DenseNet, stride: 0.005, mcl: 0.8, raw	95.1 ± 0.5	1.33 ± 0.48	92.3 ± 1.4	0.28 ± 0.09
[M84] LSTM, stride: 0.005, mcl: 0.6, raw	93.2 ± 1.0	1.75 ± 0.32	89.7 ± 1.1	0.43 ± 0.08
[M85] CNN 1D, stride: 0.005, mcl: 0.6, raw	91.4 ± 0.8	2.05 ± 0.38	89.3 ± 0.9	0.42 ± 0.06
[M86] CNN 1D, stride: 0.005, mcl: 0.8, raw	90.7 ± 1.5	2.45 ± 0.56	88.6 ± 1.1	0.39 ± 0.05
[M87] LSTM, stride: 0.005, mcl: 0.8, raw	92.2 ± 1.5	3.12 ± 0.59	88.4 ± 0.7	0.52 ± 0.04
[M88] DenseNet, stride: 0.01, mcl: 0.8, raw	90.8 ± 1.4	2.92 ± 0.65	87.9 ± 1.5	0.45 ± 0.08
[M89] DenseNet, stride: 0.01, mcl: 0.6, raw	90.8 ± 1.0	3.40 ± 1.31	87.3 ± 1.4	0.63 ± 0.17
[M91] CNN 2D, stride: 0.005, mcl: 0.6, raw	87.7 ± 0.5	2.87 ± 1.13	86.3 ± 1.5	0.55 ± 0.20
[M93] CNN 2D, stride: 0.005, mcl: 0.8, raw	88.3 ± 1.3	3.99 ± 0.91	85.8 ± 1.4	0.59 ± 0.14
[M96] CNN 2D, stride: 0.01, mcl: 0.8, raw	82.1 ± 1.4	4.33 ± 0.37	81.1 ± 2.0	0.57 ± 0.03
[M97] CNN 2D, stride: 0.01, mcl: 0.6, raw	82.3 ± 1.2	4.29 ± 2.09	80.1 ± 1.8	0.71 ± 0.18
[M101] CNN 1D, stride: 0.01, mcl: 0.6, raw	83.3 ± 2.3	4.59 ± 2.20	78.8 ± 2.6	0.89 ± 0.32
[M103] LSTM, stride: 0.01, mcl: 0.6, raw	81.9 ± 0.9	5.65 ± 1.12	78.2 ± 1.9	0.98 ± 0.15
[M106] CNN 1D, stride: 0.01, mcl: 0.8, raw	81.5 ± 1.6	5.62 ± 1.33	77.5 ± 2.4	0.81 ± 0.13
[M111] LSTM, stride: 0.01, mcl: 0.8, raw	80.0 ± 2.7	8.41 ± 2.13	75.5 ± 1.1	1.00 ± 0.17

Table 5.6: Results of the final sequence experiment without the HOG features results, sorted in descending order by test accuracy.

Figure 5.19, Figure 5.20 and Figure 5.21. Like the DenseNet models, the other two struggle to distinguish background samples from syllable samples. In general, the confusion matrix shows similar problems in all three models, differing only in degree. Besides the background problem, there are the difficulties in distinguishing B2, B3 and B4 syllables and the distinction between VS and VSV syllables, which was already evident in the previous test experiments.
Finally, looking at the LRP results Figure 5.22 of the best DenseNet model [M82], one can see that the model has learned many of the background features and somehow failed to represent features at the syllable level. However, with this result, it should be noted that the Deeptayler algorithm has been modified 4.6.3, so the

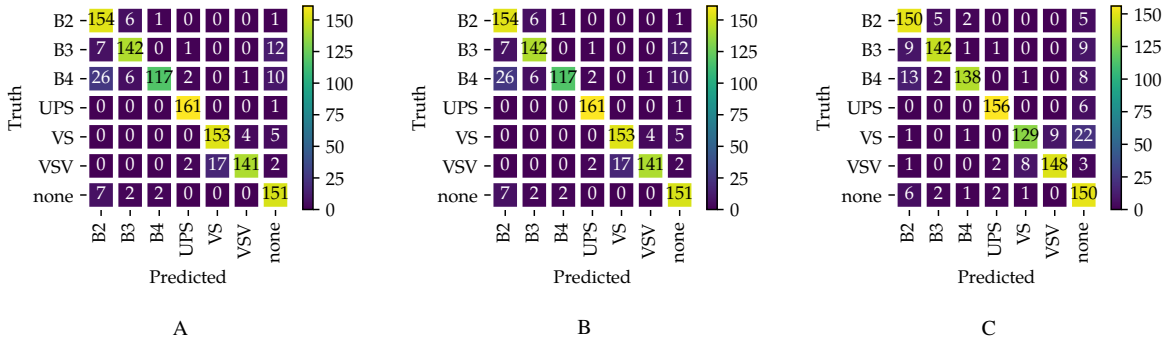

Figure 5.18: Confusion matrices of the following models performing on the test set: (A) DenseNet model [M82], (B) LSTM model [M84], (C) CNN 1D model [M85].

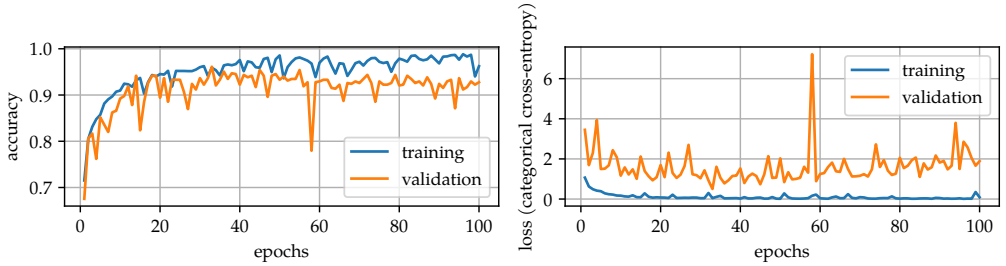

Figure 5.19: Training progression for DenseNet model [M82] on raw features.

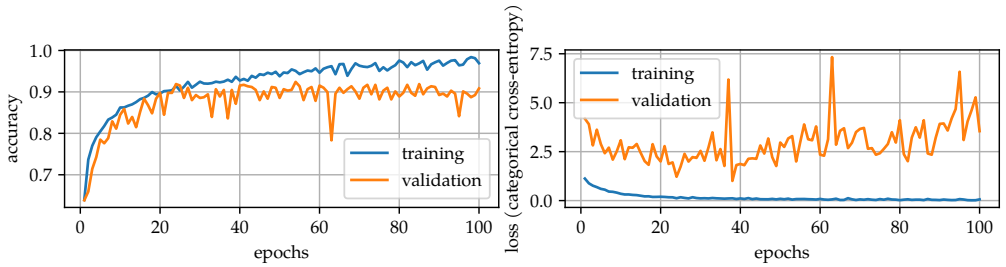

Figure 5.20: Training progression for LSTM model [M84] on raw features.

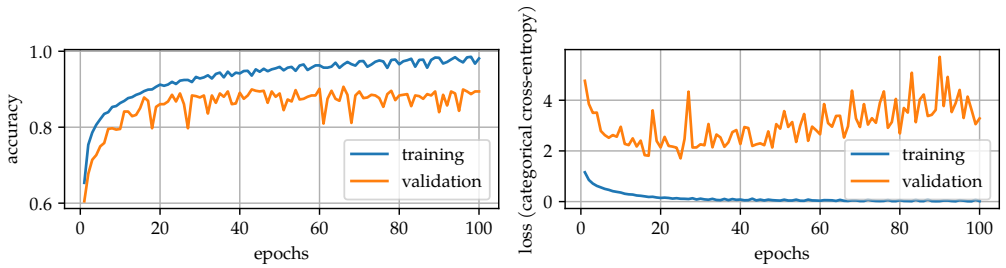

Figure 5.21: Training progression for CNN 1D model [M85] on raw features.

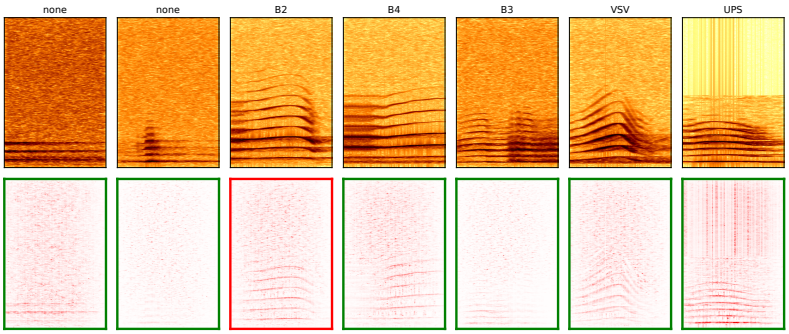

Figure 5.22: LRP of the DenseNet model [M82] on raw features (red = wrong prediction).

results should be taken with a degree of caution.

5.3.3 Discussion

The small slices with the best results are somehow surprisingly, as the size of a slice captures only a small part of the wider syllable. On the other hand it is more likely that only one syllable is present in one slice, as the shortest syllables have a duration around $5\,\mathrm{ms}$ and therefore the model learns to distinguish syllable types by the structure of syllable and not the noise. Additionally, the overlap plays a crucial role and the model shows a clear trend to smaller strides. A part of the effect can be explained by the principle of data generation by augmentation, using shorter length and steps we generate more samples with the syllable occurring more often at different positions on the slices. In addition, the syllable covers more of a slice labelled with the corresponding syllable. Resulting in an increasing amount of data with less noise in it which leads the DL model to focus more easily on the relevant structures.
Although we do not take the entire audio file into account when generating the training set by ignoring areas that are not close to a label, the number of background samples is ultimately significantly greater than the number of samples per syllable class. This could be problematic, but considering that the background samples might contain most of the variance in the data, overrepresentation from the background samples is quite desirable.
However, the high accuracy values should be taken with a grain of salt as this experiment has significantly more samples and therefore some classes can have a quite high misclassification rate for the accuracy achieved. Looking at the confusion matrix of the DenseNet model [M82] in the Figure 5.18, one can see that it has difficulty distinguishing background samples labeled as "none" from the others. This could be because the audio sources are not exhaustively labelled and thus syllables were automatically labelled as background by the split algorithm.

6 Conclusions and Outlook

We summarise what we have found and indicate where the journey should go by outlining the essence of this work in the next sections.

6.1 Conclusions

We investigated how different deep learning model architectures were able to classify the different preprocessed features, i.e. their ability to create a mapping between features and their respective syllables. Besides the promising DL model DenseNet, we adopted the other DL models from Gilles Waeber's work [Wae19], namely the LSTM, as well as the CNN 1D and the CNN 2D. We investigated some parameters of the feature processing, such as the sensitivity of the noise reduction filter, as well as the resolution of the spectrograms. Based on the results, we were able to show that ML is suitable to classify the provided syllables with an acceptable accuracy given the DL models applied. Furthermore, we elicited which preprocessings improve or worsen the DL model performance, which is partially model type specific. The experiments clearly showed that the LSTM performs better on HOG features and the CNN networks on raw features. The results in terms of spectrogram resolution were less clear. In a few cases it is obvious that higher resolution in frequency and time leads to better results, e.g. in the padding experiment high resolution spectrogram images lead to higher accuracy for the CNN 2D model. This is in line with expectations as we study high frequencies which are better reproduced in high resolution spectrograms. The effect of noise reduction is rather confusing. From the visual side, one sees that the syllable structures on the spectrogram are distorted when noise filtering is applied, but the DL models sometimes perform better and sometimes worse with noise filtered features. Overall, there is evidence that no noise reduction gives the best DL model performance and higher noise reduction worsens the results.
Besides the few parameters examined, we stuck to the features and corresponding parameters that are already implemented by the applied framework and therefore arbitrarily chosen, e.g. the selection of the slit-style HOG is never compared with a basic HOG transformation, nor are different configurations examined. This also applies to the window type of the Fourier transform. Future work should therefore take the values of these parameters with a grain of salt and would be well advised to take these values from other works that has investigated the parameter in question or to investigate them in their own experiments.
Although many works on vocalisation extract high-level features, which definitely shortens the input size, we did not consider this. Our intention was to train DL models on low-level features, which they can use to learn any high-level feature extraction. Therefore, we cannot make any statements about how learning DL models on usual high-level features compares to the variant we studied. However, this would be interesting

to examine further, as a major problem with our DL models is overfitting, which usually starts in the early epochs. This could be due to the small number of samples as opposed to the large number of features, especially for some syllable types. Therefore, instead of generating more labelled data, one could also investigate the training of the DL models using high-level features.
Concerning the difficulties with the gradual change of a syllable type during development, we were able to show with these experiments that the networks coped well. However, it should be taken into consideration that humans could only mark syllables that were recognisable to human perception and thus had already matured somewhat. Since we are currently still trying to achieve a similar accuracy to a human expert with the DL models, there is still a lot of space for future work.
During writing we stumbled over an interesting discussion about k-fold CV. One of the criticisms is the inappropriate recommendation of performing experiments with the aim of later setting $k = 10$ or $k = 5$ for model validation using k-fold CV, as this seems to lead to an inaccurate prediction error estimation [ZY15]. Moreover, the initialisation of the weights is not set to a specific seed, as are the calculations of Tensorflow and CuDNN, which contributes to a further increase in variability. According to the arguments of Zhang et al. [ZY15] and how the models are initialized the performance of the models in our results is not fully accurate and thus cannot be used for a robust comparison. How accurate the results are and what the additional variability due to omitting seed initialisation contributes is not known to the author at this time and requires further knowledge acquisition about the "variance bias tradeoff" combined with data and model specifications.
The results from the seq experiment describing the pipeline of the automated syllable type classifier prototype are promising at the first glance, but by examining the confusion matrices it becomes clear that more work is needed to distinguish between the syllable types and the background samples.
In a final experiment, we implemented a simple process for automatic syllable type classification that provides a tabular result, describing during which timespan which syllable type was recognised. This process is based on the structure of the sequence experiments and can therefore be used with all models trained in these experiments. We did some tests with the best model from the sequence experiments [M82], but the results were disappointing and highlighted the problem of syllables being confused with background and vice versa. This is shown by the fact that the DL model labels the background as a syllable or recognises a timespan as a background in the middle of a syllable. Since this problematic issue was already visible in the diffusion matrix of the test set results of the DL model in question, it is aggravated on data belonging to different recordings, showing poor generalisation and overfitting of the DL model.
Overall, we have gathered a good amount of knowledge about the classification of syllable types from vocalisations of bat pups and provide a framework for future work on this topic. How the work might continue is explained in the next section.

6.2 Outlook

We gained relatively good results from the variable length test experiment, so one could consider combining the LSTM network with the same pipeline with a syllable segmentation algorithm. There exists a lot of research on syllable segmentation with promising results for higher quality audios, i.e. with as little noise in them as possible. For high quality audios, a combination of automatic syllable segmentation and later classification with the LSTM network could give satisfying results given the current requirements. If the audio quality would drop, one could further apply the raw audio bitstream to an ANN, where the network could decide how to transform the data at a deeper level, learning a more accurate transformation for syllable classification. As this field is relatively young, current experiments need a lot of time to evaluate the concepts as there is not much work to rely on. Interesting work is addressing how the raw waveform might be represented, Okawa et al. propose a bit representation of the waveform where they convert the integer values into a list of bit vectors [Oka+19].
In the sequence experiment, we encountered the problem that a good amount of syllables were classified as background, which could be due to the not exhaustively labelled data. Besides the additional manual labeling of the data one can use the developed silence detection algorithm to filter out non silence unlabeled parts of the audio.

Based on better quality data, one could explore the sequence-to-sequence approach in combination with an ANN that analyses the bitstream directly. In the sequence-to-sequence approach, an encoder analyses the input, then a decoder uses the inner states of the encoder in combination with the previous sequence (this can also be just a start symbol) to generate a new symbol that is appended to the previous sequence. This is repeated until the newly generated symbol is the end symbol signalling the end of the "translation". This would also be a variant that allows the beginning and end of syllables in a sequence to be recognised.
For assessing the overfitting problem, one possible approach would be to generate more training samples or apply a semi supervised learning process called co-training as we have additional unlabeled data. As we already use different models this concept could be adapted with ease. An additional approach could be to study the data in a more advanced way and build the different subsets with knowledge about how the syllable structure evolves over time. This would result in the model seeing the syllables in the different developmental forms, which would ultimately improve the generalisation of the model. And as already applied, one can try different configurations on the dropout rate and affected layers.
A different direction would be to learn more about the models in human speech recognition and perhaps try to adapt the findings to bat data. Unfortunately, this direction involves information about speech structure that is lacking at current time, hence requiring more knowledge about bat song syntax and information contained in their acoustics.
Another approach would be to study the latent feature space of the bat vocalisations. This could be conducted with several generative neural networks applying uniform manifold approximation and projection (UMAP) to the songs for searching a lower dimensional space describing the inner structure as proposed by Sainburg et al. [STG20].
In addition to the considerations of increasing classification performance, functional and usable software needs intuitive usability and readable results both for humans and for software applications that further process the results. Unlike all classification considerations, this can be achieved by defining appropriate software requirements in close cooperation with clients. Since this kind of software is developed without much consideration for resources, portability or anything else, there is still a lot of room for improvement. An example could be at which stage the spectrograms are created, as when this is applied before the splitting step, one could consume much less storage space.
Recently, research into understanding human intelligence in engineering terms has focused on how humans discover and learn from the world in their early years - concepts about learning processes that can be applied to machine learning. In addition, the ongoing rapid development of DL is enabling more problems in bioacoustics to be addressed. Some of these problems are related to cognitive science, especially speech development. Thus, a promising future is emerging for gaining insights into our speech development at an accelerated pace, which could then potentially be adapted for AI processes.

A
AP1

A.1 Image sources

Following figures are modified variants of some figures in the TiKZ GitHub repository by Petar Veličković: Figure 4.9, Figure 4.8 and Figure 4.6. The DL model figures Figure 5.2 and Figure 5.1 are drawn with the code fragments from PlotNeuralNet GitHub repository by Haris Iqbal.

A.2 Supplementary Tables

Model	Validate Accuracy (%)	Loss	Test Accuracy (%)	Loss
[M82] DenseNet, stride: 0.005, mcl: 0.6, raw	95.9 ± 0.7	0.84 ± 0.23	93.7 ± 0.6	0.25 ± 0.04
[M83] DenseNet, stride: 0.005, mcl: 0.8, raw	95.1 ± 0.5	1.33 ± 0.48	92.3 ± 1.4	0.28 ± 0.09
[M84] LSTM, stride: 0.005, mcl: 0.6, raw	93.2 ± 1.0	1.75 ± 0.32	89.7 ± 1.1	0.43 ± 0.08
[M85] CNN 1D, stride: 0.005, mcl: 0.6, raw	91.4 ± 0.8	2.05 ± 0.38	89.3 ± 0.9	0.42 ± 0.06
[M86] CNN 1D, stride: 0.005, mcl: 0.8, raw	90.7 ± 1.5	2.45 ± 0.56	88.6 ± 1.1	0.39 ± 0.05
[M87] LSTM, stride: 0.005, mcl: 0.8, raw	92.2 ± 1.5	3.12 ± 0.59	88.4 ± 0.7	0.52 ± 0.04
[M88] DenseNet, stride: 0.01, mcl: 0.8, raw	90.8 ± 1.4	2.92 ± 0.65	87.9 ± 1.5	0.45 ± 0.08
[M89] DenseNet, stride: 0.01, mcl: 0.6, raw	90.8 ± 1.0	3.40 ± 1.31	87.3 ± 1.4	0.63 ± 0.17
[M90] DenseNet, stride: 0.005, mcl: 0.6, HOG	89.1 ± 1.1	3.28 ± 1.14	86.8 ± 1.1	0.71 ± 0.15
[M91] CNN 2D, stride: 0.005, mcl: 0.6, raw	87.7 ± 0.5	2.87 ± 1.13	86.3 ± 1.5	0.55 ± 0.20
[M92] DenseNet, stride: 0.005, mcl: 0.8, HOG	90.3 ± 1.4	3.43 ± 0.97	86.2 ± 0.7	0.68 ± 0.15
[M93] CNN 2D, stride: 0.005, mcl: 0.8, raw	88.3 ± 1.3	3.99 ± 0.91	85.8 ± 1.4	0.59 ± 0.14
[M94] LSTM, stride: 0.005, mcl: 0.6, HOG	86.0 ± 0.5	3.83 ± 0.73	83.1 ± 1.0	0.82 ± 0.14
[M95] LSTM, stride: 0.005, mcl: 0.8, HOG	87.0 ± 1.1	3.82 ± 1.29	83.1 ± 1.5	0.66 ± 0.17
[M96] CNN 2D, stride: 0.01, mcl: 0.8, raw	82.1 ± 1.4	4.33 ± 0.37	81.1 ± 2.0	0.57 ± 0.03
[M97] CNN 2D, stride: 0.01, mcl: 0.6, raw	82.3 ± 1.2	4.29 ± 2.09	80.1 ± 1.8	0.71 ± 0.18
[M98] CNN 2D, stride: 0.005, mcl: 0.8, HOG	81.5 ± 1.6	3.87 ± 1.15	79.6 ± 1.3	0.58 ± 0.13
[M99] LSTM, stride: 0.01, mcl: 0.8, HOG	83.1 ± 1.8	4.30 ± 1.66	79.5 ± 2.4	0.65 ± 0.17
[M100] CNN 1D, stride: 0.005, mcl: 0.8, HOG	80.8 ± 1.6	3.58 ± 0.67	79.4 ± 1.2	0.57 ± 0.06
[M101] CNN 1D, stride: 0.01, mcl: 0.6, raw	83.3 ± 2.3	4.59 ± 2.20	78.8 ± 2.6	0.89 ± 0.32
[M102] LSTM, stride: 0.01, mcl: 0.6, HOG	82.1 ± 1.7	4.76 ± 2.09	78.5 ± 1.8	0.88 ± 0.30
[M103] LSTM, stride: 0.01, mcl: 0.6, raw	81.9 ± 0.9	5.65 ± 1.12	78.2 ± 1.9	0.98 ± 0.15
[M104] CNN 2D, stride: 0.005, mcl: 0.8, HOG 3D	79.5 ± 1.5	3.90 ± 0.99	78.0 ± 1.9	0.58 ± 0.09
[M105] CNN 2D, stride: 0.005, mcl: 0.6, HOG	79.0 ± 1.3	3.59 ± 1.68	77.6 ± 0.8	0.68 ± 0.18
[M106] CNN 1D, stride: 0.01, mcl: 0.8, raw	81.5 ± 1.6	5.62 ± 1.33	77.5 ± 2.4	0.81 ± 0.13
[M107] DenseNet, stride: 0.01, mcl: 0.8, HOG	81.3 ± 1.6	6.32 ± 2.26	77.4 ± 1.4	0.97 ± 0.23
[M108] CNN 1D, stride: 0.005, mcl: 0.6, HOG	78.6 ± 1.6	3.26 ± 0.86	77.2 ± 1.4	0.69 ± 0.16
[M109] DenseNet, stride: 0.01, mcl: 0.6, HOG	80.6 ± 1.7	5.79 ± 1.84	76.5 ± 1.8	1.07 ± 0.33
[M110] CNN 2D, stride: 0.01, mcl: 0.8, HOG	77.8 ± 2.5	4.31 ± 0.64	76.2 ± 1.9	0.64 ± 0.04
[M111] LSTM, stride: 0.01, mcl: 0.8, raw	80.0 ± 2.7	8.41 ± 2.13	75.5 ± 1.1	1.00 ± 0.17
[M112] CNN 2D, stride: 0.005, mcl: 0.6, HOG 3D	76.8 ± 1.7	3.49 ± 0.79	75.2 ± 1.0	0.70 ± 0.12
[M113] CNN 1D, stride: 0.01, mcl: 0.8, HOG	77.6 ± 2.0	4.44 ± 0.83	74.8 ± 2.0	0.65 ± 0.07
[M114] CNN 1D, stride: 0.01, mcl: 0.6, HOG	77.0 ± 1.1	3.73 ± 0.96	74.3 ± 1.4	0.70 ± 0.09
[M115] CNN 2D, stride: 0.01, mcl: 0.6, HOG	76.6 ± 1.9	4.42 ± 1.39	73.9 ± 1.6	0.81 ± 0.23
[M116] CNN 2D, stride: 0.01, mcl: 0.8, HOG 3D	75.4 ± 1.8	5.27 ± 1.18	73.4 ± 1.3	0.72 ± 0.08
[M117] CNN 2D, stride: 0.01, mcl: 0.6, HOG 3D	74.8 ± 1.4	4.02 ± 0.58	72.2 ± 0.7	0.71 ± 0.05

Table A.1: Results of the final sequence experiment, sorted in descending order by test accuracy.

A.3 Supplementary Images

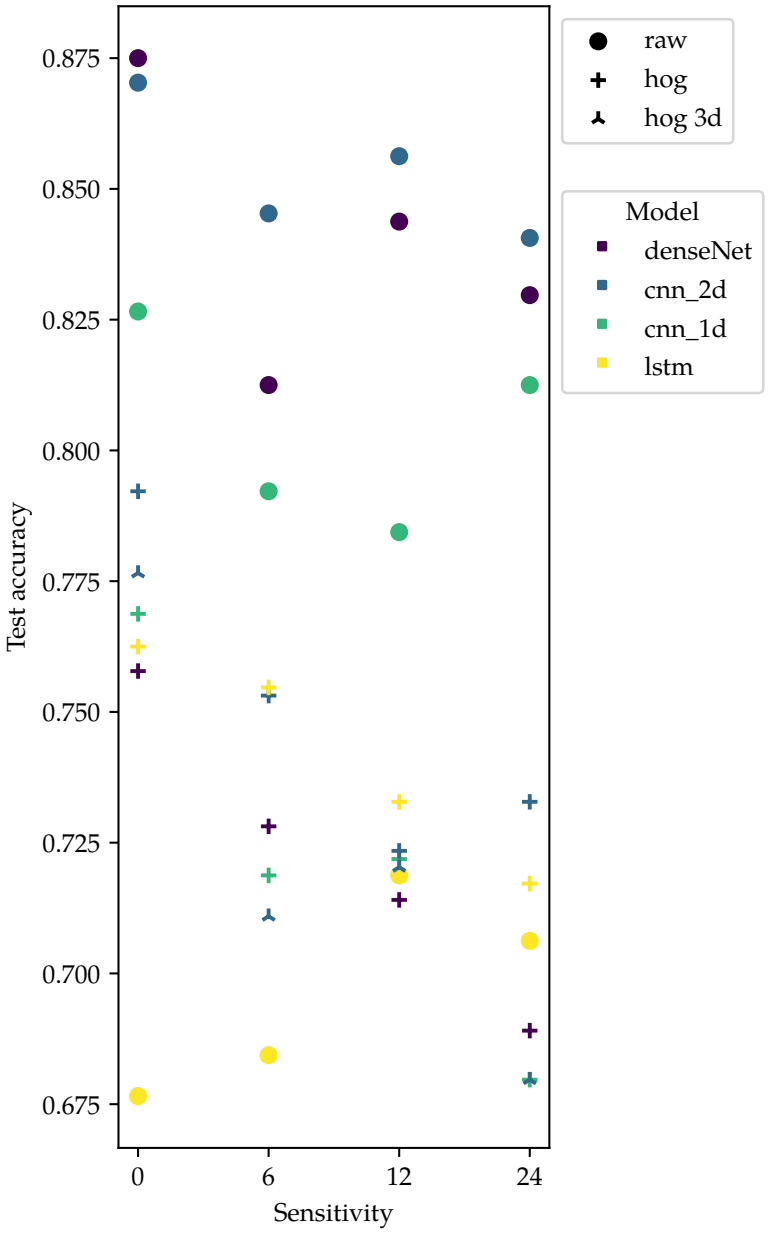

Figure A.1: Scatter plot of the distribution of the model validation accuracy of the compressed test experiment, differentiated by sensitivity and feature-type.

Models

[M1] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 49.0 ± 20.5 (27, 29, 30, 42, 54, 56, 66, 85)
Results: validation: $(93.4 \pm 4.2)\,\%$, testing: $(87.5 \pm 2.8)\,\%$.

[M2] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 41.0 ± 23.9 (15, 21, 25, 26, 40, 55, 65, 82)
Results: validation: $(92.2 \pm 5.1)\,\%$, testing: $(87.0 \pm 4.4)\,\%$.

[M3] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 35.0 ± 11.1 (25, 27, 28, 28, 34, 34, 49, 55)
Results: validation: $(91.6 \pm 5.2)\,\%$, testing: $(85.6 \pm 6.0)\,\%$.

[M4] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 37.0 ± 19.4 (12, 16, 26, 35, 37, 47, 51, 71)
Results: validation: $(91.2 \pm 5.5)\,\%$, testing: $(84.5 \pm 5.5)\,\%$.

[M5] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 43.0 ± 13.4 (26, 27, 33, 43, 44, 50, 54, 64)
Results: validation: $(90.9 \pm 5.2)\,\%$, testing: $(84.4 \pm 6.5)\,\%$.

[M6] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 39.0 ± 22.3 (21, 23, 25, 26, 31, 46, 50, 87)
Results: validation: $(90.6 \pm 8.0)\,\%$, testing: $(84.1 \pm 5.7)\,\%$.

[M7] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 56.0 ± 17.6 (33, 41, 46, 48, 59, 63, 75, 85)
Results: validation: $(90.6 \pm 5.0)\,\%$, testing: $(83.0 \pm 4.5)\,\%$.

[M8] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 37.0 ± 10.1 (21, 31, 31, 34, 37, 40, 46, 54)
Results: validation: $(90.6 \pm 4.4)\,\%$, testing: $(82.7 \pm 4.7)\,\%$.

[M9] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 60.0 ± 21.8 (31, 31, 51, 53, 69, 74, 78, 90)

Results: validation: $(91.9 \pm 5.5)\,\%$, testing: $(81.2 \pm 5.6)\,\%$.

[M10] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 48.0 ± 18.2 (27, 31, 39, 44, 48, 48, 62, 84)
Results: validation: $(87.5 \pm 5.7)\,\%$, testing: $(81.2 \pm 6.1)\,\%$.

[M11] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 42.0 ± 29.0 (8, 17, 24, 27, 34, 63, 71, 89)
Results: validation: $(87.5 \pm 4.0)\,\%$, testing: $(79.2 \pm 2.7)\,\%$.

[M12] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 31.0 ± 18.3 (20, 20, 21, 23, 24, 32, 37, 74)
Results: validation: $(86.6 \pm 5.5)\,\%$, testing: $(79.2 \pm 4.6)\,\%$.

[M13] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 43.0 ± 26.7 (10, 21, 22, 29, 46, 60, 73, 83)
Results: validation: $(86.2 \pm 5.2)\,\%$, testing: $(78.4 \pm 4.3)\,\%$.

[M14] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG 3D
Parameters: epochs: 15.0 ± 7.3 (9, 10, 12, 12, 14, 15, 16, 32)
Results: validation: $(83.8 \pm 3.3)\,\%$, testing: $(77.7 \pm 3.7)\,\%$.

[M15] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 43.0 ± 26.8 (9, 14, 35, 37, 42, 45, 73, 88)
Results: validation: $(85.6 \pm 3.5)\,\%$, testing: $(76.9 \pm 5.1)\,\%$.

[M16] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 48.0 ± 24.2 (6, 25, 41, 47, 54, 55, 72, 81)
Results: validation: $(89.7 \pm 4.9)\,\%$, testing: $(76.2 \pm 2.6)\,\%$.

[M17] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 46.0 ± 22.6 (29, 31, 36, 39, 45, 45, 46, 100)
Results: validation: $(87.8 \pm 2.5)\,\%$, testing: $(75.8 \pm 4.4)\,\%$.

[M18] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 37.0 ± 13.1 (19, 22, 32, 34, 38, 47, 52, 54)
Results: validation: $(84.1 \pm 7.3)\,\%$, testing: $(75.5 \pm 4.9)\,\%$.

[M19] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 43.0 ± 25.7 (10, 17, 35, 37, 41, 43, 73, 86)
Results: validation: $(82.8 \pm 2.1)\,\%$, testing: $(75.3 \pm 6.0)\,\%$.

[M20] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 53.0 ± 10.9 (32, 47, 49, 53, 55, 57, 65, 66)
Results: validation: $(85.3 \pm 5.7)\,\%$, testing: $(73.3 \pm 6.0)\,\%$.

[M21] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 41.0 ± 14.8 (16, 26, 39, 39, 43, 53, 55, 59)
Results: validation: $(80.9 \pm 6.1)\,\%$, testing: $(73.3 \pm 9.7)\,\%$.

[M22] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 54.0 ± 16.7 (35, 36, 42, 46, 63, 64, 73, 77)
Results: validation: $(83.1 \pm 4.2)\,\%$, testing: $(72.8 \pm 6.2)\,\%$.

[M23] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 35.0 ± 21.9 (13, 21, 26, 26, 31, 37, 38, 85)
Results: validation: $(82.5 \pm 4.0)\,\%$, testing: $(72.3 \pm 8.0)\,\%$.

[M24] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 38.0 ± 24.7 (10, 12, 25, 31, 34, 51, 69, 76)
Results: validation: $(78.4 \pm 6.5)\,\%$, testing: $(72.2 \pm 9.8)\,\%$.

[M25] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG 3D
Parameters: epochs: 26.0 ± 10.9 (9, 13, 22, 24, 26, 33, 38, 39)
Results: validation: $(80.6 \pm 5.8)\,\%$, testing: $(72.0 \pm 6.9)\,\%$.

[M26] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 44.0 ± 25.5 (14, 16, 26, 39, 47, 52, 79, 79)
Results: validation: $(79.7 \pm 8.3)\,\%$, testing: $(71.9 \pm 7.1)\,\%$.

[M27] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 33.0 ± 10.9 (14, 24, 28, 33, 37, 39, 41, 49)
Results: validation: $(79.1 \pm 5.5)\,\%$, testing: $(71.9 \pm 7.2)\,\%$.

[M28] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 42.0 ± 20.3 (13, 16, 26, 48, 52, 57, 57, 65)
Results: validation: $(82.8 \pm 6.7)\,\%$, testing: $(71.7 \pm 8.2)\,\%$.

[M29] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 74.0 ± 15.1 (56, 58, 58, 75, 83, 83, 88, 94)
Results: validation: $(83.1 \pm 5.1)\,\%$, testing: $(71.4 \pm 4.5)\,\%$.

[M30] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG 3D

Parameters: epochs: 20.0 ± 10.3 (4, 8, 16, 22, 23, 24, 26, 36)
Results: validation: $(79.7 \pm 4.9)\,\%$, testing: $(71.1 \pm 6.9)\,\%$.

[M31] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 35.0 ± 18.3 (17, 20, 25, 29, 32, 37, 49, 73)
Results: validation: $(79.1 \pm 6.4)\,\%$, testing: $(70.6 \pm 5.6)\,\%$.

[M32] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 57.0 ± 23.0 (27, 28, 48, 54, 60, 67, 75, 95)
Results: validation: $(77.2 \pm 6.3)\,\%$, testing: $(68.9 \pm 4.9)\,\%$.

[M33] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 27.0 ± 15.2 (9, 13, 17, 20, 24, 34, 48, 48)
Results: validation: $(76.6 \pm 5.2)\,\%$, testing: $(68.4 \pm 6.9)\,\%$.

[M34] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG 3D
Parameters: epochs: 34.0 ± 22.3 (4, 11, 24, 27, 36, 44, 64, 64)
Results: validation: $(78.1 \pm 6.4)\,\%$, testing: $(68.0 \pm 10.2)\,\%$.

[M35] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 48.0 ± 31.5 (14, 19, 24, 38, 39, 67, 87, 96)
Results: validation: $(78.1 \pm 6.6)\,\%$, testing: $(68.0 \pm 8.8)\,\%$.

[M36] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 28.0 ± 18.5 (10, 11, 15, 16, 27, 37, 45, 61)
Results: validation: $(77.2 \pm 4.5)\,\%$, testing: $(67.7 \pm 2.3)\,\%$.

[M37] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 68.0 ± 19.6 (42, 45, 58, 71, 72, 73, 89, 98)
Results: validation: $(93.4 \pm 2.1)\,\%$, testing: $(88.7 \pm 2.4)\,\%$.

[M38] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 61.0 ± 9.7 (50, 53, 54, 57, 59, 62, 73, 77)
Results: validation: $(93.0 \pm 2.9)\,\%$, testing: $(88.5 \pm 1.8)\,\%$.

[M39] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 65.0 ± 17.6 (44, 45, 57, 61, 63, 70, 83, 95)
Results: validation: $(91.9 \pm 1.4)\,\%$, testing: $(88.2 \pm 1.7)\,\%$.

[M40] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 56.0 ± 22.4 (28, 36, 40, 50, 57, 67, 85, 89)
Results: validation: $(93.0 \pm 2.7)\,\%$, testing: $(88.1 \pm 3.4)\,\%$.

[M41] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 59.0 ± 23.8 (30, 33, 38, 57, 61, 75, 88, 89)
Results: validation: $(93.4 \pm 1.8)\,\%$, testing: $(87.0 \pm 2.4)\,\%$.

[M42] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 53.0 ± 15.1 (37, 39, 45, 48, 53, 60, 60, 84)
Results: validation: $(92.4 \pm 2.6)\,\%$, testing: $(86.6 \pm 2.9)\,\%$.

[M43] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 40.0 ± 19.3 (10, 23, 33, 35, 37, 55, 63, 64)
Results: validation: $(78.2 \pm 4.7)\,\%$, testing: $(70.5 \pm 3.8)\,\%$.

[M44] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 28.0 ± 8.0 (16, 19, 27, 28, 29, 33, 38, 38)
Results: validation: $(73.1 \pm 3.2)\,\%$, testing: $(68.7 \pm 1.9)\,\%$.

[M45] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 38.0 ± 14.1 (25, 26, 28, 37, 38, 39, 47, 68)
Results: validation: $(72.2 \pm 2.4)\,\%$, testing: $(67.2 \pm 2.7)\,\%$.

[M46] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 33.0 ± 15.5 (16, 20, 29, 29, 29, 32, 42, 66)
Results: validation: $(72.0 \pm 2.8)\,\%$, testing: $(66.7 \pm 1.6)\,\%$.

[M47] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 24.0 ± 9.4 (14, 17, 17, 19, 20, 27, 33, 41)
Results: validation: $(72.0 \pm 1.8)\,\%$, testing: $(65.2 \pm 2.9)\,\%$.

[M48] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 25.0 ± 9.2 (14, 18, 18, 25, 25, 29, 31, 43)
Results: validation: $(72.5 \pm 3.0)\,\%$, testing: $(65.0 \pm 3.7)\,\%$.

[M49] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 30.0 ± 19.4 (13, 17, 17, 20, 22, 31, 46, 70)
Results: validation: $(69.7 \pm 2.0)\,\%$, testing: $(64.5 \pm 3.0)\,\%$.

[M50] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 52.0 ± 22.1 (25, 30, 34, 46, 53, 73, 76, 81)
Results: validation: $(93.4 \pm 4.4)\,\%$, testing: $(88.4 \pm 3.5)\,\%$.

[M51] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG

Parameters: epochs: 62.0 ± 17.4 (36, 48, 56, 58, 62, 68, 84, 88)
Results: validation: $(91.9 \pm 1.6)\,\%$, testing: $(87.1 \pm 2.1)\,\%$.

[M52] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 58.0 ± 17.7 (35, 50, 51, 53, 55, 58, 61, 97)
Results: validation: $(91.5 \pm 4.9)\,\%$, testing: $(87.1 \pm 4.4)\,\%$.

[M53] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 46.0 ± 19.5 (26, 31, 39, 40, 43, 46, 59, 88)
Results: validation: $(89.8 \pm 4.6)\,\%$, testing: $(85.0 \pm 4.8)\,\%$.

[M54] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 53.0 ± 20.5 (28, 35, 43, 45, 48, 64, 82, 82)
Results: validation: $(89.6 \pm 3.7)\,\%$, testing: $(84.5 \pm 4.6)\,\%$.

[M55] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 68.0 ± 19.0 (44, 51, 54, 61, 71, 85, 86, 96)
Results: validation: $(89.2 \pm 4.8)\,\%$, testing: $(83.8 \pm 2.3)\,\%$.

[M56] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 46.0 ± 21.1 (23, 26, 27, 36, 57, 58, 59, 82)
Results: validation: $(86.2 \pm 4.8)\,\%$, testing: $(80.8 \pm 2.7)\,\%$.

[M57] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 40.0 ± 15.2 (24, 25, 30, 38, 38, 42, 60, 65)
Results: validation: $(85.8 \pm 4.0)\,\%$, testing: $(80.3 \pm 3.9)\,\%$.

[M58] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 56.0 ± 21.2 (30, 34, 43, 45, 62, 69, 72, 91)
Results: validation: $(85.4 \pm 3.9)\,\%$, testing: $(80.3 \pm 3.7)\,\%$.

[M59] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 52.0 ± 28.4 (17, 17, 24, 54, 65, 74, 80, 83)
Results: validation: $(86.2 \pm 4.9)\,\%$, testing: $(80.0 \pm 3.2)\,\%$.

[M60] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 60.0 ± 28.0 (24, 35, 37, 52, 63, 79, 94, 98)
Results: validation: $(86.0 \pm 4.4)\,\%$, testing: $(79.7 \pm 3.4)\,\%$.

[M61] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 70.0 ± 15.4 (45, 54, 66, 68, 74, 82, 83, 91)
Results: validation: $(83.0 \pm 3.0)\,\%$, testing: $(77.7 \pm 5.4)\,\%$.

[M62] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 65.0 ± 26.8 (15, 34, 68, 71, 77, 77, 81, 96)
Results: validation: $(84.1 \pm 5.1)\,\%$, testing: $(76.9 \pm 5.4)\,\%$.

[M63] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG 3D
Parameters: epochs: 50.0 ± 24.9 (18, 22, 36, 39, 59, 71, 77, 81)
Results: validation: $(81.8 \pm 3.1)\,\%$, testing: $(76.7 \pm 4.2)\,\%$.

[M64] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 59.0 ± 22.5 (32, 36, 44, 53, 62, 69, 81, 97)
Results: validation: $(82.8 \pm 4.4)\,\%$, testing: $(76.5 \pm 2.8)\,\%$.

[M65] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 59.0 ± 22.6 (29, 34, 39, 63, 65, 69, 81, 91)
Results: validation: $(82.4 \pm 3.3)\,\%$, testing: $(76.5 \pm 3.3)\,\%$.

[M66] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 68.0 ± 18.3 (40, 50, 63, 66, 67, 72, 85, 98)
Results: validation: $(80.9 \pm 4.5)\,\%$, testing: $(76.2 \pm 3.8)\,\%$.

[M67] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 45.0 ± 25.8 (18, 23, 27, 31, 44, 55, 81, 84)
Results: validation: $(83.7 \pm 2.9)\,\%$, testing: $(76.0 \pm 3.2)\,\%$.

[M68] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 62.0 ± 20.4 (31, 44, 45, 58, 75, 79, 80, 85)
Results: validation: $(83.5 \pm 4.2)\,\%$, testing: $(75.8 \pm 4.1)\,\%$.

[M69] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 49.0 ± 14.9 (26, 36, 46, 46, 47, 55, 64, 73)
Results: validation: $(81.1 \pm 4.3)\,\%$, testing: $(75.4 \pm 5.3)\,\%$.

[M70] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 53.0 ± 19.9 (23, 36, 39, 45, 67, 67, 69, 79)
Results: validation: $(82.8 \pm 4.3)\,\%$, testing: $(75.3 \pm 2.3)\,\%$.

[M71] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 40.0 ± 11.1 (29, 30, 30, 38, 39, 43, 54, 58)
Results: validation: $(82.6 \pm 3.2)\,\%$, testing: $(75.1 \pm 3.3)\,\%$.

[M72] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG

Parameters: epochs: 55.0 ± 21.8 (19, 37, 49, 57, 58, 60, 65, 94)
Results: validation: $(83.0 \pm 2.8)\,\%$, testing: $(75.0 \pm 7.6)\,\%$.

[M73] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG 3D
Parameters: epochs: 20.0 ± 13.0 (4, 9, 11, 15, 25, 26, 31, 43)
Results: validation: $(80.5 \pm 4.2)\,\%$, testing: $(74.5 \pm 3.8)\,\%$.

[M74] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG 3D
Parameters: epochs: 38.0 ± 25.6 (11, 20, 22, 23, 34, 42, 68, 84)
Results: validation: $(79.7 \pm 3.8)\,\%$, testing: $(74.1 \pm 2.2)\,\%$.

[M75] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 49.0 ± 12.5 (30, 37, 38, 47, 58, 59, 60, 62)
Results: validation: $(79.4 \pm 3.2)\,\%$, testing: $(73.4 \pm 3.5)\,\%$.

[M76] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 54.0 ± 22.6 (17, 37, 45, 46, 50, 72, 77, 84)
Results: validation: $(79.9 \pm 4.3)\,\%$, testing: $(73.2 \pm 5.6)\,\%$.

[M77] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG
Parameters: epochs: 58.0 ± 26.0 (16, 34, 46, 57, 64, 70, 88, 92)
Results: validation: $(79.0 \pm 4.5)\,\%$, testing: $(72.6 \pm 6.9)\,\%$.

[M78] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 42.0 ± 14.6 (24, 27, 29, 39, 45, 51, 56, 64)
Results: validation: $(79.7 \pm 3.0)\,\%$, testing: $(72.0 \pm 3.4)\,\%$.

[M79] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Test Features: HOG 3D
Parameters: epochs: 42.0 ± 31.4 (8, 14, 22, 25, 37, 60, 81, 91)
Results: validation: $(81.3 \pm 4.7)\,\%$, testing: $(71.8 \pm 4.2)\,\%$.

[M80] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 45.0 ± 23.1 (24, 25, 33, 38, 43, 50, 54, 96)
Results: validation: $(69.3 \pm 5.0)\,\%$, testing: $(63.5 \pm 1.5)\,\%$.

[M81] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Test Features: raw
Parameters: epochs: 32.0 ± 21.4 (11, 11, 13, 22, 39, 41, 49, 70)
Results: validation: $(71.8 \pm 1.6)\,\%$, testing: $(63.4 \pm 3.9)\,\%$.

[M82] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 56.0 ± 19.9 (22, 33, 45, 65, 65, 67, 68, 79)
Results: validation: $(95.9 \pm 0.7)\,\%$, testing: $(93.7 \pm 0.6)\,\%$.

[M83] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 41.0 ± 27.6 (13, 16, 20, 20, 48, 52, 73, 84)
Results: validation: $(95.1 \pm 0.5)\,\%$, testing: $(92.3 \pm 1.4)\,\%$.

[M84] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 49.0 ± 20.3 (20, 27, 41, 43, 54, 55, 70, 80)
Results: validation: $(93.2 \pm 1.0)\,\%$, testing: $(89.7 \pm 1.1)\,\%$.

[M85] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 50.0 ± 22.5 (33, 37, 38, 40, 41, 49, 66, 100)
Results: validation: $(91.4 \pm 0.8)\,\%$, testing: $(89.3 \pm 0.9)\,\%$.

[M86] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 32.0 ± 5.3 (22, 27, 29, 31, 33, 36, 37, 37)
Results: validation: $(90.7 \pm 1.5)\,\%$, testing: $(88.6 \pm 1.1)\,\%$.

[M87] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 48.0 ± 16.6 (24, 27, 36, 52, 56, 58, 63, 67)
Results: validation: $(92.2 \pm 1.5)\,\%$, testing: $(88.4 \pm 0.7)\,\%$.

[M88] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 30.0 ± 15.1 (9, 17, 19, 33, 34, 34, 36, 58)
Results: validation: $(90.8 \pm 1.4)\,\%$, testing: $(87.9 \pm 1.5)\,\%$.

[M89] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 48.0 ± 21.5 (25, 31, 33, 34, 41, 71, 71, 78)
Results: validation: $(90.8 \pm 1.0)\,\%$, testing: $(87.3 \pm 1.4)\,\%$.

[M91] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 25.0 ± 23.9 (10, 11, 11, 14, 23, 25, 25, 82)
Results: validation: $(87.7 \pm 0.5)\,\%$, testing: $(86.3 \pm 1.5)\,\%$.

[M93] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 35.0 ± 32.2 (9, 11, 15, 20, 24, 28, 83, 89)
Results: validation: $(88.3 \pm 1.3)\,\%$, testing: $(85.8 \pm 1.4)\,\%$.

[M96] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 11.0 ± 2.1 (7, 9, 10, 10, 12, 12, 13, 13)
Results: validation: $(82.1 \pm 1.4)\,\%$, testing: $(81.1 \pm 2.0)\,\%$.

[M97] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw

Parameters: epochs: 20.0 ± 16.4 (10, 12, 13, 14, 14, 19, 20, 60)
Results: validation: $(82.3 \pm 1.2)\,\%$, testing: $(80.1 \pm 1.8)\,\%$.

[M101] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 46.0 ± 20.6 (23, 26, 34, 35, 44, 66, 66, 77)
Results: validation: $(83.3 \pm 2.3)\,\%$, testing: $(78.8 \pm 2.6)\,\%$.

[M103] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 38.0 ± 14.0 (19, 28, 31, 32, 35, 42, 49, 64)
Results: validation: $(81.9 \pm 0.9)\,\%$, testing: $(78.2 \pm 1.9)\,\%$.

[M106] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 35.0 ± 14.8 (15, 26, 26, 27, 29, 47, 53, 55)
Results: validation: $(81.5 \pm 1.6)\,\%$, testing: $(77.5 \pm 2.4)\,\%$.

[M111] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: raw
Parameters: epochs: 25.0 ± 10.2 (14, 14, 21, 24, 25, 25, 37, 43)
Results: validation: $(80.0 \pm 2.7)\,\%$, testing: $(75.5 \pm 1.1)\,\%$.

[M90] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 57.0 ± 26.3 (15, 33, 36, 59, 71, 76, 84, 85)
Results: validation: $(89.1 \pm 1.1)\,\%$, testing: $(86.8 \pm 1.1)\,\%$.

[M92] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 52.0 ± 27.1 (22, 33, 33, 43, 50, 51, 93, 94)
Results: validation: $(90.3 \pm 1.4)\,\%$, testing: $(86.2 \pm 0.7)\,\%$.

[M94] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 77.0 ± 14.8 (56, 60, 71, 74, 75, 87, 94, 96)
Results: validation: $(86.0 \pm 0.5)\,\%$, testing: $(83.1 \pm 1.0)\,\%$.

[M95] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 58.0 ± 18.3 (32, 38, 48, 58, 63, 64, 67, 90)
Results: validation: $(87.0 \pm 1.1)\,\%$, testing: $(83.1 \pm 1.5)\,\%$.

[M98] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 15.0 ± 4.9 (8, 11, 13, 14, 15, 16, 19, 24)
Results: validation: $(81.5 \pm 1.6)\,\%$, testing: $(79.6 \pm 1.3)\,\%$.

[M99] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 43.0 ± 16.5 (24, 27, 34, 41, 41, 48, 58, 74)
Results: validation: $(83.1 \pm 1.8)\,\%$, testing: $(79.5 \pm 2.4)\,\%$.

[M100] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 32.0 ± 10.7 (17, 18, 29, 33, 35, 36, 43, 47)
Results: validation: $(80.8 \pm 1.6)\,\%$, testing: $(79.4 \pm 1.2)\,\%$.

[M102] *LSTM Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 65.0 ± 21.7 (45, 45, 48, 48, 66, 85, 91, 94)
Results: validation: $(82.1 \pm 1.7)\,\%$, testing: $(78.5 \pm 1.8)\,\%$.

[M104] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG 3D
Parameters: epochs: 14.0 ± 4.1 (8, 10, 12, 13, 15, 16, 17, 21)
Results: validation: $(79.5 \pm 1.5)\,\%$, testing: $(78.0 \pm 1.9)\,\%$.

[M105] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 18.0 ± 8.1 (10, 12, 14, 15, 17, 19, 20, 36)
Results: validation: $(79.0 \pm 1.3)\,\%$, testing: $(77.6 \pm 0.8)\,\%$.

[M107] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 40.0 ± 27.5 (3, 14, 22, 26, 46, 59, 67, 80)
Results: validation: $(81.3 \pm 1.6)\,\%$, testing: $(77.4 \pm 1.4)\,\%$.

[M108] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 43.0 ± 17.1 (27, 30, 31, 32, 35, 57, 66, 67)
Results: validation: $(78.6 \pm 1.6)\,\%$, testing: $(77.2 \pm 1.4)\,\%$.

[M109] *DenseNet 121 Model, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 42.0 ± 25.8 (8, 8, 27, 45, 46, 58, 68, 76)
Results: validation: $(80.6 \pm 1.7)\,\%$, testing: $(76.5 \pm 1.8)\,\%$.

[M110] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 14.0 ± 2.2 (11, 12, 13, 13, 13, 15, 17, 17)
Results: validation: $(77.8 \pm 2.5)\,\%$, testing: $(76.2 \pm 1.9)\,\%$.

[M112] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG 3D
Parameters: epochs: 18.0 ± 6.1 (11, 14, 15, 17, 17, 20, 21, 31)
Results: validation: $(76.8 \pm 1.7)\,\%$, testing: $(75.2 \pm 1.0)\,\%$.

[M113] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 30.0 ± 12.3 (18, 20, 20, 22, 29, 35, 42, 52)
Results: validation: $(77.6 \pm 2.0)\,\%$, testing: $(74.8 \pm 2.0)\,\%$.

[M114] *CNN Model with 1D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG

Parameters: epochs: 37.0 ± 12.9 (21, 24, 30, 33, 36, 48, 51, 56)
Results: validation: $(77.0 \pm 1.1)\,\%$, testing: $(74.3 \pm 1.4)\,\%$.

[M115] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG
Parameters: epochs: 23.0 ± 12.0 (14, 14, 14, 17, 24, 24, 24, 50)
Results: validation: $(76.6 \pm 1.9)\,\%$, testing: $(73.9 \pm 1.6)\,\%$.

[M116] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG 3D
Parameters: epochs: 16.0 ± 6.4 (6, 11, 13, 15, 17, 21, 24, 24)
Results: validation: $(75.4 \pm 1.8)\,\%$, testing: $(73.4 \pm 1.3)\,\%$.

[M117] *CNN Model with 2D convolution, using k-fold cross-validation*
Dataset: Simple Call Seq Features: HOG 3D
Parameters: epochs: 18.0 ± 4.5 (13, 13, 16, 18, 18, 18, 21, 27)
Results: validation: $(74.8 \pm 1.4)\,\%$, testing: $(72.2 \pm 0.7)\,\%$.

List of Tables

List of Figures

List of Acronyms

ADAM adaptive moment estimation. 21, 67

AI artificial intelligence. 15, 67

ANN artificial neural network. 11, 20–22, 25, 27, 48, 49, 67

CNN convolutional neural network. 11, 24, 34, 67

CV cross-validation. 13, 26, 48, 66, 67

DFT discrete Fourier transform. 18, 67

DL deep learning. 2, 6, 11, 20, 21, 28, 29, 33, 35, 36, 39, 43, 46–50, 67

GPU graphics processing unit. 3, 29, 67

HOG histogram of oriented gradients. 2, 19, 31–38, 40–44, 47, 51, 65–67

LRP layer-wise relevance propagation. 27, 30, 40, 41, 44, 46, 67

LSTM long short-term memory. 19, 23, 29–32, 34–38, 40, 41, 43, 45, 47, 48, 66, 67

mcl min cover length. 43, 44, 51, 67

ML machine learning. 6–8, 10, 11, 15, 16, 18–20, 25–27, 47, 66, 67

MLP multi layer perceptron. 21, 22, 24, 67

NLP natural language processing. 6, 7, 67

nrs noise reduction sensitivity. 33, 34, 36, 40, 67

RMS root mean squared. 16, 67

RNN recurrent neural network. 23, 29, 67

SGD stochastic gradient descent. 21, 67

UMAP uniform manifold approximation and projection. 49, 67

xpps horizontal (x) pixels per second. 37–40, 43, 67

Bibliography

[VW20] Sonja C. Vernes and Gerald S. Wilkinson. *Behaviour, biology and evolution of vocal learning in bats*. 2020. DOI: 10.1098/rstb.2019.0061.

[Knö14] Mirjam Knörnschild. *Vocal production learning in bats*. 2014. DOI: 10.1016/j.conb.2014.06.014.

[Kan+20] Yue Kang et al. *Natural language processing (NLP) in management research: A literature review*. Apr. 2020. DOI: 10.1080/23270012.2020.1756939. URL: https://www.tandfonline.com/doi/abs/10.1080/23270012.2020.1756939.

[Sto+19] Dan Stowell et al. "Automatic acoustic identification of individuals in multiple species: Improving identification across recording conditions". In: *Journal of the Royal Society Interface* 16.153 (2019). ISSN: 17425662. DOI: 10.1098/rsif.2018.0940.

[Wir+19] Morgan Wirthlin et al. *A Modular Approach to Vocal Learning: Disentangling the Diversity of a Complex Behavioral Trait*. Oct. 2019. DOI: 10.1016/j.neuron.2019.09.036.

[Amo+19] Dario Amodei et al. "AI and Compute". In: *Blog Open AI* (2019), pp. 1–11. URL: https://openai.com/blog/ai-and-compute/%20https://blog.openai.com/ai-and-compute/.

[HCF02] Marc D. Hauser, Noam Chomsky, and W. Tecumseh Fitch. *Neuroscience: The faculty of language: What is it, who has it, and how did it evolve?* Nov. 2002. DOI: 10.1126/science.298.5598.1569. URL: https://science.sciencemag.org/content/298/5598/1569%20https://science.sciencemag.org/content/298/5598/1569.abstract.

[Tec10] W. Tecumseh Fitch. *The evolution of language*. Vol. 9780521859936. Cambridge University Press, Jan. 2010, pp. 1–611. ISBN: 9780511817779. DOI: 10.1017/CBO9780511817779. URL: https://www.cambridge.org/core/books/evolution-of-language/2347BC6741639875250495BA3435056F.

[Fit18] W. Tecumseh Fitch. *The biology and evolution of speech: A comparative analysis*. Jan. 2018. DOI: 10.1146/annurev-linguistics-011817-045748. URL: https://doi.org/10.1146/annurev-linguistics-.

[FHB10] W. Tecumseh Fitch, Ludwig Huber, and Thomas Bugnyar. *Social Cognition and the Evolution of Language: Constructing Cognitive Phylogenies*. Mar. 2010. DOI: 10.1016/j.neuron.2010.03.011. URL: /pmc/articles/PMC4415479/?report=abstract%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4415479/.

[BV04] Oliver Behr and Otto Von Helversen. "Bat serenades - Complex courtship songs of the sac-winged bat (Saccopteryx bilineata)". In: *Behavioral Ecology and Sociobiology* 56.2 (June 2004), pp. 106–115. ISSN: 03405443. DOI: 10.1007/s00265-004-0768-7. URL: https://link.springer.com/article/10.1007/s00265-004-0768-7.

[EPV89] MARCEL Eens, RIANNE Pinxten, and RUDOLF Verheyen. "Temporal and sequential organization of song bouts in the starling". In: *Ardea* 77.6 (1989).

[PM71] Roger S. Payne and Scott McVay. *Songs of humpback whales*. Aug. 1971. DOI: 10.1126/science.173.3997.585. URL: https://science.sciencemag.org/content/173/3997/585%20https://science.sciencemag.org/content/173/3997/585.abstract.

[Wei+14] Michael Weiss et al. "The use of network analysis to study complex animal communication systems: A study on nightingale song". In: *Proceedings of the Royal Society B: Biological Sciences* (2014). ISSN: 14712954. DOI: 10.1098/rspb.2014.0460.

[Ker+16] Arik Kershenbaum et al. "Acoustic sequences in non-human animals: A tutorial review and prospectus". In: *Biological Reviews* (2016). ISSN: 1469185X. DOI: 10.1111/brv.12160.

[MPS04] Gerhard Manteuffel, Birger Puppe, and Peter C. Schön. "Vocalization of farm animals as a measure of welfare". In: *Applied Animal Behaviour Science* (2004). ISSN: 01681591. DOI: 10.1016/j.applanim.2004.02.012.

[PJ12] Christopher I. Petkov and Erich D. Jarvis. *Birds, primates, and spoken language origins: Behavioral phenotypes and neurobiological substrates*. 2012. DOI: 10.3389/fnevo.2012.00012.

[Jan14] Vincent M. Janik. *Cetacean vocal learning and communication*. 2014. DOI: 10.1016/j.conb.2014.06.010.

[RC14] Colleen Reichmuth and Caroline Casey. *Vocal learning in seals, sea lions, and walruses*. 2014. DOI: 10.1016/j.conb.2014.06.011.

[Poo+05] Joyce H. Poole et al. "Elephants are capable of vocal learning". In: *Nature* (2005). ISSN: 00280836. DOI: 10.1038/434455a.

[MB20] Pedro Tiago Martins and Cedric Boeckx. *Vocal learning: Beyond the continuum*. Mar. 2020. DOI: 10.1371/JOURNAL.PBIO.3000672. URL: https://doi.org/10.1371/journal.pbio.3000672.

[Vih14] Marilyn May Vihman. *Phonological Development: the first two years*. 2014. ISBN: 9781118342794.

[OLL80] D.K. OLLER. "THE EMERGENCE OF THE SOUNDS OF SPEECH IN INFANCY". In: *Child Phonology*. Elsevier, Jan. 1980, pp. 93–112. DOI: 10.1016/b978-0-12-770601-6.50011-5.

[KM96] Patricia K. Kuhl and Andrew N. Meltzoff. "Infant vocalizations in response to speech: Vocal imitation and developmental change". In: *The Journal of the Acoustical Society of America* (1996). ISSN: 0001-4966. DOI: 10.1121/1.417951.

[Knö+10] Mirjam Knörnschild et al. "Complex vocal imitation during ontogeny in a bat". In: *Biology Letters* 6.2 (Apr. 2010), pp. 156–159. ISSN: 1744-9561. DOI: 10.1098/rsbl.2009.0685. URL: https://royalsocietypublishing.org/doi/10.1098/rsbl.2009.0685.

[KBV06] Mirjam Knörnschild, Oliver Behr, and Otto Von Helversen. "Babbling behavior in the sac-winged bat (Saccopteryx bilineata)". In: *Naturwissenschaften* (2006). ISSN: 00281042. DOI: 10.1007/s00114-006-0127-9.

[Zha+19] Kangkang Zhang et al. "Comparing context-dependent call sequences employing machine learning methods: An indication of syntactic structure of greater horseshoe bats". In: *Journal of Experimental Biology* (2019). ISSN: 00220949. DOI: 10.1242/jeb.214072.

[Qia+17] Kun Qian et al. "Active learning for bird sound classification via a kernel-based extreme learning machine". In: *The Journal of the Acoustical Society of America* (2017). ISSN: 0001-4966. DOI: 10.1121/1.5004570.

[Tch+04] O. Tchernichovski et al. "Studying the song development process: Rationale and methods". In: *Annals of the New York Academy of Sciences*. 2004. DOI: 10.1196/annals.1298.031.

[Mol+08] Csaba Molnár et al. “Classification of dog barks: A machine learning approach”. In: *Animal Cognition* (2008). ISSN: 14359448. DOI: 10.1007/s10071-007-0129-9.

[CMN19] Kevin R. Coffey, Russell G. Marx, and John F. Neumaier. “DeepSqueak: a deep learning-based system for detection and analysis of ultrasonic vocalizations”. In: *Neuropsychopharmacology* (2019). ISSN: 1740634X. DOI: 10.1038/s41386-018-0303-6.

[PGG10] Luca Pozzi, Marco Gamba, and Cristina Giacoma. “The use of artificial neural networks to classify primate vocalizations: A pilot study on black lemurs”. In: *American Journal of Primatology* (2010). ISSN: 02752565. DOI: 10.1002/ajp.20786.

[Tur+16] Hjalmar K. Turesson et al. “Machine learning algorithms for automatic classification of marmoset vocalizations”. In: *PLoS ONE* (2016). ISSN: 19326203. DOI: 10.1371/journal.pone.0163041.

[STG20] Tim Sainburg, Marvin Thielk, and Timothy Q. Gentner. “Finding, visualizing, and quantifying latent structure across diverse animal vocal repertoires”. In: *PLoS Computational Biology* (2020). ISSN: 15537358. DOI: 10.1371/journal.pcbi.1008228.

[Mac+18] Oisin Mac Aodha et al. “Bat detective—Deep learning tools for bat acoustic signal detection”. In: *PLoS Computational Biology* 14.3 (2018), pp. 1–19. ISSN: 15537358. DOI: 10.1371/journal.pcbi.1005995.

[Che+20] Xing Chen et al. “Automatic standardized processing and identification of tropical bat calls using deep learning approaches”. In: *Biological Conservation* (2020). ISSN: 00063207. DOI: 10.1016/j.biocon.2019.108269.

[He+16a] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2016. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.90.

[Wae19] Gilles Waeber. “Bird Voice Deep Learning”. In: (2019).

[Hua+17] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vol. 2017-Janua. Institute of Electrical and Electronics Engineers Inc., Aug. 2017, pp. 2261–2269. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.243. URL: http://arxiv.org/abs/1608.06993.

[Kan+94] Jagmeet S. Kanwal et al. “Analysis of acoustic elements and syntax in communication sounds emitted by mustached bats”. In: *Journal of the Acoustical Society of America* (1994). ISSN: NA. DOI: 10.1121/1.410273.

[Lat+19] Ella Z. Lattenkamp et al. “The Vocal Repertoire of Pale Spear-Nosed Bats in a Social Roosting Context”. In: *Frontiers in Ecology and Evolution* 7.APR (Apr. 2019), p. 116. ISSN: 2296-701X. DOI: 10.3389/fevo.2019.00116. URL: https://www.frontiersin.org/article/10.3389/fevo.2019.00116/full.

[BKV09] Oliver Behr, Mirjam Knörnschild, and Otto Von Helversen. “Territorial counter-singing in male sac-winged bats saccopteryx bilineata: Low-frequency songs trigger a stronger response”. In: *Behavioral Ecology and Sociobiology* (2009). ISSN: 03405443. DOI: 10.1007/s00265-008-0677-2.

[Beh+06] Oliver Behr et al. “Territorial songs indicate male quality in the sac-winged bat Saccopteryx bilineata (Chiroptera, Emballonuridae)”. In: *Behavioral Ecology* (2006). ISSN: 10452249. DOI: 10.1093/beheco/arl013.

[EK13] Maria Eckenweber and Mirjam Knörnschild. “Social influences on territorial signaling in male greater sac-winged bats”. In: *Behavioral Ecology and Sociobiology* (2013). ISSN: 03405443. DOI: 10.1007/s00265-013-1483-z.

[McF+20] Brian McFee et al. *librosa.effects.split — librosa 0.8.0 documentation*. 2020. URL: https://librosa.org/doc/latest/generated/librosa.effects.split.html?highlight=split#librosa.effects.split.

[RG15] Alain Rakotomamonjy and Gilles Gasso. “Histogram of gradients of time-frequency representations for audio scene classification”. In: *IEEE/ACM Transactions on Audio Speech and Language Processing* (2015). ISSN: 23299290. DOI: 10.1109/TASLP.2014.2375575.

[YK17] Wenjun Yang and Sridhar Krishnan. “Combining Temporal Features by Local Binary Pattern for Acoustic Scene Classification”. In: *IEEE/ACM Transactions on Audio Speech and Language Processing* (2017). ISSN: 23299290. DOI: 10.1109/TASLP.2017.2690558.

[Sun+14] Bo Sun et al. “Combining multimodal features with hierarchical classifier fusion for emotion recognition in the wild”. In: *ICMI 2014 - Proceedings of the 2014 International Conference on Multimodal Interaction*. 2014. ISBN: 9781450328852. DOI: 10.1145/2663204.2666272.

[DT05] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*. 2005. ISBN: 0769523722. DOI: 10.1109/CVPR.2005.177.

[Sur+18] Thattapon Surasak et al. “Histogram of oriented gradients for human detection in video”. In: *Proceedings of 2018 5th International Conference on Business and Industrial Research: Smart Technology for Next Generation of Information, Engineering, Business and Social Science, ICBIR 2018*. 2018. ISBN: 9781538652541. DOI: 10.1109/ICBIR.2018.8391187.

[TT09] Kengo Terasawa and Yuzuru Tanaka. “Slit style HOG feature for document image word spotting”. In: *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*. 2009. ISBN: 9780769537252. DOI: 10.1109/ICDAR.2009.118.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[Nie15] M A Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: https://books.google.ch/books?id=STDBswEACAAJ.

[LBH15] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. *Deep learning*. 2015. DOI: 10.1038/nature14539.

[Ola15] Christopher Olah. “Understanding LSTM Networks [Blog]”. In: *Web Page* (2015). ISSN: 0717-6163.

[SZ15] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. 2015.

[He+16b] Kaiming He et al. “Identity mappings in deep residual networks”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2016. ISBN: 9783319464923. DOI: 10.1007/978-3-319-46493-0{_}38.

[Hua+] Gao Huang et al. *DenseNet/models at master · liuzhuang13/DenseNet · GitHub*. URL: https://github.com/liuzhuang13/DenseNet/tree/master/models.

[Sri+14] Nitish Srivastava et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Tech. rep. 2014, pp. 1929–1958. DOI: 10.5555/2627435.2670313.

[ZY15] Yongli Zhang and Yuhong Yang. “Cross-validation for selecting a model selection procedure”. In: *Journal of Econometrics* (2015). ISSN: 18726895. DOI: 10.1016/j.jeconom.2015.02.006.

[CGN16] Juan Gabriel Colonna, João Gama, and Eduardo F. Nakamura. “How to correctly evaluate an automatic bioacoustics classification method”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2016. ISBN: 9783319446356. DOI: 10.1007/978-3-319-44636-3{_}4.

[Oka+19] Masaki Okawa et al. “Audio classification of bit-representation waveform”. In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*. 2019. DOI: 10.21437/Interspeech.2019-1855.