Masterarbeit - Protokoll

Vincent Kather

Sommersemester 2022

Contents

List of Figures

List of Tables

Papers

Datum: 4.Mai

.1 Humpback whale song hierarchical structure: Historical context and discussion of current classification issues

- humpback whale songs hypothesized to play a role in female attraction or mediate male-male interactions [3]
- four core features have been observed:
    1. populations of males that are in acoustics contact sing similar songs
    2. hierarchical structure is observed globally, details vary following isolation of populations from one another
    3. song patterns change over time resulting from indivieual males modifying the spectral and temporal features of units, as well as order and repetition
    4. males in acoustics contact include similar changes in songs, even as these change over time
- variations in song units and patterns, are important for understanding the function of song within this species and the influence of sexual selection on singing behavior
- "There is clearly a need for the development of automated classification systems in order to handle large data sets of recordings; however, no system has yet been demonstrated which compares with the pattern recognition skills of the human brain." - ich bin dran
    - song session $\Longrightarrow$ series of songs with pauses less than a minute
    - song $\Longrightarrow$ combination of multiple distinc themes
    - theme $\Longrightarrow$ similar phrases comprise a theme
    - phrase $\Longrightarrow$ set of units
    - subphrase $\Longrightarrow$ sequence of one or more units, sometimes repeated in series
    - motif $\Longrightarrow$ units of repetition
      $\hookrightarrow$ similar: containing only one type of unit
      $\hookrightarrow$ dissimilar: containing two or more different units, repeated in combination
    - unit $\Longrightarrow$ shortest continuous sound
    - subunit $\Longrightarrow$ pulses to rapid to be individually discriminated
- syllables is ambiguous and thus shouldn't be used, instead use unit
- die unterteilung in phrases ist sehr subjektiv, da zusammengehörigkeiten nicht objektiv begründet werden können

- desplite challenges in labeling and delineating phrases, phrase duration is very stable over years compared to songs and other structural elements
  $\hookrightarrow$ strongly suggests, that phrases are essential elements of communication to humpback whales

- guideline to delineating phrases:

  1. consectutive units, similar structure shouldn't be seperated within a phrase but bundled as parts of a subphrase
  2. phrases should be seperated so that there is not an incomplete phrase at the end of a sequence
  3. transitional phrases entail units of two different phrase types, which subphrases occur previously and subsequently.
     $\longrightarrow$ Example: ab ab ab ad cd cd cd
     $\longrightarrow$ in this case the ad woule be the transitional phrase
  4. care should be taken when trying to distinguish between a inherent variation within a phrase and changing of units, structure or composition to a point where the definition of a new phrase seems more sensible
     $\longrightarrow$ spectrograms should be provided to support the decision
  5. phrase duration should be measured from the onset of one phrase until the onset of the subsequent phrase
     $\longrightarrow$ variation within phrase duration is the smallest compared to that of other structures within a population
  6. phrase delineation should always be based on recordings of multiple individuals rather than a single isolated individual

- fundamental theme is a concept that is ephemeral as so much changes between seasons

- Payne and Payne:

  - static themes: sequence of nearly identical phrases
  - shifting themes: successive phrases evolve from one to the other
  - unpatterned themes: variable number of units lack organisation cannot be subdivided into repeating phrases —> rather rare
    $\longrightarrow$ theme consists of a single non repeated phrase

- structure can be preserved over the span of several years, even though the individual units undergo changes

- song structure is sometimes cyclical, meaning that one theme always leads to another and in the end back to the first, this is however only sometimes the case

- there seems to be a general rule of certain themes leading to other themes, however, exceptions to the rule do exist

- in bird singing literature, songs are identified as having longer pauses between them compared to elements within the song

- song birds might be a working analogon to humpback themes
- song length increased due to acoustic disturbances
  $\hookrightarrow$ although there are opposing results
  $\longrightarrow$ seeing that the song duration metric is complicated, singers could also increase the number of repetitions due to the occurrence of noise
  $\longrightarrow$ increases in song duration are more likely the result of more variation in theme sequences, singing themes that would not otherwise occur sequentially before arriving back at the beginning, if there is a beginning
  $\longrightarrow$ as soon as theme order is less stable, quantifying a song length becomes difficult
  $\hookrightarrow$ the underlying problem is, that the definition of a song is vague, is it the occurrence of a specific theme labeled as 'beginning', is it a sequence of individual themes? this decision greatly impacts song duration and theme composition within a song
  $\hookrightarrow$ if the theme order is invariant song definition is more stable and the subsequent metrics of song duration make sense, if however theme order is variant and no hierarchy seems apparent the meaning of a 'song' and the consequential metrics are problematic
- males seem to switch between different phrase types a lot more in the presence of another singer
  $\longrightarrow$ similar observations on birds might indicate an escalation of aggression between two males

.2 Garland-2020-Frontiers in Psychology

- study of bird songs has provided information on learning mechanisms [4]
  $\longrightarrow$ songs are learned rather than inherited
- understanding of cultural evolution through social learning and underlying hormonal and neural pathways
- production learning: a form of social learning, resulting from experience with signals of other individuals, an animal learns to modify the form of its own signal
  $\longrightarrow$ considered rare in vertebrates (wirbeltiere)
- only one songbird species has a way of creating new songs on a yearly bases, similar to humpbacks $\Longrightarrow$ the corn bunting (Emberiza (Miliaria) calandra)
- whether songs serve the purpose of finding a mate, male-male interaction, or multi-message signal is still debated
- humpback songs sessions can last for many hours
- it's assumed that there is a worldwide inherited repertoire of song units, and new songs are still comprised of these units but changed in their order and structure
- humpbacks are considered to be *eventual variety* singers $\Longrightarrow$ the repeat phrases before moving to the next theme
- males in a given population will sing the same song type
  $\hookrightarrow$ strong cultural conformity
- no song has reappeared (since recording started)
- the changing and creating of new songs and what individual plays a key role is still elusive
- song sharing occurs for three reasons:
  1. sharing through shared breeding grounds/migratory routes
  2. sharing through males visiting more than one wintering ground in consecutive years
  3. sharing through males visiting more than one breeding ground within a breeding season
- *song revolution* $\Longrightarrow$ song from one population appears in other population and rapidly replaces it entirely
  $\longrightarrow$ this process occurs regularly with songs taking 2 years to spread from east Australia, where they emerge all the way to French Polynesia
- in South Pacific several songs can be existent at the same time
- in North Pacific populations are more constrained by landmasses, thus fewer variations in songs coexist
- new songs appear to spread from larger populations to smaller ones

- the quest for novelty is driven by sexual selection, however it is constrained, otherwise empirical findings would show much more diversification of songs
- a *switch when similar* rule seems to guide the application of new songs
  $\hookrightarrow$ changes from old to new songs happen at units that are similar in both old and new
  $\hookrightarrow$ at the same time a remembering of last seasons song seems to exist, preventing the humpback from switching to old songs even though units are similar
  $\longrightarrow$ similar discrimination processes exist in songbirds learning process
- while a song evolves, it increases in complexity, only to be replaced by a far simpler song in the next *song revolution*
- song complexity may resemble cognitive skills and thus used by prospective mates
- differences between corn buntings and humpbacks song learnings:
  - corn bunting songs change evolutionary $\Longrightarrow$ new songs have similarities with old songs
    $\hookrightarrow$ humpback song changes are revolutionary $\Longrightarrow$ new songs have not similarities with old songs
  - population size of birds is roughly 100 s and for humpbacks is 1000-40000
    $\hookrightarrow$ humpback songs transported tens of kms
    $\longrightarrow$ bird songs reach tens of males, humpback songs reach hundreds of males
    $\hookrightarrow$ for high fidelity audio humpbacks would need to be within 10 km of source to pick up on high freq information, so audience size might be comparable
  - new songs can come from isolated males from other populations for humpbacks, whereas they come from within the same population for corn buntings
  - corn buntings song change seems to be triggered by reproductive success i.e. successful males will most likely be copied from by other birds
    $\longrightarrow$ mating success is not as obvious for humpbacks and thus a less likely metric to decide whom to copy
    $\hookrightarrow$ research for humpback still in progress, one hypothesis states that whales might try to interrupt singers that escort a female and if they fail to succeed in interrupting them, they instead copy from them
- similarities between corn buntings and humpbacks:
  - similar song structures
  - song evolution (of the same song) is most probably the result of production errors in both species
    $\hookrightarrow$ however does not explain revolutions
- whether humpback songs convey fitness information that females use in mate choice is an open research question
- decreasing population sizes for corn buntings may also influence the origin of new songs
  $\hookrightarrow$ flourishing population sizes for many humpback populations might influence song learning in a different way

- "Unfortunately, no songbird or any other animal species to date have exhibited the rapid and repeated population-wide replacement of a cultural phenotype as observed in humpback song revolutions."
- a comparison between humpback song revolutions and rapid cultural change in human pop music and slang change might be worth looking into

.3 Terrestrial, Semi-aquatic, and Fully Aquatic Mammal Sound Production Mechanisms

- Author: Joy S. Reidnberg [8]
- odontocedes can use echolocation sounds for navigation and prey tracking
- all mammals use pneumatic machanisms to generate sounds
- larynx evolved to keep water out of a buoyancy organ in fish
  $\longrightarrow$ today its main function remains protective, but in terrestrial mammals it prevents incursions of foreign material into the lungs
- vocal cords, as well as cartilages act as splashguards, deflecting food and water away from the opening. They interlock the larynx with the rear of the nasal vacity
  $\longrightarrow$ this interlock channels airflow through the larynx and seperates it from the swallowing pathway $\Longrightarrow$ this division of airspace allows breathing and swallowing $\Longrightarrow$ a crucial feat to detect predetors while feeding
- one of the last features to be added to the larynx was phonation $\Longrightarrow$ producing the fundamental frequency of vocalizations
- the vocal cords are rather folds, stacked tissue of varying density periodically compressing air
  $\longrightarrow$ the geometry of the supralaryngeal vocal tract (vocal tract downstream of larynx) adds resonances and thus harmonics of the fundamental frequency
  $\hookrightarrow$ vocal folds are the source, supralaryngeal vocal tract is the filter
- semiaquatic mammals (spend time both on land and in water: hippopotamus, seal, polar bear, otters, etc.) keep their vocal folds shut under water to prevent drowning
  $\longrightarrow$ most solely vocalize on land
  $\hookrightarrow$ exception: hippopotamus and pinnipeds (seals, sea lions, walruses)
  - hippos produce sounds in air and underwater
  - hippos have their vocal folds oriented parallel (rather than perpendicular) to the tracheal airflow
  - this anatomy is similar to their close relatives the cetaceans
  - hippos can produce sounds with their moths open without being in danger of drowning
    $\longrightarrow$ their orientation of vocal folds alongside tall vartilages tha interlock the larynx with the nasal cavity protects agains water intake
  - pinnipeds vocal folds may be oriented parallel (seal lion) or perpendicular (seal) to the tracheal airflow
    $\hookrightarrow$ open-mouthed vocalizations are dangerous $\Longrightarrow$ vocaliation mostly occurs when the mouth is closed
- fully aquatic mammals (sirenians (manatees and gugongs) and cetaceans) produce sound nearly exclusively underwater

- U shaped pair of vocal folds oriented parallel to tracheal airflow
- if the front of the larynx is closed, air is led through epiglottic and corniculate cartilages $\Longrightarrow$ diversion of airflow
  $\longrightarrow$ airflow causes fold vibrations, leading to low-frequency sounds
  $\longrightarrow$ air can be recycled for next vocalizations
- if the front of the larynx is opened, vocal folds are closed, air passes above the epiglottis and between piared flaps of corniculate cartilage
  $\longrightarrow$ flaps may clap together and produce pulsed sounds
- the respiratory tract of cetaceans has evolved unlike that of sirenians, causing cetaceans to be able to vocalize with open mouths, whereas sirenians can't
- unlike terestrial mammals, cetaceans primarily utilize surrounding tissues rather than air in the supralaryngeal vocal tract to carry vibrations through neck and head outwards
  $\longrightarrow$ the density of the surrounding tissues is similar to that of seawater, thus smaller loss in transmission due to change in impedance
- mysticetes and odontocetes have developed differing sound generation mechanisms, as they have evolved along different trajectories.
  $\longrightarrow$ mysticetes produce infrasound
  $\longrightarrow$ odontocetes produce ultrasound
- mysticetes and odontocetes have undergone an evolutionary divergence as their vocal folds are both parallel to the tracheal airflow but 180° reversed
- **Mysticetes:**
  * primarilly produce infrasound to be able to communicate over vast distances
  * similar to other mammals mysticetes produce sound laryngeally
  * have evolved tall corniculate cartilages that also protext the larynx from behind, creating a circumferential protection around the larynx
    $\longrightarrow$ food and water are swallowed past the larynx but cannot slip through and end up accidentally inhaled
  * larynx is larger than either one of the whales lungs
    $\longrightarrow$ larynx volume and surface area allows for loud and low-frequency sounds, amplified by larger resonant volumes corresponding with longer wavelengths and increased amplification
  * Laryngeal sac also plays a crucial role in respiratory and buoyancy control
  * laryngeal sac is expanded as air is used to vocalize and contracted, sending the air back to the lungs to be reused for vocalization
  * diving whales decending to deep depths require large volumes as the increase in ambient pressure with decending decreases the air volume, resulting in limited abilities in terms of amplification and low-frequency sound generation
  * Laryngeal sac walls can vibrate, transferring the vibrations through the overlying throat muscles, blubber, and skin into the surrounding water
  * opposite the laryngeal opening, a pair of tissue flaps can be clapped together to generate pulsed sounds

  $\longrightarrow$ clapping, a more broadband, especially high frequency containing signal, can be used to echolocate
  * video of mysticete larynx: https://youtu.be/rysR5SNwrn8
  * see fig. 1

- **Odontocetes:**
  * generate high frequency sounds used for navigation and finding prey
  * use the nasal region to generate clicks
  * too have a complex structured larynx with an undetermined function
    $\longrightarrow$ echolocation sounds are produced in the nasal region
    $\hookrightarrow$ larynx might be important for production of communication sounds, not however for echolocation
  * odontocete larynx can be open while protecting the respiratory tract. shape is similar to snorkel.
    $\longrightarrow$ the complexity of the larynx suggests that it must have an essential and nearly constant role for airflow, and as no breathing is occurring underwater its most likely necessary for $\Longrightarrow$ sound production.
    $\hookrightarrow$ the level of protection is superior to that of mysticetes, allowing odontocetes to swim upside down while making noises and still not getting water into the respiratory tract
  * Sound production through the larynx is theoretically possible (yet never observed or recorded!) through three options:
    1. air flows through the larynx and causes vibrations of laryngeal tissue
    2. air is simply channeled constantly to the nasal region to be used for sound generation there
    3. elevating the larynx to increase pressure in the nasal cavity, driving air upward through sound-generating tissues located there. (previously observed through endoscopy)
  * the nostrils of odontocetes have fused to become a single blowhole, unlike mysticetes who have a pair of blowholes
  * below the nostril are two nasal passageways and a complex arrangement of air sacs, nasal plugs and fatty structures.
    $\longrightarrow$ the epicranial (above skull) structures (homologous to facial tissue) comprise muscles controlling nasal plugs, to prevent water from entering, the other structures aid sound generation
    $\hookrightarrow$ they are made up of: bilaterally paired sets of air sacs, fatty structure (melon), and two sets of paired fat bodies (phonic lips $\Longrightarrow$ sonar sources)
  * phonic lips can clap against each other when air passes between them, or it can produce whistles when air passes along their ridges
  * Sperm whales only have one of these pair of lips https://www.youtube.com/watch?v=sW7o5IC2io0
  * Dolphins, having two pairs of lips can produce different clicks simultaneously or whistle and click simultaneously

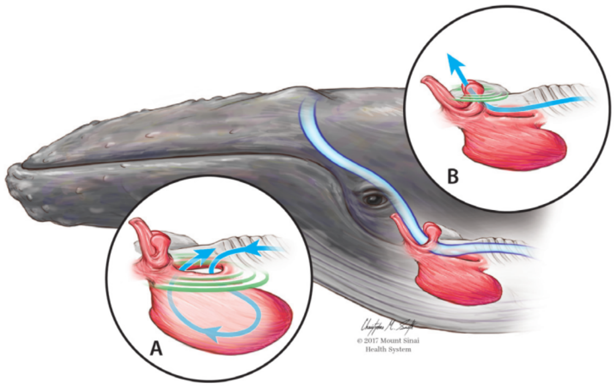


Figure 1: Sound production in Mysticetes. A: Larynx is closed, air is lead into laryngeal sac, airflow diversion causes low frequency vibrations. B: epliglottic cartilage is closed, corniculate cartilage open, causing air to pass through it. Corniculate cartilage can be closed periodically causing pulsed sounds.

    * downstream of the phonic lips, air is guided through three pairs of air sacs, located underneath the blowhole
      $\longrightarrow$ the blowhole is closed during this process
    * air sacs are inflated allowing for longer airflow for sound production (same in mysticetes)
      $\longrightarrow$ like in mysticetes, air is reused several times for sound production
    * high vibration speeds of phonic lips generate ultrasonic clicks, that are transmitted as vibrations to attached fat bodies (dorsal bursae)
      $\longrightarrow$ vibrations are carried through the anterior dorsal bursa to a contiguous larger fat body in the forhead (melon)
    * the melon is a biconvex shaped organ, made up of isovaleric acid (saturated fatty acid), with similar density as seawater (small loss due to impedance change)
      $\longrightarrow$ the isovaleric acid or spermaceti is one of the reasons it was heavily hunted as the huge amounts of the fatty acid was used in cosmetics, lubricant, and expecially candles
    * the shape of the melon can be altered through facial muscles, thus focusing the direction of the sound beam
    * see fig. 2
  - "The divergence into opposite ends of the frequency spectrum highlights how these two taxa, although related, pursued very different niches for their acoustic abilities."

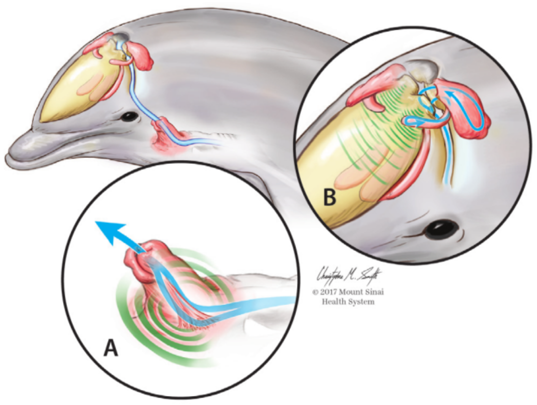


Figure 2: Sound production in Odontocetes. A: larynx evolved in a way to ideally protect respiratory tract from water and can potentially aid in sound production (though never observed). B: highly compex nasal ragion able to produce loud ultrasonic clicks and whistles used for ecolocation.

.4 The devil is in the detail: Quantifying vocal variation in a complex, mulit-levelled, and rapidly avolving display

- Author: Ellen C. Garland and Luke Rendell [5]
- *song types* - different versions of a song, alterations in themes and phrases
- the unidirectionality in song transmission is not well understood, however it is possible to use diffrences in the song to identify different dialects and also poulations at any point in time.
- aiming to quantify similarities between different humpback whale songs, the Levenshtein distance (LD) has outperformed all other metrics, .2.1
  $\longrightarrow$ the LD is a robust metric, preferable to Markov chains, Markov hidden models, or Shannon entropy
- to ensure comparable outcomes between different lengthened strings, the LD is standardised by the length of the longer string in the pair, LD similarity index:

$$\mathrm{LSI}(a,b) = 1 - \frac{\mathrm{LD}(a,b)}{\max(\mathrm{len}(a), \mathrm{len}(b))} \tag{1}$$

- as the LD only compares but doesn't account for the different levels of hierarchy in the humpback song, an additional weighting system is proposed
- all recorded units were compared which reduced the 750 recorded units to 96 unique units
  $\hookrightarrow$ units were subjected to Classification and Regression Tree analysis, **CARF** as well as Random Forest trees
- CART 77% and RF 73% classification overlap with human classifiers
- to cluster data, the average of each variabel (ex. maximum frequency) was calculated and z-transformed

- 11 acoustic features were extracted and Euclidian distance was calculated based on them $\longrightarrow$ features: Duration, f_{min}, f_{max}, f_{start}, f_{end}, $f_{range} = f_{max}/f_{min}$, $f_{trend} = f_{start}/f_{end}$, Bandwidth $f_{max} - f_{min}$, Inflections (Wendpunkte) count($\ddot{f}(t) = 0$), $f(\text{mag}_{max})$, Pulse rate

- the normalized Euclidian distance was calculated:

$$d(x,y) = \frac{\sqrt{\Sigma_i (z(x_i) - z(y_i))^2}}{\max(d)} \tag{2}$$

- d was used as a weighting penalty for substitutions in LD calculations
  $\longrightarrow$ with linear d, very different units were off the scale and similar units very unpenalized, making comparisons unpractical
  $\hookrightarrow$ introduction of exponential normalized cost: $\text{exp\_cost}(x,y) = 1 - e^{-\beta d(x,y)}$, with $\beta = [1, 0.5, 0.25]$

- $\beta = 1$ is close to unweighted (everything $= 1$), $\beta = 0.25$ comes closet to linear weighing

- other options of weighting systems would be penalty matrix based on the output of node weights, Euclidian distances, or Cartesian distances

- as phrase duration has proven to be one of thee most stable components of humpback whale song, insertions and deletions are more heavily penalized than subsitutions

- analysis was practiced on 3 levels:

  A a string represents a sequence of units, assigning them to a phrase

  B a string represents a sequence of units to identify a median unit sequence per phrase/theme, this is the sequence with the highest summed similarity (LSI) to all other members within the group
  $\longrightarrow$ this provides a representative string for the set that can be used to compare among sets

  C a string represents a sequence of phrases (song) assigning them to a song type

  $\longrightarrow$ C is only based on unweighted analyses

- to attain sampling distributions, the data is bootstrapped with multi-scale bootstrap resampling 1000 times
  $\longrightarrow$ AU significance: p $>$ 95%
  $\longrightarrow$ BP significance: p $>$ 70%

- when first analyzed with unweighted LD, similarity between 4 chosen representative themes was single digit percentages, similarity within a theme (comparing median phrase with other phrases) was between 44 % and 73 %, and increased to 53 % to 79 % once $\beta = 1$ weighting was applied.

- because phrase length was chosen as important and substitutions are less penalized by lower β weighting values, a threshold exists, where phrase length is considered more important than actual similarity, as substitutions are less penalized than deletions and

insertions.
$\longrightarrow$ insertions and deletions maintained penalty scores of 1, and substitution penalty scores needed to exceed 0.6 to manage this phenomenon

- the study shows that weighting schemes offer a methodology to prefer one kind of similarity over others, however, all weighting schemes come with consequences and researchers should be cautious of the way research questions are framed and if the weighting scheme really favors it

- many different methods can be used to favor certain substitutions over others, i.e. every euclidian distance over 0.5 leads to a substitution penalty of 1

- ultimately weighting of LD analyses will aid in clarifying fine-scale differences between humpback songs, potentially helping to find dialects and theme progressions.

- the usage of median strings can be used to find the most representative phrase for each theme (intra-individual) and these can then be compared to other individuals to understand differences in cultural diversity within a population
$\longrightarrow$ ANOVA (.3.4) could be used to further explore diversity

.5 Deep Machine Learning Techniques for the Detection and Classification of Sperm Whale Bioacoustics

- authors: Peter Bermant, Michael Bronstein [2]
- cateceans pose an important taxa to study langauge, as the difference in environment can be revealing from a comparative perspective in regards to aquatic communication
- sperm whales mainly communicate through codas - 3-40 broadband clicks that are in general transmitted in socializing groups
- aside from communicative codas, sperm a variety of click-baed sounds exist:
  1. usual clicks for echolocation and foraging
  2. foraging for short-range prey detection and feeding
  3. reverberating slow clicks (clangs) produced only by mature males
- NN applied in 4 ways:
  1. detection of echolocation clicks using CNN
  2. classification of codas into categorical types using a Long Short-Term Memory (LSTM) RNN-based approach
  3. recognition of coval clan coda dialects with LSTM RNNs
  4. identification of individual whales based on coda production with LSTM RNNs
- LSTM RNNs are effective for this because their architectures aebanle the network to preserve the non-independent sequential order of input data
- 99.5% train accuracy with 650 spectrogram images (325 click and no-click)
  $\longrightarrow$ echolocation classification task works very well with the CNN approach
- For LSTM RNN classification of different cods, a network is pretrained on a proxy-task (a task similar to the final task, but requiering less data), this is called **self-supervised learning**
  $\longrightarrow$ in this case learning the time interval between subsequent clicks
- utiziling transfer learning procedures, the network is then trained on coda type, vocal clan and individual whale identity classification
- first perform PCA for dimensionality reduction from 256 to 20, then t-SNE to reduce from 20 to 3
- results from both echolocation and coda classification were very postive

.6 Acoustic Detection of Humpback Whales Using a Convolutional Neural Network

- Ann N. Allen [1]
- second author Matt Harvey, Software Engineer, Google AI Perception
- some have tried power law approaches, to review power in frequency bands and detect based on that, the abundance of other noises in the ocean however, makes this approach less practical
- 187000h of recordings, with the technique of active learning, to increase the training set size
- recordings orifinate from 13 sites in the North Pacific, using bottom-mounted High Frequency Acoustic Recording Pachages (HARPs), equipped with omni-directional hydrophones, with a flat ($\pm$2dB) hydrophone sensitivity from 10 Hz to 100 kHz (with -200dB V/μPa)
- monitoring duration varied from a month, to 13 years
- recording schedules ranging from continuous to 5 min of every 40 min
- recording files consisted of 75 s blocks of audio (*segments*)
- deployment depths $\in [111, 1266]m$, avg = 730m
- raw data sampled at 200 kHz, low-pass filtered at 10Hz, resampled at 10 kHz
  $\longrightarrow$ effective bandwidth: [10, 5000] Hz (13 TB of data)
- binary classification task, based on a image classification CNN model
- model results were compared to energy-based detectors as a baseline
- per channel energy normalization (PCEN) was applied to the spectrograms
- spectrogram creation:
  - STFT with Hann window of length 1024 samples (100 ms)
  - output size: 128 x 96 (txf)
  - strides of 10, 30, 50 ms
  - along f axis, trials binned using a triangular mel filterbank over squared FFT magnitudes
  - three different amplitude compression functions were used: log, root compression and PCEN
    $\longrightarrow$ PCEN: normalizing by componentwise dividing the STFT with temporally smoothed version of the same STFT and then apply root compression
    $\hookrightarrow$ smoothing is done for each channel independently, smoothing constant: s = 0.04

- the model was based on the architecture by Hershey et al. (2017) found best at detecting audio event classes in YouTube videos
  $\longrightarrow$ ResNet-50 CNN with a lowered stride from 2 to 1 of the initial 7x7 conv layer (images are smaller than on the original model)
- data augmentation happened on-the-fly, with time shift by ±16 bins relative to non-overlapping grid, done dynamically so perturbations were varied accross epochs
- **active learning**
  - iterative strategy that improves an existing classifier by using it to selec the most valuable examples for additional human annotation and subsequent retraining
  - model predictions were reevaluated to speed up anotation process and at the same time throw out negatives that were close calls (boat noise, minke whale calls)
    $\longrightarrow$ resulting audited data: 291.8 h of reviewed recordings for train, test, and validation set
- baseline performances were set by a reimplemented generalized power-law detection algorithm for humpbacks (GPL) and a simple band-limited energy detector
- results: average precision = 0.97, AUC-ROC = 0.992

.7 All units are equal in humpback whale songs, but some are more equal than others

- by Eduardo Mercado III [7]
- controvertial author
- quotes from George Orwells animal farm in title
- claims that contrary to most other researchers, units are not constant throughout the years
- because units are not constant, and they vary in the distance over which they are audible, the theory of the humpback whales song being solely for sexual selection and display of intellectual capacity does not hold
- Mercado argues that the frequencies in units are precisely chosen and altered to generate sonar that can give information about changing environments and prey
-

Examples

Datum: 12.05.22

.1 Statistics

.1.1 Bootstrapping Dendrograms

Paper: Bootstrap confidence levels for phylogenetic trees https://www.pnas.org/doi/10.1073/pnas.93.23.13429?url_ver=Z39.88-2003&rfr_id=ori:rid:crossref.org&rfr_dat=cr_pub%20%200pubmed

- procedure:
  1. collect data and arrange it in **x** so that
     - rows correspond to different species
     - columns correspond to sequence
       $\longrightarrow$ humpback units, phrases, letters of a word ...
  2. create distance matrix **D**
     - distance matrix has dimensions (len(x) x len(x))
     - quantify distance of each row with each other row using a distance metric like LD .2.1
  3. create phylogenetic tree
     - use connection algorithm to cluster elements, one clustering per generation, reducing the size of the distance matrix by one/generation
- this procedure is then repeated with bootstrapped datasets:
  - a bootstrapped datamatrix **x*** is created using the same rows but the columns are randomly selected from **x**
- this way trees are built in the same manner several hundred times
- the proportions of bootstrap trees resulting in the same topology with the original are calculated
  $\longrightarrow$ these proportions are bootstrap confidence values or **bootstrap probability (BP)**
  $\hookrightarrow$ so the numbers on merging branches indicate the percentage in how many cases these were merged together at that stage
- the underlying assumption of course is that columns, so sequence elements are independent of another, questionable if this holds for humpback whale units
- the underlying idea behind bootstrapping is to use the observed data **x** and the bootstrappend data **x***, attain estimates $\hat{\theta}^*$ and $\hat{\theta}$ and calculate $\hat{\theta}^* - \hat{\theta}$ to then get a probability and a standard deviation, telling us that knowing $\hat{\theta}^* - \hat{\theta}$ we can estimate $\hat{\theta} - \theta$ with the same margin of error (= std from $\hat{\theta}^* - \hat{\theta}$)
  $\longrightarrow$ $\hat{\theta}$ in this case is the observed phenomenon and θ is the phenomenon

.1.2 ANOVA

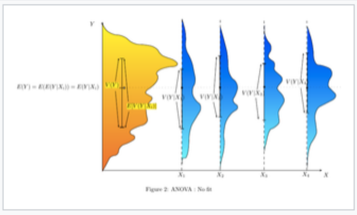
- from wikipedia: goal: find out what factors play into the weight of a dog aat a dog show
- a dog show has numerous dogs on display
- a histogram of all dog weights is made
- first approach would be to divide the dogs based on characteristics hoping, that the decomposition into its aspects will yield correlations
- each group should have a low variance of dog weights and the mean of the groups should be distinct, so that the make up for different parts of the overall histogram
  $\longrightarrow$ analogous to a fourier transformation - dissecting a group into its fundamental components
- in a first attempt the group was split by age and length of hair, then by pet and working breed, and finally by breed
  $\longrightarrow$ the first two options both yield 4 resulting groups (2^2) whereas the last one yields as many groups as there are breeds
- the last option is the most feasable in this case, because the last one had the most distinct means and no overlap, see fig. 3
  $\longrightarrow$ the group histograms of the last split are **linear independent**

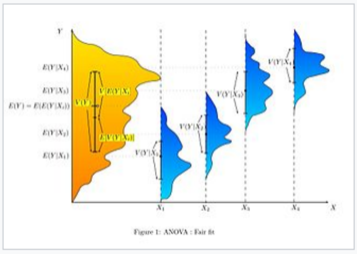
.2 Machine Learning

.2.1 Perceptron training

training a perceptron with scikit learn:

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
iris = load_iris()
X = iris.data[:, (2, 3)] # petal length, petal width
y = (iris.target == 0).astype(np.int) # Iris Setosa?
per_clf = Perceptron()
per_clf.fit(X, y)
y_pred = per_clf.predict([[2, 0.5]])
```


No fit: Young vs old, and short-haired vs long-haired


Fair fit: Pet vs Working breed and less athletic vs more athletic

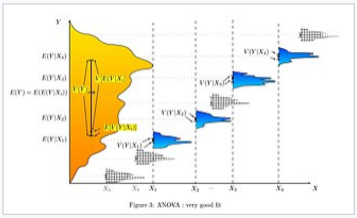

Very good fit: Weight by breed

Figure 3: ANOVA example: dog weights. Different attempts to find the factors underlying dog weights. The last example is the best fit, because all histograms distinct and don't overlap.

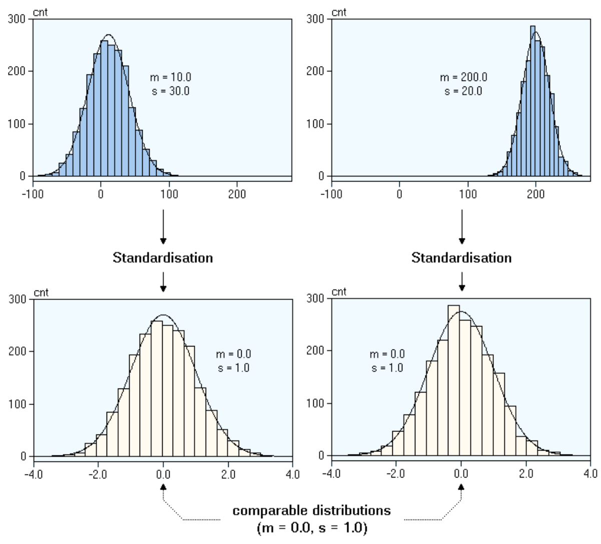


Figure 4: Example of z-transformation. unique distribution of data point relative to one another is preserved, but mean and std are standardized.

Statistics

Datum: 11.05.22

.1 Transformations

.1.1 Z-Transformation (also standardization or auto-scaling)

- a **z-score** (standard score) calculates the distance of a given point from the mean in multiples of the std
- z-Scores become comparable by measuring in multiples of std of sample
  $\longrightarrow$ **mean of z-transformed data is always 0**
  $\longrightarrow$ **std of z-transformed data is always 1** (if original distribution is normal)
- ex. see fig. 4

.1.2 Unweighted Pair Grouped Method with Arithmetic Mean - UPGMA

- the clusters with the smallest distance between them are clustered and the distance matrix is recalculated
  $\longrightarrow$ the distance between two clusters is the average pair wise distance of all objects within both clusters

$$d_{(A\cup B),X} = \frac{|A|d_{A,X} + |B|d_{B,X}}{|A| + |B|} \tag{3}$$

  $\longrightarrow$ where $d_{(A\cup B),X}$ is the distance between the new cluster $A \cup B$ and X, and $|A|$ denotes the size of A

.1.3 Bootstrapping

- bootstrapping is a statistical method that resamples a single data set to create many simulated samples
- bootstrapped samples have the same size as the original but will have some data points more than once and others not at all
  $\longrightarrow$ bootstrapping assumes that the existing data set is a accurate representation of the actual population
- using computers datasets are bootstrapped thousands of times, when plotting the resulting distribution of mean values of the sampled data a normal distribution will occur, (almost) regardless of the distribution in the data set
  $\longrightarrow$ this is called a sampling distribution of means (if calculated analytically its a probability distribution)
  $\hookrightarrow$ if the top and bottom 2.5 % are cut off, we are left with a 95 % confidence interval, meaning that the mean of our original sample dataset will lie within that interval with a probability of 95 %

.1.4 Phylogenetic tree

- phylogenetic trees, or evolutionary trees are branching diagrams showing the evolutionary relationships among various biological species based on similarities and differences in physical or genetic characteristics
- Dendrograms
  - dendrograms are representations of trees used in **hierarchical clustering**, computational biology, and phylogenetics
    $\longrightarrow$ hierarchical clustering can be **Agglomerative** - bottom-up; or **Divisive** - top-down
    $\hookrightarrow$ bootstrapping is a Agglomerative approach
- **bootstrapping dendrograms**
  - data points are merged, one per generation, reducing the data matrix by one dimension per generation
    $\longrightarrow$ mergers are evaluated using tests:
    * bootstrap probability (BP) value: computed by normal bootstrap resampling, see .1.1 (much simpler than AU)
      $\longrightarrow$ BP is the frequency that a value appears in the bootstrap replicates
    * approximately unbiased (AU) p-value: computed by multi scale bootstrap resampling proving a superior approximation to unbiased p-values
      $\longrightarrow$ text
  - when picturing the clustering process, there exists a boundary that has to be overcome for the merger to happen. So the clustering process is a distance measurement in a 2 dimensional space

  - while BP is equivalent to assuming a flat Bayesian prior for μ (an clustering candidate is equally possible to lie anywhere within the space), AU takes into account the shape of the boundary between both states and is thus a more precise measure
  - while the boundary between the two states can be a straight line, it can also be a circle due to radial conditions (for the straight line case BP and AU are identical, for the circular boundaries they are not)
  - AU requires higher bootstrapping rates than BP
  - BP is said to have a downward bias, though this is disputed

- CCC is a metric to quantify the overall associations within the data by the dendrogram see .2.2

.2 Metrics

.2.1 Levenshtein Distance

- LD is a string metric
  $\longrightarrow$ measures distance between two text strings for approximate string matching or comparison (used in fuzzy string searching), must satisfy the triangle inequality (two sides of a triangle must be greater or equal to third side)

- LD between two words is the minimum number of single-character edits required to change one word into the other

- definition:

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0 \\ |b| & \text{if } |b| = 0 \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \text{if } |a| = |b| \\ 1 + \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise} \end{cases} \tag{4}$$

  $\longrightarrow$ tail is a string lacking the first element, $|*|$ denote length of string

.2.2 Cophenetic Correlation Coefficient

- CCC is a measure how faithfully a dendrogram (tree diagram often used in taxonomy) preserves the pairwise distances between the unmodeled data points
  $\longrightarrow$ CCC > 0.8 is considered high and thus a good representation of associations within the data

.3 Tools

.3.1 statistical significance

- a result has statistical significance when it is very unlikely to have occurred given the null hypothesis (H_0)

- the significance level, α, is the probability of a study to reject H_0 if given $H_0 ==$ True
$\longrightarrow$ α is a measure of the strength of the evidence that must be present in your sample before you will reject H_0 and call the effect statistically significant
$\longrightarrow$ α is determined before the experiment
    - p-value in null-hypothesis significance testing is the probability of obtaining test results at least as extreme as the result observed, given that $H_0 ==$ True
    - small p-value $\Longrightarrow$ extreme observed outcome would be very unlikely under H_0
    $\longrightarrow$ lower p-values indicate greater evidence against H_0
    $\longrightarrow$ p-values help in determining which hypothesis the data supports
- if $p \leq \alpha$ you can reject H_0 and conclude the event is statistically significant $\Longrightarrow$ the evidence in your sample is strong enough to be able to reject H_0 at population level
$\longrightarrow$ conventionally $\alpha = 5\%$ or lower, depending on the field of study

.3.2 z-test

- a z-test determines whether two population means are different when the variances are known an the sample size is large (>30)
$\longrightarrow$ normal distribution and knowledge of std are required

$$Z = \frac{\bar{X} - \mu_0}{\sigma} \tag{5}$$

$\longrightarrow$ $\bar{X}$ is mean of sampled data, μ_0 is the expected value, σ is the std

- after the z-score is calculated it gets evaluated in the hypothesis test, where it's compared to the value chosen in the null- and alternative hypotheses
-

.3.3 Two-Tailed Test

- used in null-hypothesis testing and testing for statistical significance
- used when a hypothesis states that a sample is either significantly greater or less than the mean of a population
- depending on how critical consequences are, conventions can either require the number of data points that must exist within the acceptance range to be 5% (1.96) or 0.001%
$\longrightarrow$ the acceptance range is also referred to as alpha

.3.4 Analysis of Variance - ANOVA

- ANOVA is an analysis tool that splits an observed accumulated variability found inside a data set into two parts
    - systematic factors: have a statistical influence on the data set
    - random factors: don't have an influence on the data set

- ANOVA can be used to determine the influence that independent variables have on the dependent variables in a regression study
- ANOVA is the extension of the t- and z-tests

$$F = \frac{\text{MST}}{\text{MSE}} \tag{6}$$

$\longrightarrow$ F = ANOVA coefficient, MST = Mean sum of squares due to changes, MSE = Mean sum of squares due to error

- ANOVA is used to test the difference between two or more means

Machine Learning

Datum: 11.05.22

.1 Algorithms

.1.1 Decision Tree

- hierarchical structure with binary division based on single feature and threshold

.1.2 Random Forest

- train a group of Decision Tree classifiers, each on a different random subset of the training set
- Predictions are made by getting all predictions of the individual trees and predicting the class with the most votes
  $\longrightarrow$ even though its simple it's one of Machine Learnings most powerful algorithms
  $\longrightarrow$ Ensemble Learning algorithms like random forests are good to use near the end of a project, once you have already built a few good predictors to combine them into a very good predictor
  $\hookrightarrow$ winning ML solutions often involve several Ensemble methods

.1.3 Classification and Regression Tree - CART

- set is split into two parts based on one feature and one corresponding threshold
  $\longrightarrow$ feature and threshold are chosen to produce the purest subsets (weighted by their size)
- the algorithm then minimizes the following cost function:

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \tag{7}$$

  $\longrightarrow$ with $G_{left/right}$ - measure of impurity of the left/right subset
  $\longrightarrow$ and $m_{left/right}$ - number of instances in left/right subset

- algorithm works recursively, splitting into smaller and smaller sets until no minimization is reached or maximum depth is achieved
  $\hookrightarrow$ CART is a *greedy algorithm* - greedily searches for optimum split at each level, regardless of consequential impurity at lower levels; *greey algorithms* produce reasonably good results, not optimal ones. Finding optimal trees is a NP-Complete Problem $\Longrightarrow$ requires $O(\exp(m))$ time, so reasonably good must be settled for

.1.4 Principal Component Analysis - PCA

- PCA is the most popular dimensionality reduction algorithm

- PCA requires a centered dataset around the origin (Scikit-Learn does this automatically, others dont)
- a principal component is the i_{th} unit vector yielding the largest amount of variance (they are orthogonal)
- PCA uses the principal components to create a plane closest to the data points

.1.5 t-Distributed Stochastic Neighbor Embedding (t-SNE)

- unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data
- contrary to PCA, t-SNE preserves only small-pariwise distances or local similarizies whereas PCA only preserves large pairwise distances
- Swiss Role example, works well with t-SNE not so much with PCA
- t-SNE algorithm calculates a similarity measure between paris of instances in high and low dimensional space and tries to optimize both similarity measures using a cost fucntion.
  1. measure similarities between points in high dimensional space
     $\longrightarrow$ perform gaussian distribution around all the points, calculate density for every point, renormalize $\Longrightarrow$ get probabilities for each point, which are proportional to the similarity meaning points with similar values, they will be similar
  2. repeat 1. but use Student t-distribution with only 1 degree of freedom (Cauchy distribution) instead of gaussian distribution, see fig. 5. This gives us a set of probabilities in the low dimensional space
  3. we want the two sets of probabilities to be similar, so: measure the difference between the distributions using **Kullback-Liebler divergence** (asymmetrical approach, for efficient comparisons) and use gradient descent to minimize the KL cost function.
- an important distinction from PCA is that t-SNE alters the inputs, so its only for exploration

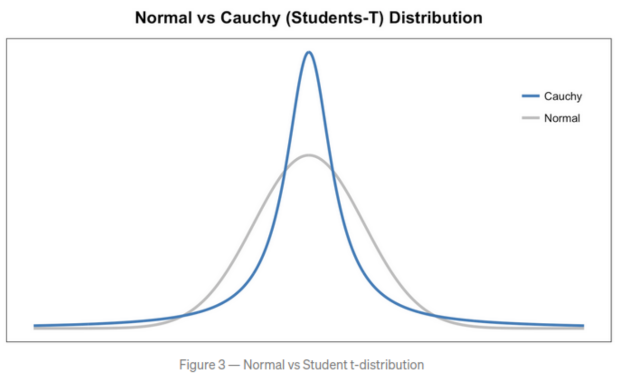


Figure 5: Cauchy vs Gaussian Distribution. The Cauchy distribution is used in t-SNE to favor small distance similarities.

.2 Tools

.2.1 Spectrogram and STFT windowing

- both the raw data (time-resolved) and the spectrogram data (time- and frequency resolved) need to be segmented in windows to be analyzed
- **STFT**
  - STFT:
  $$X(m, \omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \tag{8}$$
  $\Longrightarrow$ where w[...] is the window function, most commonly Hann window, x = signal, and ω is frequency
  - windowing in STFTs is critical as frequency resolution depends on window length:
  $$\Delta f = \frac{fs}{n_{samples}} \tag{9}$$
  - bigger windows lead to better frequency resolution but worse time resolution
  - a hop length, which defaults to the window length, can be specified to have overlapping windows:
  $$n_{overlap} = n_{window} - n_{hop} \tag{10}$$
- **Spectrogram Windowing**
  - for CNN classifiers, working with spectrograms, the algorithm needs a standardized window length
  - the windows which are classified are **context windows** $\Longrightarrow$ **context window length** is the number of samples used in one context window
  - each context window will lead to one classification and thus yield one score per class $\longrightarrow$ for detectors only one score
  - hop size is critical for spectrogram classifiers, because units could be cut up by windowing, thus a smaller hop size increases the likelihood of the unit being fully preserved in a window

  - the context window hop size is thus the number of samples by which a context window is shifted
  - the context window hop size has to be a multiple of the STFT hop size, as the spectrogram can only be shifted by multiples of the fourier transformed windows
  - the number of scores resulting from a classification can be calculated:

$$n_{scores} = \text{floor}(\frac{n_{signal} - n_{context\,window}}{n_{context\,window\,hop}} + 1) \tag{11}$$

    $\Longrightarrow$ where $n_{context\,window\,hop} = \text{floor}(\frac{n_{context\,window\,hop}}{n_{stft\,hop}}) * n_{stft\,hop}$
    $\longrightarrow$ meaning that the context window hop is a multiple of the stft hop
    $\Longrightarrow$ the $-n_{context\,window}$ is necessary, as the first context window starts with the context window length and every subsequent context window is shifted by the context window hop

.2.2 Singular Value Decomposition

- SVD can decompose a dataset matrix $\mathbf{X}$ into the matrix multiplication of 3 matrices: $U\,\Sigma\,V^T$, with $V = \begin{pmatrix} c_1 & c_2 & \dots & c_n \end{pmatrix}$, where c_i are the principal components
- coding example:

```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]
```

.2.3 Self-organizing (feature) map - SOM

- unsupervised machine learning technique used to produce a 2D representation of higher order dimensional data while preserving the topological structure.
- is trained with competitive learning
  $\longrightarrow$ form of unsupervised learning in ANN's where nodes compete for the right to respond to a subset of the input data

.2.4 Cross Entropy

- different functions are used to minimize the cost function
- cross-entropy is frequently used to measure how good the match is between a set of estimated class probabilities and the target classes
- cross entropy cost function:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)}) \tag{12}$$

  $\Longrightarrow$ $y_k^{(i)}$ is the probability that the ith instance belongs to class k - either 1 or 0 (belongs or doesn't belong)

-

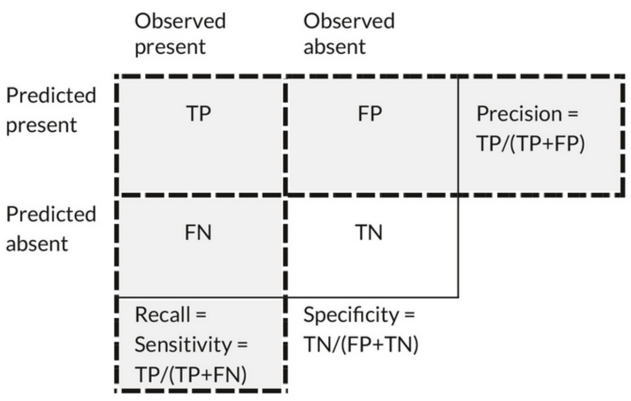


Figure 6: Confusion matrix showing performance measures for ROC and PR.

.2.5 Data Augmentation

- Data Augmentation can be used to generate more training instances from the available data
- augmentations should be subtle enough so that a human can't tell the difference between the original and the augmented data
- common means of augmentation for images are cropping, rotating, shifting, changing lighting conditions, and any combination of these
- it's important, that only augmentations are used that can be learned, so an addition of white noise does not count as augmentation

.3 Precision Metrics in Binary Classification Tasks

- if class predictions are based on probabilities instead of direct class prediction, thresholds to guide choices can be learned
- this allows to set different weights fo **False Positives** and **False Negatives** depending on the consequences of such an error
- performance measures shown in fig. 6
- in general:
  - ROC curves should be used when there are roughly equal numbers of observations for each class
  - PR cuves should be used when there is a moderate to large class imbalance

.3.1 Receiver Operating Characteristic Curve (ROC)

- plot of false positives rate (x-axis) versus the true positive rate (**Sensitivity**) (y-axis) for different threshold values between 0 and 1
  $\longrightarrow$ false alarm vs. hit rate

- True pos. rate = TP / (TP + FN)
  $\longrightarrow$ describes how good the model is at predicting the positive class when the actual outcome is positive
  $\hookrightarrow$ also referred to as **Sensitivity**

- False pos. rate = FP / (FP + TN)
  $\longrightarrow$ describes how often a positive is predicted when the actual outcome is negative
  $\hookrightarrow$ also called **inverted Specificity**

- Specificity = TN / (TN + FP)
  $\longrightarrow$ describes how often a negative was predicted when the actual outcome is negative

- False pos. rate = 1 - Specificity

- Area Under the Curve (AUC) of ROC curve can be used to summarize a models skill
  $\longrightarrow$ **skill** of a model is the characteristic of assigning a higher probability to a randomly chosen real positive, than a negative on average
  $\hookrightarrow$ skilful models are represented by curves that bow up to the top left of the plot

- a model with no skill is represented at the center point (0.5, 0.5)

- a model with no skill at every threshold is represented by a diagonal line from the bottom left of the plot to the top right and has a AUC of 0.5

- a model with perfect skill is represented at point (0, 1)

- a model with perfect skill at every threshold is represented by a vertical line from the bottom left to the top left and then a horizontal line to the top right (AUC = 1)

.3.2 Precision-Recall Curve (PR)

- plot of the **Precision** (y-axis) and **Recall** (x-axis) for different thresholds

- Precision = TP / (TP + FN)
  $\longrightarrow$ describes how good a model is at predicting the positive class
  $\hookrightarrow$ also referred to as the positive predictive value

- Recall = TP / (TP + FN) (= Sensitivity)

- PR curves are useful in cases where there are many examples of no event and only a few examples of an event

- PR curves don't care about TN, because they are only interested in positives (in the minority class)

- a model with no skill and varying threshold is represented by a horizontal line, the hight depends on the class balance of the data

- a model with perfect skill is represented at the point (1, 1)

- a skilful model is represented by a curve that bows towards (1, 1) above the flat line of no skill

- PR curves can be evaluated with the **F-Measure**: harmonic mean of precision and recall rates, and AUC
- F-Measure summarizes skill for a specific threshold value
- AUC summarizes the skill of amodel across thresholds

.4 Deep Neural Networks

.4.1 Basic Structure

- from [6] (Chapter 10)
- Deep Neural Networks are ANN's with a of deep stack hidden layers
  - ANN (artificial neural networks) first comprised of Perceptron (invented in 1957)
    $\longrightarrow$ based on an artificial neuron called threshold logic unit (TLU):
    * inputs with weights are computed into a weighted sum, then step function is applied outputting: $h_w(x) = \text{step}(z)$, while $z = x^T w$, w = weights, x = inputs
    * common step functions are sign function (-1 if z<0, 0 if z = 0, 1 if z > 0) or heavyside (0 if z<0, 1 otherwise)
    * TLU can be used for simple linear binary classification (just like Logistic Regression classifier or linear SVM)
  - a Perceptron is composed of a single layer of TLUs, all TLUs are connected to all inputs
    $\longrightarrow$ if all neurons in a layer are connected to eveery neuron in the previous layer (i.e., the inputs) it's called a **dense layer**.
    $\hookrightarrow$ for consistence, usually input neurons are drawn, just outputting whatever input they are fed
    $\longrightarrow$ a bias feature is generally added ($x_0 = 1$) it's typically represented using a special type of neural called bias neuron, outputting 1 all the time - so to say the first input
  - a 2 input - 3 output perceptron is shown in 7
  - calculating the outputs:
    $$h_{W,b} = \Phi(XW + b) \tag{13}$$
    * W - weight matrix: all weights except for bias
    * b - bias vector: all connection weights between bias neuron and artificial neurons
    * X - input matrix
    * Φ - activation function
  - "Cells that fire together, wire together." - Hebbian learning
    $\longrightarrow$ connection weight between two neurons is increased when output is same
  - perceptrons are trained so that after a training instance is run and a prediction is made, for every output neuron producing a incorrect prediction, the connection weights from the inputs that would have yielded the correct predictions are strengthened
    $$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \tag{14}$$

        * $w_{i,j}$ - connection weight between ith neuron and jth output neuron
        * x_i - ith input value
        * $\hat{y}_j$ - output of jth output neuron
        * y_j - target output of jth ourput neuron
        * η - learning rate
    - all decision boundaries from output neurons are linear in perceptrons, making them incapable of learning complex patterns (like Logistic Regression classifiers)
      $\longrightarrow$ if training instances are linearly separable, the algorithm would converge to a solution
    - Perceptron learning algorithm is equivalent to **Stochastic Gradient Descent** with no penalty (no regularization) and constant learning rate
      $\hookrightarrow$ SGD - picks random training points to not use every training point, so it's good for large training sets (it's important that the training instances are truly chosen at random! - otherwise only one class gets learned before the next class is introduced)
      $\longrightarrow$ perceptrons give out hart class predictions, they don't provide probabilities for classes like Logistic Regression classifiers
    - to be able to solve a XOR, which is not possible with standard perceptron logic, a MLP (multi-layer perceptron) needs to be used, adding an extra layer - a hidden layer
    - lower layers are close to input, upper layers are close to output
      $\longrightarrow$ every layer except for the output layer has a bias neuron and is fully connected to the next layer

- to train MLP's backpropagation was instroduced
  $\longrightarrow$ for every training instance the networks is run through forwards and backwards (happens in parallel)
  $\hookrightarrow$ one iteration of the entire training set is an **epoch**

- on the backwards pass, the network calculates the loss-function comparing the desired output values with the actual output values, returning a measure of error
  $\longrightarrow$ it then works its way back to the input computing how much each output contributed to the error
  $\hookrightarrow$ once the errors are all calculated, a Gradient Descent is performed for every weight in the network

- backpropagation relies on unsymmetric weights, that's why randomly initializing the weights is crucial

- for Gradient Descent to properly work, a continuous function was needed, introducting the **logistic function**: $\sigma(z) = 1/(1 + e^{-z})$
  $\longrightarrow$ other famous activation functions are:

    - **hyperbolic tangent function**: $\tanh(z) = 2\sigma(2z) - 1$
      $\longrightarrow$ goes from -1 to 1, unlike σ (from 0 to 1), can speed up convergence as values center around 0

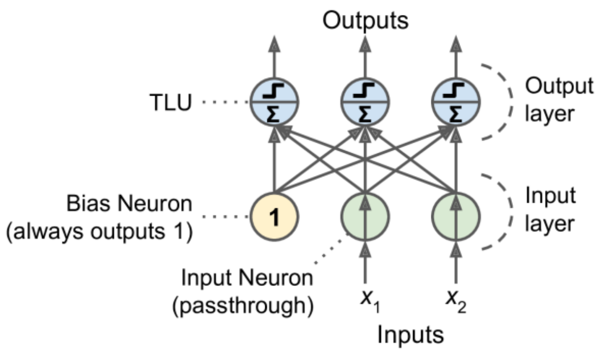


Figure 7: Perceptron structure. structure of a simple perceptron with 2 inputs and 3 outputs.

- **Rectified Linear Unit functio**n: $\text{ReLU}(z) = \max(0, z)$
  $\longrightarrow$ not differentiable at z=0, very fast to compute, doesn't have a max wich can be advantageous

- why are activation functions needed in the first place?
  $\longrightarrow$ if no activation functions are used, all weights are just chained, leading to only linear functions ($f(x) = 2x + 3$ and $g(x) = 3x - 3 \Longrightarrow f(g(x)) = 2(3x - 3) + 3$), so activation functions introduce the non-linearity enabling us to solve non-linear boundaries between classes

- depending on the output of the NN, it can be good to have a activation function in the output layer, limiting values

- error functions to use:
  - typically: mean squared error
  - in case of many outliers: mean absolute error
  - alternatively: Huber loss, which combines both: quadratic when error is smaller than threshold (mostly 1) and linear when larger
    $\longrightarrow$ less sensitive to outliers whilst more accurate and faster in converging than mean absolute error

- standard MLP classification setup: [6] page 291

- MLP vs CNN vs RNN
  - MLP: vectors as inputs
  - CNN: tensor as input, is able to understand spatial resolution, thus better for complex images
  - RNN: designed to work for problems related to sequence like sequence of words in sentences, or sounds in speech recognition

.4.2 Long Short-Term Memory network (LSTM)

- advanced RNN (sequential network), that allows information to persist
  $\longrightarrow$ can learn in what way sequences are relevant
  $\longrightarrow$ can handle vanishing gradient problem
- RNNs can remember short sequences, but with longer sequences, they can't because of vanishing gradients, LSTMs are designed to also remember long sequences

.4.3 Intuition for NN structure

- the book [6] led me to the website `www.playground.tensorflow.org`, which is ideal to play around and get a good intuition for neural network structures
- when choosing the spiral data, the only activation function that works well is tanh, simply because all other functions work on a linear basis
- all input properties are required to sufficiently approximate the complex shape of a spiral
- at least 3 hidden layers, rather 4 are needed to sufficiently approximate the spiral
- more neurons, which are the analogon to feature maps, take more computational power but are able to approach the spiral far faster than having only 2 or 3 neurons per hidden layer
- a set up of 4 hidden layers and 6 neurons each yields a very good loss value after less than 200 epochs
- the tool shows, that similar results can be reached with different network structures taking different amount of computational time

.4.4 Challenges of Training Deep Neural Networks

- **vanishing gradients** - gradients often get smaller and smallers as the algorithm progresses down to the lower layers, resulting in the Gradient Descent update leaving the lower layer connection weights almost unchanged
- **exploding gradients** - in RNN (recurrent neural networks) this problem can arise causing diverging results
- the main driver is the logistic activation function, which saturates for very high values (also negative) and results in very small gradients
  $\longrightarrow$ these then cascade down and there is little gradient left to change low layer weights
- randomly initializing the weights in the beginning can fight this issue
- choice of activation function also plays a bis role for the vanishing gradients - ReLU performs better than logistic activation function, however the gradient for negative values is 0, thus if in a cascade neurons are connected that all have negative values, the wont learn anymore

- scaled exponential linear unit (SELU) is a activation function that approaches -1 for negative values and α for positive values, wich α being a hyperparameter
  $\longrightarrow$ though this takes longer, it avoids both the dead neuron and the vanishing gradients problem making it a very good activation function for many problems
  $\longrightarrow$ when used, the initialization of the weights has to be LeCun normalization: kernel_initializer="lecun_normal"
  $\longrightarrow$ network needs to be sequential, otherwise self-normalization is not guaranteed and the SELU might be worse than others

- in general SELU > ELU (similar to SELU but z(0) = 0) > leaky ReLU ($\alpha = 0.3$ is keras' standard, but it can be changed) > ReLU > tanh > logistic
  $\longrightarrow$ if SELU not possible, then ELU

```
# for leaky ReLU:
leaky_relu = keras.layers.LeakyReLU(alpha = 0.2)
layer = keras.layers.Dense(10, activation = leaky_relu,
                            kernel_initializer = "he_normal")

# for SELU:
layer = keras.layers.Dense(10, activation="selu",
                            kernel_initializer="lecun_normal")
```

.4.5 Optimizations

- **Batch Normalization** - used to zero-center and normalize each input just so that they can be shifted and scaled thereafter, but the shifting and scaling is learnable
  $\longrightarrow$ BN is so powerful, that it vastly reduces number of epochs, acts as a regularization (fighting overfitting) and allows the usage of saturating activation functions (tanh, even logistic)
  $\longrightarrow$ each epoch will take longer, but convergence will be reached far sooner than without BN

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(300, activation="elu",
                       kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(100, activation="elu",
                       kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(10, activation="softmax")
    ])
```

- the batch normlalization is computed by calculating the mean and standard deviation of every input, or every batch, there are several parameters than get learned in the BN, due to the rescaling and shifting
  $\longrightarrow$ additionally there are hyperparameters, momentum (used when updating moving averages; usually close to 1) and axis (dictates which axis should be normalized; defaults

to -1), which get tweaked on a global level
$\hookrightarrow$ axis can also be list, if pixels should be treated independently: axis = [1, 2]

- **Optimizers** - optimizers help speed up the learning by realizing if they are on a plateau or in a valley and accelerate learning rate or other hyperparameters to speed up the search for the global minimum

- most common: adaptive moment estimation (ADAM) combines benefits of momenum optimization and RMSProp
$\longrightarrow$ in general Adam > RMSProp > Nesterov Accelerated Gradient > Gradient Descent
$\hookrightarrow$ if model performs bad try Nesterov Accelerated Gradient instead of Adam, the data might not like adaptive momentum optimizers, but in general they are the best (currently)

- it might be advantageous to use a learning rate scheduler, to make use of large learning rates at first and then reduce it to slow down

.5 Convolutional Neural Networks (CNN)

.5.1 History and Concept

- summary of [6] chapter 14

- CNNs were invented following new findings about the human visual perctive system, showing that different neurons wihtin an area of the receptive field recognize different patters - i.e., some horizontal lines, some vertical lines, some round shapes ...

- using conventional MLPs for complex image recognition leads to a huge number of weights very fast, so CNN was introduced using alternative concepts to avoid millions of weights

- led to invention of the neocognitron, which is now called CNN

- CNNs use fully connected layers (dense layers) and sigmoid activation functions, additionally they use convolutional layers and pooling layers

  - **convolutional layer**: neurons are not connected to every pixel (input) neuron, but rather only to pixels in the receptive field
  $\longrightarrow$ this allows to focus on low-level features on the first hidden layer, then assemble those together to get higher-level features etc, see 8
  - layers are now 2d grids, rather than flat 1d vectors
  - usually zeros are padded to the smaller layers, to maintain the same size
  - the overlap step size between the layers is called a *stride* - stride = 1 dimensionality reduction of -1 in both dimensions, stride = 2 $\Longrightarrow$ -2 ...

- **filters** can be used to bias the algorithm in a certain direction

  - while converging a square of pixels is reduced to a single value, if this square has a inherent weighting of all 0s and only 1s in one line, the resulting layer will filter for horizontal images, see 9

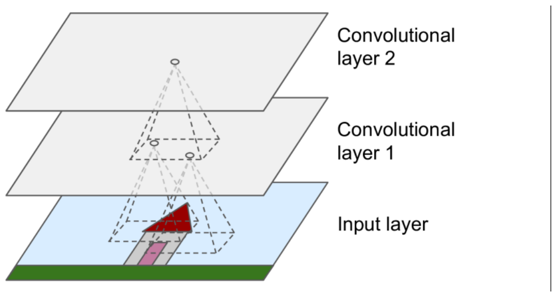


Figure 8: Convolutional Layer. Convolutional layers don't connect every neuron to every neuron of the lext layer, but rather group the converging process, thus reducing the number of weights, whilst promoting patterns in a specific area.

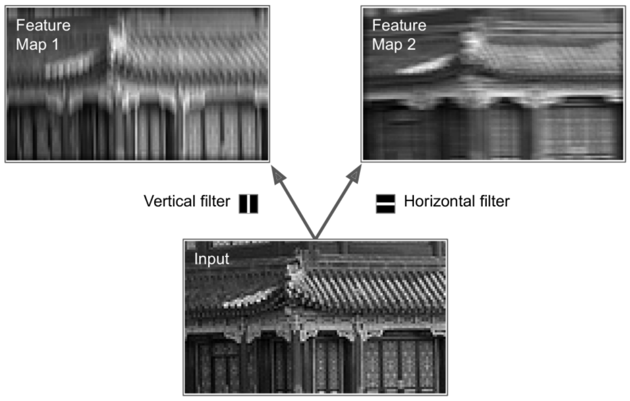


Figure 9: Filter applied between convolutional layers. The inherent weighting in the converging square shapes the bias of the next layer.

  - the filters get optimized automatically, causing certain patterns to be far more prominent than others

- the number of filters is not limited to one, usually convolutional layers have multiple filters and output multiple feature maps - one per filter
- so for one are of pixels different filters are applied, allowing the layer to detect different patterns in the same area
  $\longrightarrow$ the subsequent layer will have the same convergence, leading to pattern recognition on the already abstracted previous feature map - this way small and large scale patterns can be learned
- neurons of one feature map all have the same parameters: weights and bias term $\Longrightarrow$ this dramatically reduces the number of parameters in the model
  $\longrightarrow$ this way a pattern in one location will also be recognized in another location
- the equation used to compute the output of a neuron in a conv layer:

$$z_{i,j,k} = bias_{FeatMap} + \sum_{row=0}^{height-1} \sum_{col=0}^{width-1} \sum_{PrevFeatMap=0}^{FeatMapsInPrevLayer-1} x_{(i \cdot stride_{row}+row), (j \cdot stride_{col}+col), PrevFeatMap} \cdot w_{row, col, PrevFeatMap, FeatMap} \tag{15}$$

$\longrightarrow$ x is output neuron in layer l-1 (in $(i \cdot stride_{row} + row,\, j \cdot stride_{col} + col)$ and PrevFeatMap)
$\longrightarrow$ w is the connection weight between neuron of layer l and its input (in (row, col) and FeatMap)
$\longrightarrow$ z is the output of neuron (in (i, j) and FeatMap)

.5.2 Implementation in TensorFlow

- input images are represented as 3D tensor - [height, width, channels]
  $\longrightarrow$ mini-batch is thus a 4D tensor - [mini-batch size, height, width, channels]
  $\longrightarrow$ layers of convolutional layer also 4D tensor - [height, width, FeatMapsInPrevLayer, FeatMapInCurrentLayer]
  $\longrightarrow$ bias terms of convolutional layers are 1D tensor - [FeatMapInCurrentLayer]

- Implementation in tensorflow using 2 images with 256 depth per pixel:

```
from sklearn.datasets import load_sample_image

# Load sample images
china = load_sample_image("china.jpg") / 255
flower = load_sample_image("flower.jpg") / 255

images = np.array([china, flower])
batch_size, height, width, channels = images.shape

# Create 2 filters
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1 # vertical line
filters[3, :, :, 1] = 1 # horizontal line

outputs = tf.nn.conv2d(images, filters, strides=1, padding="SAME")
plt.imshow(outputs[0, :, :, 1], cmap="gray") # plot 1st image's 2nd
  feature map
plt.show()
```

  $\longrightarrow$ mini-batch size is 2 in this case, as we have 2 images
  $\longrightarrow$ padding = 'SAME' - zero padding is used when necessary. number of output neurons = ceil(number of input neurons / stride)
  $\hookrightarrow$ alternatively padding = 'VALID' - zero padding is not used by convolutionaly layer, this may lead to it ignoring some rows and columns

- filters were fixed here, but usually filters would be trainable variables. this can all be done in one step:

```
   conv = keras.layers.Conv2D(filters = 32, kernel_size = 3, strides = 1,
  padding ='SAME', activation = 'relu')
```

  $\longrightarrow$ this creates a Conv2D layer with32 filters, each 3x3, stride of 1, SAME padding and RELU activation function

- as there are already a lot of choices for hyperparameters in CNN layers, cross-validation can be helpful to find optimal parameters, which is however, very time consuming
- During a training session, all weights need to be stored in memory, ready to be updated when the backpropagation happens, as a result, the amount of memory (RAM) required is NumOfFeatureMaps x HeightOfFeatureMap x WidthOfFeatureMap x DepthOfFloats x NumOfInstances
  $\longrightarrow$ so 200 x 150 x 100 x 32 x 100 = 1.2 GB (200 features maps with size 150 x 100, 32-bit floats, 100 instances) - can get large very quickly
- Out of Memory Errors can be handled by higher strides, removing some layers, 16-bit floats instead of 32

.5.3 Pooling layers

- pooling layers shrink or subsample the input image, thus converging an array to a single value
  $\longrightarrow$ the idea is obviously a reduction in computational load and memory usage - ideally leading to a lower risk of overfitting
- instead of having a filter that applies a weighting scheme but preserves the size of the input image and feature map, the pooling layer aggregates all pixel values and combines them into one values
  $\longrightarrow$ this combination can be based on functions like max or mean - i.e. *max pooling layer*
- max pooling layers can be advantageous as they have a certain invariance. The max value in a square or rectangle might remain the same even if the image is slightly rotated or the scale is slightly different
  $\longrightarrow$ for classification tasks this can be crucial as not to miss patterns due to slightly different angles of a shot or different scale
  $\hookrightarrow$ not so much relevant for spectrograms, as dimensions and freq resolution is always the same
- this invariance is of course also an information loss and can be a problem, all depends on the task at hand

  ```
  max_pool = keras.layers.MaxPool2D(pool_size = 2)
  ```

  $\Longrightarrow$ this will create a max pooling layer with a 2x2 kernel and a stride of 2 (defaults to kernel size) and VALID padding
- AvgPool2D would create a average pooling layer, however usually the most prominent pixel should be preserved, resulting in max pooling layers being far more commonly used
- pooling layers can also be used to learn depth-wise, for example different filters in the convolutional layers apply different rotations and then the pooling layer identifies the filter that best identifies the feature, see fig. 10
- to use a depth-wise pooling layer, do:

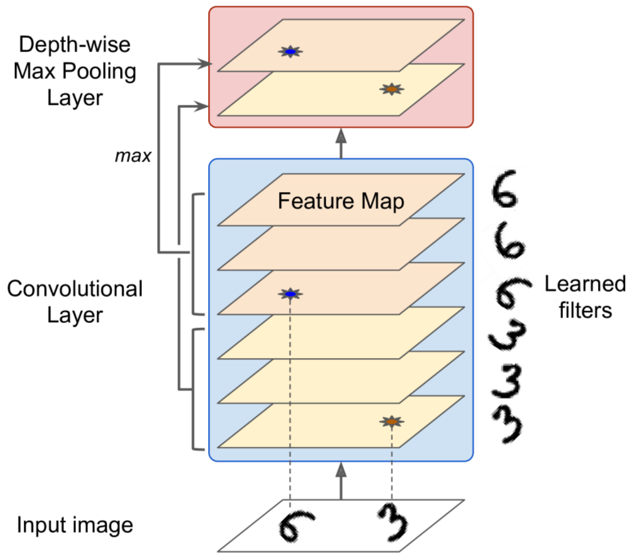


Figure 10: CNN learning hand written images with pooling along depth-dimension. Convulutional layers are used to filter for different rotations of hand written digits and then pooling layers are used to find the rotations best preserving the feature.

```
output = tf.nn.max_pool(images,
                        ksize = (1, 1, 1, 3),
                        strides = (1, 1, 1, 3),
                        padding = 'VALID')
```

$\Longrightarrow$ the (1, 1, 1, 3) indicates that the kernel size and stride along the batch, height and width sould be 1. the last digit defines the kernel size and stride along the depth dimension (must be a divisor of the input depth/number of previous feature maps)

- to include it in Keras models, you can wrap it in a Lambda layer (or create a custom Keras layer):

```
depth_pool = keras.layers.Lambda( lambda X: tf.nn.max_pool(X,
                                     ksize = (1, 1, 1, 3),
                                     strides = (1, 1, 1, 3),
                                     padding = 'VALID') )
```

- *global average pooling* layer is common nowadays - computes the mean of entire feature map into one value

```
global_avg_pool = keras.layers.GlobalAvgPool2D()
global_avg_pool = keras.layers.Lambda( lambda X: tf.reduce_mean(X,
                                     axis=[1, 2]) )
# the second one does the same as the above
```

.5.4 CNN Architectures

- typically CNNs are comprised of a few convolutional layers (in addition to a subsequent ReLU layer) then a pooling layer, more convolutional layers (+ ReLU), then another

pooling layer and so forth
$\longrightarrow$ image gets smaller due to pooling
$\longrightarrow$ number of feature maps increases, image gets *deeper* resulting from convolutional layers
$\hookrightarrow$ finally, a regular feedforward neural network comprised of fully connected layers (+ReLU) and a final layer output a prediction (or class probabilities using for example a softmax (function to draw boundaries between classes) layer)

- convolutional kernels that are too large can be problematic, use two stacked layers with 3x3 kernels rather than one 5x5 kernel
$\longrightarrow$ less parameters and less computations required
$\hookrightarrow$ exception: initial layer: 5x5 and stride = 2 is good for first layer - decreases dimension but loss is not to heavy (might not apply for spectrograms)

- example implementation for a MNIST dataset:

```
from functools import partial

DefaultConv2D = partial(keras.layers.Conv2D,
                        kernel_size=3,
                        activation='relu',
                        padding="SAME")

model = keras.models.Sequential([
        DefaultConv2D(filters=64, kernel_size=7,
                      input_shape=[28, 28, 1]),
        keras.layers.MaxPooling2D(pool_size=2),
        DefaultConv2D(filters=128),
        DefaultConv2D(filters=128),
        keras.layers.MaxPooling2D(pool_size=2),
        DefaultConv2D(filters=256),
        DefaultConv2D(filters=256),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Flatten(),
        keras.layers.Dense(units=128, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(units=64, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(units=10, activation='softmax'),
])
```

$\Longrightarrow$ the partial in the beginning allows us to define a default that doesn't need to be repeated over and over
$\longrightarrow$ with input image size of 28x28 and grayscale, a stride of 7 with no stride because images are not that large
$\longrightarrow$ pooling layer, reducing the spatial dimension by factor of 2
$\longrightarrow$ repeat the structure of 2 conv layers and one pooling layer
$\hookrightarrow$ depending on input image size this process can be repeated more than twice
$\longrightarrow$ doubling of filters after pooling layers (which reduce spatial dimensions by factor 2) is common, as number of higher level features (embedded deeper into image, only get

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Dense	-	10	-	-	RBF
F6	Dense	-	84	-	-	tanh
C5	Convolution	120	1x1	5x5	1	tanh
S4	Avg Pooling	16	5x5	2x2	2	tanh
C3	Convolution	16	10x10	5x5	1	tanh
S2	Avg Pooling	6	14x14	2x2	2	tanh
C1	Convolution	6	28x28	5x5	1	tanh
In	Input	1	32x32	-	-	-

Table 1: Architecture of LeNet-5. With increasing layer number, maps increase and size decreases resulting from the pooling layers.

revealed after a lot of pooling) is bigger than lower level features
$\longrightarrow$ finally the fully connected network, made of 2 hidden layers and dense output layer. inputs must be flattened because dense networks expect 1D arrays
$\hookrightarrow$ the Dropout layers with a dropout rate of 50 % are used to reduce overfitting

- this is not state of the art but it is a good network, reaching 92 % accuracy on a MNIST fashion dataset
- ILSVRC ImageNet challenge is a good measure of keeping up to date with new networks, in the last 5 years major improvements in the winning networks have been achieved, being able to differentiate between 120 dogbreeds for example

.5.5 LeNet-5

- most widely known CNN architecture
- used for MNIST handwritten digits
- structure see table 1
- Avg Pooling used instead of max pooling - each neuron computes the mean of its inputs, multiplies the result by a leanable coefficient (one per map) and adds a trainable bias term (also 1 per map) and then applies activation function
- Out layer computes the matrix multiplication of inputs and wight vector, each neuron outputs the square of the distance between its input vector and its weight vector - the output then measures how much the image belongs to a particular digit class
  $\longrightarrow$ cross entropy (sec .2.4) preferred as a cost function as it penalizes bad predictions much more, yielding larger gradients and converging faster

.5.6 AlexNet

- won the ILSVRC challenge by large margin
- was the first to stack convolutional layers directly
- table 2 shows the architecture
- AlexNet uses local response normalizaion (LRN) after C1 and C3
  $\longrightarrow$ the most strongly activated neurons inhibit other neurons in the same position but neighboring faeture maps, ths forcing them to explore a wider range of features, improving generalization

$$b_i = a_i(k + \alpha \sum_{j=j_{low}}^{j_{high}} a_j^2)^{-\beta} \quad \text{with} \begin{cases} j_{high} = \min(i + \frac{r}{2}, f_n - 1) \\ j_{low} = \max(0, i - \frac{r}{2}) \end{cases} \tag{16}$$

  $\longrightarrow$ b_i - normalized output of neuron located in feature map i, at some position
  $\longrightarrow$ a_i - activation of that neuron after ReLU before normalization
  $\longrightarrow$ k, α, β, r - hyperparameters, with k = bias, r = depth radius
  $\longrightarrow$ f_n is the number of feature maps

- if r=2 and a neuron has a strong activation, the neurons in similar positions and neighboring feature maps will be inhibited
- for AlexNet: $r = 2, \alpha = 0.00002, \beta = 0.75, k = 1$
  $\longrightarrow$ can be altered with

```
tf.nn.local_response_normalization()
```

.5.7 GoogLeNet

- uses *inception modules* allowing it to vastly reduce number of parameters
  - inception modules employ different layer combinations that are used in parallel and concatenated in the end (using tf.concat(axis=3 (depth)))
  - different kernel sizes are used and combined, also 1x1 kernels are used, that don't reduce in spatial dimensions but rather in depth dimensions, outputting fewer feature maps than were inputted
  - fig. 11 shows an inception module
  - every inception layer requires 6 hyperparameters - the kernel sizes of the respective convolutional layers and the max pooling layer
- fig .5.7 shows the GoogLeNet architecture
- first the image is reduced in size by the first two layers
  $\longrightarrow$ a LRN is used to learn a variety of features
  $\longrightarrow$ a convolutional network acts as a bottleneck layer - reducing dimensionality (and

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Dense	-	1000	-	-	-	Softmax
F9	Dense	-	4096	-	-	-	ReLU
F8	Dense	-	4096	-	-	-	ReLU
C7	Convolution	256	13x13	3x3	1	SAME	ReLU
C6	Convolution	384	13x13	3x3	1	SAME	ReLU
C5	Convolution	384	13x13	3x3	1	SAME	ReLU
S4	Max Pooling	256	13x13	3x3	2	VALID	-
C3	Convolution	256	27x27	5x5	1	SAME	ReLU
S2	Max Pooling	96	27x27	3x3	2	VALID	-
C1	Convolution	96	55x55	11x11	4	VALID	ReLU
In	Input	3 (RGB)	227x227	-	-	-	-

Table 2: Architecture of AlexNet. Much larger than the LeNet-5, and the first to stack convolutional layers.

thus parameters)
$\longrightarrow$ another LRN
$\longrightarrow$ max pooling, reduces spatial dimensions
$\longrightarrow$ a stack of nine inception layers with some max pooling in between to reduce dimensions and speed up the net
$\longrightarrow$ global avg pooling gives average of entire feature map (stripping any spatial dimension)
$\longrightarrow$ dropout for regularization, fully connected with 1000 units (1000 classes) and softmax activation to get class probabilities
$\hookrightarrow$ the global average pooling layer can be used, seeing that not much spatial information was left after the 5 fold dimensionality reduction, and location is not of interest, so the crucial information is, does this feature exist or not - which is a lot quicker with just one values per feature map

- the massive decrease in spatial information vastly sped up the process, as the final dense layers where only one dimensional (compared to AlexNet where the dense layers were huge)

.5.8 VGGNet

- very simple classical architecture:
  - 2 or 3 convolutional layers $\Longrightarrow$ pooling layer $\Longrightarrow$ 2 or 3 convolutional layers $\Longrightarrow$ pooling ... $\Longrightarrow$ dense layer with 2 hidden layers and output
  - all filters were 3x3

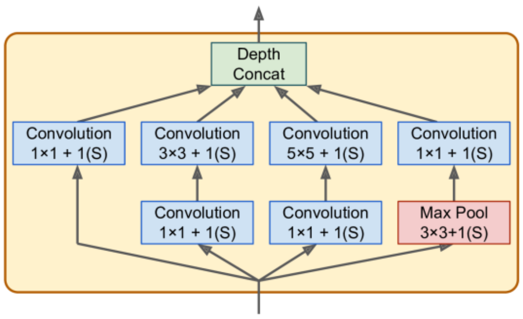


Figure 11: Inception Module. Inception module used for example in GoogLeNet. '3x3 + 1(s)' means 3x3 kernel, stride of 1, and same padding. Because stride is 1 everywhere the results have the same spatial dimensions and can be concatenated in the final step.

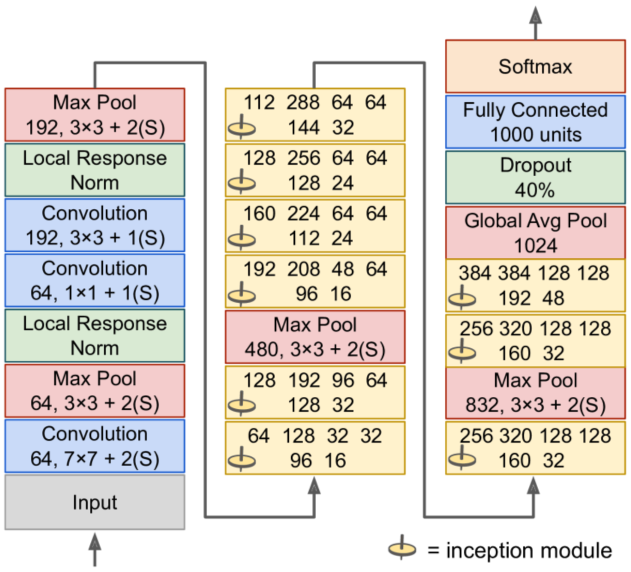


Figure 12: Architecture of GoogLeNet. This massive CNN is GoogLeNet. The inception modules show the number of feature maps outputted by each of the convolutional layers in the inception modules. All convolutional layers use ReLU activation.

.5.9 ResNet

- very deep model with 152 layers but few parameters
- the key is **residual learning**
  - fig. 13 shows, the approach of residual learning:
    $\longrightarrow$ the input is added to the output of the model, resulting in the model having to learn $f(x) = h(x) - x$ rather than $f(x) = h(x)$, with $h(x)$ being the function we want to model
  - as the weights are random in the beginning, the activation values will be close to 0, so $f(x) \approx 0$ in the early learning phases, thus $f(x) = h(x) - x \iff h(x) \approx x$, which is close to the input
    $\longrightarrow$ this allows to speed up the training process, because the learning process is a lot quicker in the beginning
  - a network with many skip connections will be able to learn even if some of the convolutionaly layers have not started to learn yet
  - a deep residual netweork can be seen as a stack of residual units, where each residual unit is a stamm neural network with a skip connection
  - fig. 14 shows the primary advantage of using residual neural networks (i.e. empolying skip connections)
- Fig. 15 shows the architecture of the ResNet
- the structure is similar in that of the GoogLeNet when comparing the first and last layers. The main difference is the usage of residual units vs. inception units
- residual units are comprised of two convolutional layers (no pooling layer!) with Batch Noramlization and ReLU activation
  $\longrightarrow$ 3x3 kernels and preserving spatial dimensions
- number of feature maps is doubled every few residual units, when stride of 2 is used for one unit, halving width and height
  $\longrightarrow$ as this conflicts with the skip connection, which preserves its dimensions, the skip connection is passed through a convolutional layer with stride 2 and kernel size 1x1
- ResNet architectures are quite common, with ResNet-34 being a much employed model
  $\longrightarrow$ 34 layers (only counting convolutional and dense layers), with 3 residual units (RU) with 64 feat maps, 4 RUs with 128 maps, 6 RUs with 256 maps, and 3 RUs ith 512 maps
- larger ResNets than ResNet-34 tend to use RUs with 3 instead of 2 convolutional layers

.5.10 Xception

- author of Keras (Francois Chollet) merged GoogLeNet and ResNet replacing inception modules with *depthwise separable convolution*

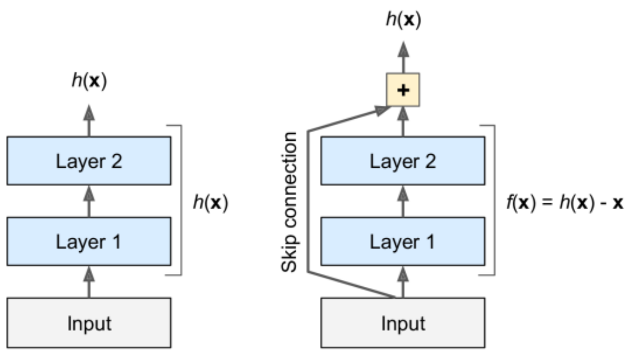


Figure 13: Residual learning. By skipping the modelled function a different function is learned which turns out to be advantageous.

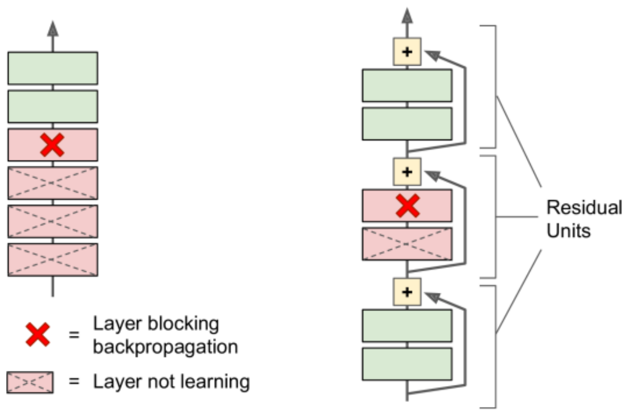
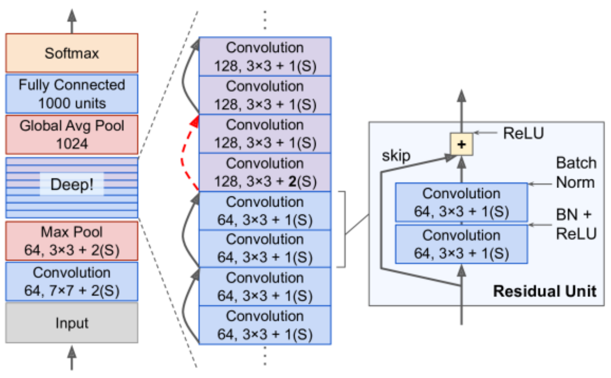


Figure 14: Regular deep neural network vs. deep residual network. The left side shows a conventional deep neural network, where learning is not possible if one layer in the chain has not learned yet, blocking backpropagation, whereas the residual neural network can learn anyways, due to the skip connections.



Softmax
Fully Connected 1000 units
Global Avg Pool 1024
Deep!
Max Pool 64, 3×3 + 2(S)
Convolution 64, 7×7 + 2(S)
Input
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 2(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
skip
ReLU
Batch Norm
BN + ReLU
Convolution 64, 3×3 + 1(S)
Convolution 64, 3×3 + 1(S)
Residual Unit

Figure 15: Architecture of ResNet. Similar to GoogLeNet in the beginning and end, but with a lot of residual units in the middle in contrast to the inception units.

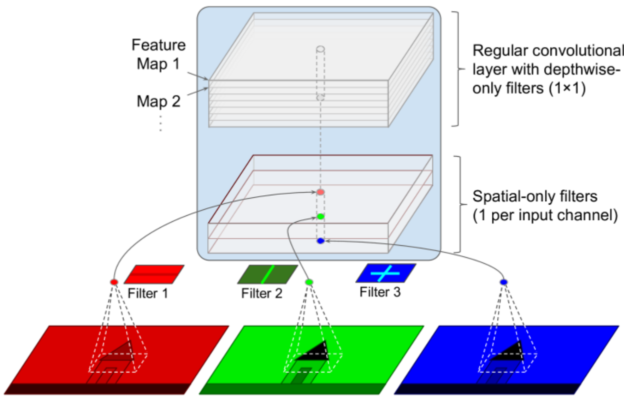

Feature Map 1
Map 2
Regular convolutional layer with depthwise-only filters (1×1)
Spatial-only filters (1 per input channel)
Filter 1
Filter 2
Filter 3

Figure 16: Depthwise Seperable Convolution is used to independently evalueate spatial and depthwise patterns contrary to regular convolutional layers that look at them jointly.

  - fig. 16 shows a depthwise seperable convolutional layer, it takes an input (usually would not be used after the input, but rather further on in the network) and evaluates spatial-only filters (1 per input channel, further on in the network these correspond to feature maps) and at the same time applies depthwise-only filters finally connecting the two
  - seperable convolutions use less parameters, les smemory, and less computations and usually perform better

- the Xception network combines regular convolutional layers and the seperable convolution layers, thus combining methods of independently looking for cross-channel patterns as well as jointly. This has yielded strong results

.5.11 SENet

- squeeze-and-Excitation Network - similar in structure to ResNets or inception networks, but the performance is boosted by the introduction of an **SE Block** to every unit in the original architecture
  - fig. 17 shows in what way the SE Block is included in the original architecture
  - a SE Block analyzes the output of the unit it's attached to
  - only looks depth-wise
  - sees what features are active together
  - it applies the spatial concept of firing and wiring together and applies it to features
    $\longrightarrow$ if the features of eyes, noses and mouths always fire together and in one case only nose and mouth features fire up, the SE block will inhibit other irrelevant features to boost the eyes feature
    $\hookrightarrow$ this can be advantageous when there is something blocking an eye which can thereby be revealed
  - fig. 18 shows how the input feature map is multiplied by the recalibrated feature map, causing features that would otherwise pass unnoticed to be noticed

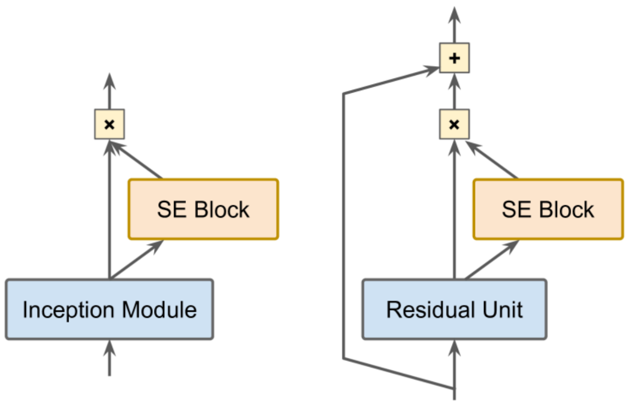


Figure 17: SE-Block in inception modules and residual units. The introduction of this small neural network boosts the performance of the original network.

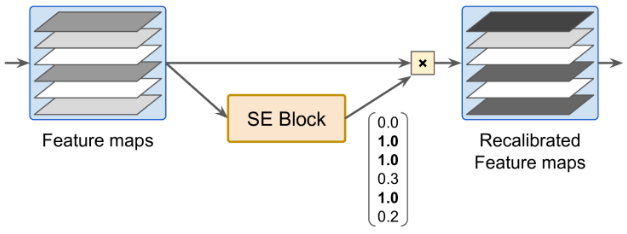


Figure 18: Recalibration by SE Block. This causes a boosting of features that would normally light up with other features, removing ambiguity.

- SE Blocks are just composed of 3 elements:
  $\longrightarrow$ Global Average Pooling layer - gives out one value per feature map
  $\longrightarrow$ hidden dense layer using ReLU activation - vastly reduces dimension causing the SE Block to learn what belongs together (low dimensional vector representation -> embedding)
  $\longrightarrow$ dense output layer using sigmoid activation - takes embeddin and outputs recalibration vector with values between 0 and 1 per feature map
- irrelevant features get numbers close to 0 - get ignored, relevant features get values close to 1 - unity

.5.12 Implementation of a ResNet-34 CNN Using Keras

- CNN implementation is straight forward for the shown models
- create standard convolutional layer first to not repeat every time:

```
DefaultConv2D = partial(keras.layers.Conv2D, kernel_size = 3,
                        strides = 1, padding = 'SAME', use_bias = False
)
```

- create ResidualUnit layer

```
class ResidualUnit(keras.layers.Layer):
  def __init__(self, filters, strides=1,
              activation="relu", **kwargs):
    super().__init__(**kwargs)
    self.activation = keras.activations.get(activation)

    self.main_layers = [
          DefaultConv2D(filters, strides=strides),
          keras.layers.BatchNormalization(),
          self.activation,
          DefaultConv2D(filters),
          keras.layers.BatchNormalization()
      ]

    self.skip_layers = []
    if strides > 1:
      self.skip_layers = [
        DefaultConv2D(filters, kernel_size=1, strides=strides),
                      keras.layers.BatchNormalization()
                      ]

  def call(self, inputs):
    Z = inputs
    for layer in self.main_layers:
      Z = layer(Z)

    skip_Z = inputs
    for layer in self.skip_layers:
      skip_Z = layer(skip_Z)

    return self.activation(Z + skip_Z)
```

- now that the class is defined, every RU can be treated as a layer in a sequential model:

```
model = keras.models.Sequential()
model.add(DefaultConv2D(64, kernel_size=7, strides=2,
                        input_shape=[224, 224, 3]))

model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation("relu"))
model.add(keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"
))

prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    model.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters

model.add(keras.layers.GlobalAvgPool2D())
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation="softmax"))
```

- the iterable, that the for loop iterates through is necessary to get the correct number of feature maps for the RUs

- to make sure that the stride only gets increased on the third and last iteration of one feature map size, the if clause checks for filter == prev-filters

.5.13 Easy Guide to Prediddctions with Pretrained Models

- standard models like GoogLeNet or ResNet readily available through the keras API:

```
model = keras.applications.resnet50.ResNet50(weights = 'imagenet')
```

- this will yield a ResNet-50 model and download weights pretrained on the ImageNet dataset
  ⟶ images have to have right size (224 x 224)
  ⟶ resizing is possible with

```
tf.image.resize(__your_images__, [__new_x_and_y_dimensions__])
tf.image.crop_and_resize() # can be used in case preservation of aspect
 ratio is crucial
```

- models have inherent preprocessing pipelines, to apply them to your inputs, your pixels in the images need a range from 0 - 255,
  ⟶ both can be achieved in one step:

```
inputs = keras.applications.resnet50.preprocess_input(__your_images__ *
 255) # assuming images where between 0 and 1
```

- finally predictions can be made:

```
Y_test = model.predict(inputs)
```

- resulting Y-test is a matrix with dimensions = [number of images, number of classes]
- to decode the matrix and show the top k number of predictions:

```
top_K = keras.applications.resnet50.decode_predictions(Y_proba, top=3)
for image_index in range(len(images)):
    print("Image #{}".format(image_index))
    for class_id, name, y_proba in top_K[image_index]:
        print(" {} - {:12s} {:.2f}%".format(class_id, name, y_proba *
100))
    print()
```

- this is a very easy setup to test pretrained models for your purpo(i)ses

.5.14 Pretrained Models for Transfer Learning

- when datasets of a pretrained model was too different from your dataset, but you still want to make use of their learning time, use **transfer learning**
- especially the lower levels can be resused, saving a lot of computational time

```
import tensorflow_datasets as tfds

dataset, info = tfds.load('tf_flowers', as_supervised = True
                          with_info = True)
dataset_size  = info.splits["train"].num_examples # 3670
class_names = info.features["label"].names # ["dandelion",
                                            "daisy", ...]
n_classes = info.features["label"].num_classes # 5
```

- this is a TensorFlow Dataset of flowers that we want to use in an Xception model to classify them
- we still need to split our dataset into train, test, and validation

```
test, train, valid = tfds.Split.TRAIN.subsplit([10, 15, 75])

test_set = tfds.load("tf_floswer", split = test, as_supervised = True)
valid_set = tfds.load("tf_floswer", split = valid, as_supervised = True
)
train_set = tfds.load("tf_floswer", split = train, as_supervised = True
)
```

- to make the images of the new dataset compatible with the model, the images needs to be resized and the preprocess_input function needs to be run:

```
def preprocess(image, label):
    resized_image = tf.image.resize(image, [224, 224])
    final_image = keras.applications.xception.preprocess_input(
resized_image)
    return final_image, label
```

- now we apply the preprocessing to all 3 sets and shuffle train set, additionally we add batching and prefetching:

```
batch_size = 32
train_set = train_set.shuffle(1000).repeat()
train_set = train_set.map(preprocess).batch(batch_size).prefetch(1)
valid_set = valid_set.map(preprocess).batch(batch_size).prefetch(1)
test_set = test_set.map(preprocess).batch(batch_size).prefetch(1)
```

- at this stage optional augmentation methods could be introduced by replacing preprocess with other methods
  $\longrightarrow$ for example: tf.image.random_crop() or tf.image.random_flip_left_right()

- now that the input is ready, we load the Xception model without the global average pooling layer and the dense output layer, instead we add our own

```
base_model = keras.applications.xception.Xception(weights="imagenet",
                                                  include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
output = keras.layers.Dense(n_classes, activation="softmax")(avg)
model = keras.models.Model(inputs=base_model.input, outputs=output)

for layer in base_model.layers:
    layer.trainable = False
```

- the last part is important to freeze the weights while we train our new layers, this will be changed after a while
- finally the training is performed:

```
optimizer = keras.optimizers.SGD(lr=0.2, momentum=0.9, decay=0.01)
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=optimizer, metrics=["accuracy"])
history = model.fit(train_set,
            steps_per_epoch=int(0.75 * dataset_size / batch_size),
            validation_data=valid_set,
            validation_steps=int(0.15 * dataset_size / batch_size),
            epochs=5)
```

- after a while the validation accuracy should reach 75-80%, and stagnate, indicating that the top layers are well trained and it's time to unfreeze the weights:

```
for layer in base_model.layers:
    layer.trainable = True

optimizer = keras.optimizers.SGD(lr=0.01, momentum=0.9, decay=0.001)
model.compile(...)
history = model.fit(...)
```

- the recompiling of the model is necessary for the changes to take effect
- the learning rate is far lower than before to make sure that the top layer weights don't get ruined again
- using all hidden layers except for the output layer of a pretrained model

```
model_A = keras.models.load_model("my_model_A.h5")
model_B_on_A = keras.models.Sequential(model_A.layers[:-1])
model_B_on_A.add(keras.layers.Dense(1, activation="sigmoid"))
# the above method will also alter model_A when training,
# if that is a problem, do the following:
model_A_clone = keras.models.clone_model(model_A)
model_A_clone.set_weights(model_A.get_weights())
```

- the output layer of model_B_on_A needs to be trained while the other layers need to remain the same:

```
for layer in model_B_on_A.layers[:-1]:
    layer.trainable = False
model_B_on_A.compile(loss="binary_crossentropy",
                      optimizer="sgd",
                      metrics=["accuracy"])
```

Taxonomy

Datum: 04.Mai

.1 Pfad zu Kateceanen

- hierarchy of taxonomic ranks:
  $\longrightarrow$ Life $\Longrightarrow$ Domain $\Longrightarrow$ Kingdom $\Longrightarrow$ Phylum $\Longrightarrow$ Class $\Longrightarrow$ Order $\Longrightarrow$ Family $\Longrightarrow$ Genus $\Longrightarrow$ Species

- Rangstufen der Taxonomie:
  $\longrightarrow$ Lebewesen $\Longrightarrow$ Domäne $\Longrightarrow$ Reich $\Longrightarrow$ Stamm $\Longrightarrow$ Klasse $\Longrightarrow$ Ordnung $\Longrightarrow$ Familie $\Longrightarrow$ Gattung $\Longrightarrow$ Art

- Animalia (Kingdom) $\Longrightarrow$ Chordata (Phylum) $\Longrightarrow$ Mammalia (Class) $\Longrightarrow$ Artiodactyla (also even-toed ungulates $\Longrightarrow$ lollinger) (Order)

- Tierreich (Reich) $\Longrightarrow$ Chordatiere (Stamm) $\Longrightarrow$ Säugetiere (Klasse) $\Longrightarrow$ Laurasiatheria (Überordnung)

- Catecean (Infraorder)

- Katecean (Ordnung)
  $\hookrightarrow$ (umgangssprachl. Wal, wobei Wale eignetlich nur bartenwale bezeihnen)

.2 Cetaceans

.2.1 Baleen whale (mysticetes) (Parvorder) - Bartenwale (Unterordnung)

Rorqual (balaenopteridae) (Family) - Furchenwale (Familie)

- Megaptera (Genus) - Megaptera (Gattung)
  $\hookrightarrow$ verfügt nur über eine Art, den Buckelwal
  $\longrightarrow$ Humpback whale (Species) - Buckelwal (Art)

- Balaenoptera (Genus) - Balaenoptera (Gattung)
  $\longrightarrow$ Blue whale (Species) - Blauwal (Art)
  $\longrightarrow$ Minke whale (Species) - Zwergwal (Art)
  $\longrightarrow$ Fin whale (Species) - Finnwal (Art)
  $\hookrightarrow$ second longest animal after blue whale

Balaenidae (Family) - Glattwale (Familie)
$\hookrightarrow$ wurden am stärksten durch Jagen dezimiert

- Right whale (eubalaena) (Genus) - Eubalaena (Gattung)
  $\longrightarrow$ North atlantic right whale (eubalaena glacialis) (Species) - Atlantischer Nordkaper (Art)

- Balaena (Genus)
  $\longrightarrow$ Bowhead whale (balaena mysticetus) (Species) - Grönlandwal (Art)

.2.2 Toothed whale (odontocetes) (Parvorder) - Zahnwale (Unterordnung)

⟶ (- Delphinida (Clade))
⟶ (- Delphinoidae (Superfamily) - Delfinartige (Überfamilie))
Oceanic dolphins (delphinidae) (Family) - Delfine (Familie)

- Orcinus (Genus) - Orcinus (Gattung)
  ⟶ Orca or killer whale (Species) - Schwertwal auch Orca, Killerwal (Art)

⟶ (- Physeteroidea (Superfamily))
Physeteridae (Family)

- Physeter (Genus)
  ⟶ Sperm whale (physeter macrocephalus) (Species) - Pottwal (Art)
  ↪ largest toothed animal

References

[1] A. N. Allen, M. Harvey, L. Harrell, A. Jansen, K. P. Merkens, C. C. Wall, J. Cattiau, and E. M. Oleson. A Convolutional Neural Network for Automated Detection of Humpback Whale Song in a Diverse, Long-Term Passive Acoustic Dataset. *Frontiers in Marine Science*, 8, 2021.

[2] P. C. Bermant, M. M. Bronstein, R. J. Wood, S. Gero, and D. F. Gruber. Deep machine learning techniques for the detection and classification of sperm whale bioacoustics. *Scientific reports*, 9(1):1–10, 2019. Publisher: Nature Publishing Group.

[3] D. M. Cholewiak, R. S. Sousa-Lima, and S. Cerchio. Humpback whale song hierarchical structure: Historical context and discussion of current classification issues. *Marine Mammal Science*, 29(3):E312–E332, 2013. Publisher: Wiley Online Library.

[4] E. C. Garland and P. K. McGregor. Cultural transmission, evolution, and revolution in vocal displays: insights from bird and whale song. *Frontiers in psychology*, page 2387, 2020. Publisher: Frontiers.

[5] E. C. Garland, L. Rendell, M. S. Lilley, M. M. Poole, J. Allen, and M. J. Noad. The devil is in the detail: Quantifying vocal variation in a complex, multi-levelled, and rapidly evolving display. *The Journal of the Acoustical Society of America*, 142(1):460–472, 2017. Publisher: Acoustical Society of America.

[6] A. Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Inc., 2019.

[7] E. Mercado and C. E. Perazio. All units are equal in humpback whale songs, but some are more equal than others. *Animal Cognition*, 25(1):149–177, Feb. 2022.

[8] J. S. Reidenberg. Terrestrial, semiaquatic, and fully aquatic mammal sound production mechanisms. *Acoust Today*, 13(2):35–43, 2017.