Vilken fågel sjunger?

Maja Gajic, Kåre von Geijer, Emil Sandelin, Oskar Åström
Handledare: Maria Sandsten

February 2020

Abstract

This study investigates how different visualisations of birdsongs affect a neural networks ability to classify the songs. The used visualisation are a vector representation of the audiofile, a spectrogram with high frequency resolution and a spectrogram with high time resolution. The results show that a spectrogram with high time resolution is superior to the other methods.

Keywords: Convolutional Neural Network, Bird Song, Spectrogram, Simulation of Sound, MatLab

Innehåll

1 Problemformulering

Syftet med projektet är att jämföra olika sätt att klassificera vilken fågel som sjunger, givet en ljudfil. Specifikt delas ljudfilerna upp i korta fågelkvitter (*stavelser*) som analyseras var för sig med hjälp av neurala nätverk i Matlab.

Som indata till de neurala nätverken kommer tre olika representationer skickas in. Först bara en vektor av ljudet, sen två spektrogram med olika upplösning. Dessutom kommer två olika metoder användas för att välja ut de data som nätverken tränas respektive testas på.

Slutligen kommer dessa olika metoder testas på två olika datamängder. Den första består av filer från talgoxar, bofinkar och gråsparvar. Den andra mängden består av samma filer samt ytterligare filer från blåmesar, bergfinkar och pilfinkar.

2 Sammanfattning av använd litteratur

Handledaren gav oss tio artiklar som behandlade liknande problem. Dessa rapporter användes som inspiration och faktainsamling för att utveckla den metod som används. Främst användes två av rapporterna som hjälp för att utveckla vissa delar av metoden. Dessa finns i källförteckningen. Dessutom användes dokumentationen från Matlab för att läsa på om olika funktioner.

3 Teori

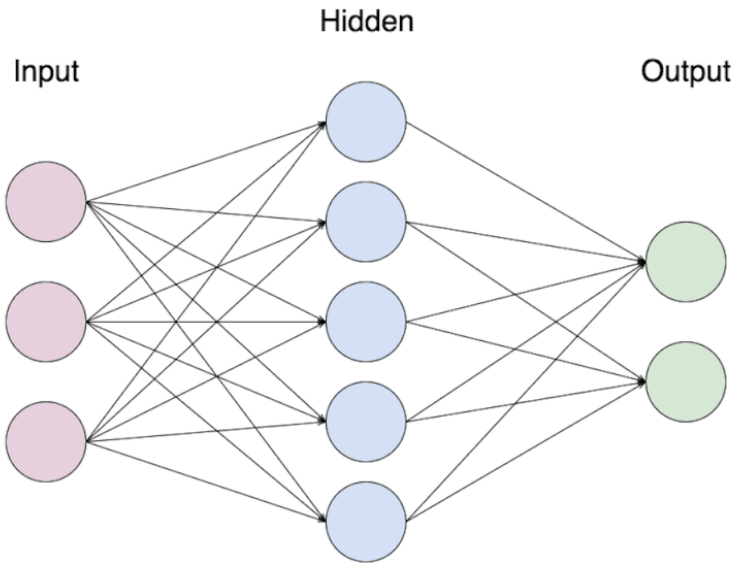
För att förstå arbetet underlättar det att introducera vissa begrepp. Det vi kallar en *ballad* refererar till en hel ljudfil av fågelsång. Ofta runt en minut lång. En *strof* refererar till ett stycke av vad som uppfattas som kontinuerlig fågelsång. Dessa är separerade av ett längre stycke tystnad som ofta fylls med brus. En strof är uppbyggd av mindre delar som kallas *stavelser*. En exakt definition av stavelser ges inte men de kan tänkas som enstaka läten som är cirka 170 ms långa. Men deras längd kan variera ganska mycket. Allt mellan en till tio stavelser kan bygga upp en strof.

3.1 Convolutional Neural Network

I projektet användes convolutional neural network (CNN) för att klassificera de olika fågelsångerna. Convolutional neural networks är inspirerade av hur hjärnans neuroner fungerar för att uppfatta information, och består av neuroner som är ordnade i olika lager. Antalet neuroner i första lagret är lika med antalet sampel i datamängderna och det sista lagret har lika många neuroner som antalet kategorier som datamängden kan klassificeras som. Däremellan finns ett antal ”gömda” lager. Varje neuron i ett lager är anslutet till alla neuroner i nästa, se Figur 1. Detta görs genom en viktad funktion mellan lagrena där varje neuron i det tidigare lagret har en associerad vikt (*weights*). Till den viktade delen av funktionen adderas även en konstant som kallas för funktionens *bias*. Ett

CNN lär sig klassificera data genom att appliceras på redan kategoriserad data, uppdelat i träningsdata och valideringsdata. Resterande data kan användas för att utvärdera modellen och kallas testdata. En bild från träningsdatan tar sig igenom hela nätverket och sedan beräknas resultatet och jämförs med den förväntade klassificeringen. Genom optimeringsalgoritmer beräknar det sista lagret vilka parametrar hos dess funktioner som kan ändras för att ge rätt resultat för den bilden, samt vilken input från det tidigare lagret som hade gett bäst resultat med de nuvarande parametrarna. Detta skickas sedan tillbaks till lagret innan som upprepar detta. När alla lager har kommit fram till en effektiv förändring för att just denna bild ska få ett bra resultat upprepas denna optimering för nästa bild. När ett antal bilder har undersökts tar nätverket medelvärdet av alla dessa önskade förändringar och applicerar detta på sig själv. En sådan cykel kallas för en *iteration*. När alla bilder i träningssetet har undersökts har det gått en *epok*. Efter ett visst antal epoker testas nätverket på valideringsdata, för att avgöra om det fungerar väl på data det inte tränats på. Antalet epoker som körs är förutbestämt av användaren. Utöver det kan nätverkets inlärningshastighet, *learningRate*, ändras. Variabeln kan jämföras med stegstorleken i numerisk differentialanalys, det måste vara litet nog för att konvergera men beräkningarna måste avslutas i rimlig tid.

Ett CNN är lätt att använda eftersom det är självlärande, användaren behöver då ingen kunskap om de exakta detaljerna om hur det fungerar [4].



Figur 1: Grafisk representation av hur ett neuralt nätverk fungerar.

3.2 Spektrogram

Ett spektrogram är en grafisk represantation av tids- och frekvensanalys, där själva grafen är en bild av intensiteten av frekvenserna. När spektrogrammet skapas kan man använda sig utav fönster $h[n]$ med olika storlek, dessutom kan dessa fönster överlappa varandra. Spektrogrammet av en signal $x[n]$ beräknas som

$$spectrogram(m,h) = |X(m,h)|^2$$

där,

$$X(m,h) = \sum_{n=-\infty}^{\infty} x[n]h[m-n]e^{-jwn}$$

är korttids Fouriertransform. Fönsterstorleken innebär varierande upplösning i tid och frekvens. Ett långt fönster ger bättre frekvensupplösning medan ett kort fönster ger bättre tidsupplösning. [5]

4 Metod

Här ges en beskrivning av vad som gjordes i projektet. I nästa del *Implementation* beskrivs mer ingående hur varje del implementerades och utfördes.

Detta projekt valde att stycka upp balladerna i stavelser som sedan analyserades. För att dela upp balladerna i stavelser skapades metoden *strophecut*. Därefter skulle stavelserna behandlas på olika sätt och användas på olika sätt som indata till CNN.

Stavelserna processades på tre olika sätt innan de skickades in till nätverken. Det enklaste var att skicka in vektorn av ljudet i stavelsen som indata, detta benämner vi som *ljudvektorer*. Sen skickades även två olika spektrogram in, det ena hade hög tidsupplösning, som vi benämner *HTS*, medan det andra hade låg tidsupplösning men då högre frekvensupplösning, som vi benämner *LTS*. Anledningen till detta var att se om CNN tränades bättre på redan behandlad data som spektrogram eller ren indata som ljudfilerna av stavelserna, samt att kunna jämföra hur effektiviteten av nätverket berodde på vilket spektrogram som analyserades.

Det prövades även två olika sätt att välja träningsdata, valideringsdata och testdata. Den första metoden benämns hädanefter som *burkmetoden*. Där används 70% av stavelserna för varje art som träningsdata, 15% som valideringsdata och 15% som testdata. Resultat härifrån blir hur bra de tränade CNN gissar på stavelser i testdatan.

Det andra sättet kallas hädanefter som *fågelmetoden* och där väljes en individ från varje art och alla dess stavelser sparas undan som testdata. Sen delas resterande data av stavelser upp så att 80% blir träningsdata och 20% valideringsdata. Ett CNN tränas på träningsdata samt valideringsdata för att lära sig klassificera stavelser. Som resultat fås som i burkmetoden hur bra tränade CNN klassificerar stavelser. Men här klassar även det tränade CNN fåglarna i testdatan i sin helhet genom att ta genomsnitt av de klassade stavelserna.

När CNN tränas på stavelser som beskrivs i styckena ovan används även två olika dataset på varje metod. Det första och mer grundläggande består av ballader från tre fåglar, nämligen talgoxe, bofink och gråsparv. Det andra lägger till tre ytterligare fåglar och består därmed av ballader från bergfink, blåmes, bofink, gråsparv, pilfink och talgoxe. Det som gör det andra datasetet svårare är att balladerna för vissa fåglar liknar varandra väldigt mycket. Talgoxe och blåmes, bofink och bergfink, samt gråsparv och pilfink liknar varandra mycket vilket gör det svårare att skilja dem åt.

Slutligen har en simulationsstudie gjorts. Där skapdes sex olika typer av simulerade signaler som efterliknar fågelsång. Realiseringar av dessa signaler användes sen som tränings-, validerings- och testdata för ett CNN. Meningen med dessa simuleringar var att påvisa att den använda strukturen för CNN fungerar.

5 Implementation

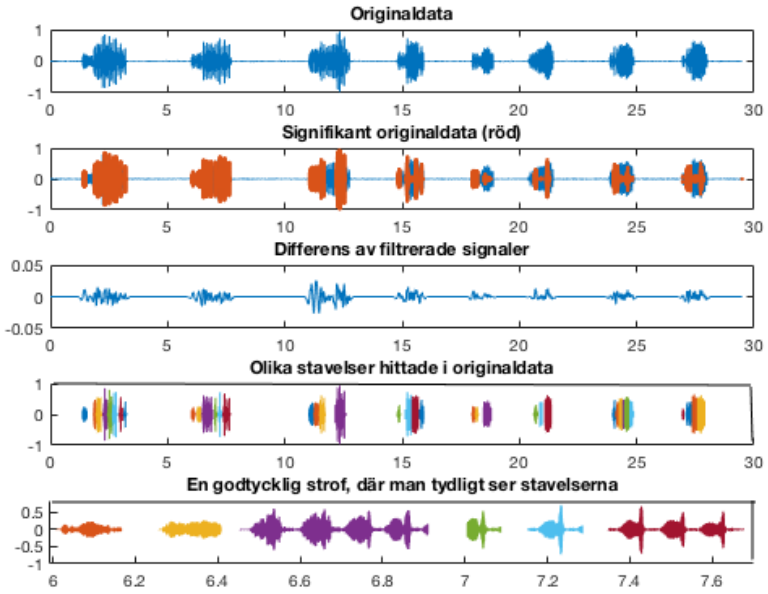
Här beskrivs mer ingående hur varje del i metoden genomfördes.

5.1 Utklippning av stavelser

För att stycka upp varje ballad i stavelser skapades metoden *strophecut*, vilken bygger på en metod använd i artikeln [2]. I *strophecut* används två Moving Average filter för att urskilja var det finns fågelsång. Det ena filtret är 360 ms och det andra 90 ms. Balladens värden i kvadrat filtreras med de två filtren och resultaten sparas i P_{kort} och $P_{lång}$ för det korta resp långa filtret. För att sedan hitta var det finns signifikant fågelsång tittar vi i alla punkter och de som uppfyller

$$P_{kort}(t) > P_{lång}(t) + (1 - tol) \cdot max(P_{lång}(t))$$

markeras som signifikant fågelsång. Där står *tol* för tolerans som vi konsekvent har satt vid 0.9 för att undvika att plocka upp brus. Indexen där det finns signifikant ljud i sparas i en separat vektor X där sedan beräknas differensen mellan intilliggande element, vilket då ger storleken av de tysta intervallen. Om ett tyst intervall är längre än 60 ms sägs det separera två stavelser. På detta sätt hittas alla stavelser i balladen. Detta missar ofta kanterna av stavelserna så när stavelserna sparas tas alla ljudpunkter inom 36 ms till höger och vänster om den utmarkerade stavelsen med. I Figur 2 illustreras hur en ballad styckas upp i stavelser. I den översta grafen så visas hela ljudfilen och grafen under visar vad *strophecut* har identifierat som signifikant ljud. Den mittersta grafen visar hur differensen av de olika filtren kan identifiera områden med ljud. Den fjärde grafen visar varje färg en stavelse i originaldata, och den sista grafen är en förstorad bild av den fjärde grafen.



Figur 2: Grafisk representation av hur utklippning av stavelser går till.

5.2 Ljudvektor som indata

Funktionen *strophecut* används för att stycka upp balladerna till ljudfiler av stavelser. För att ett CNN ska kunna fungera måste indatan vara av samma upplösning. Därför zero-paddas alla returnerade stavelser så att alla blir lika långa som den längsta stavelsen. Därefter sparas de som png-filer, redo att skickas in i ett CNN. Den största stavelsen är 37882 sampel lång vilket motsvarar nästan en hel sekund.

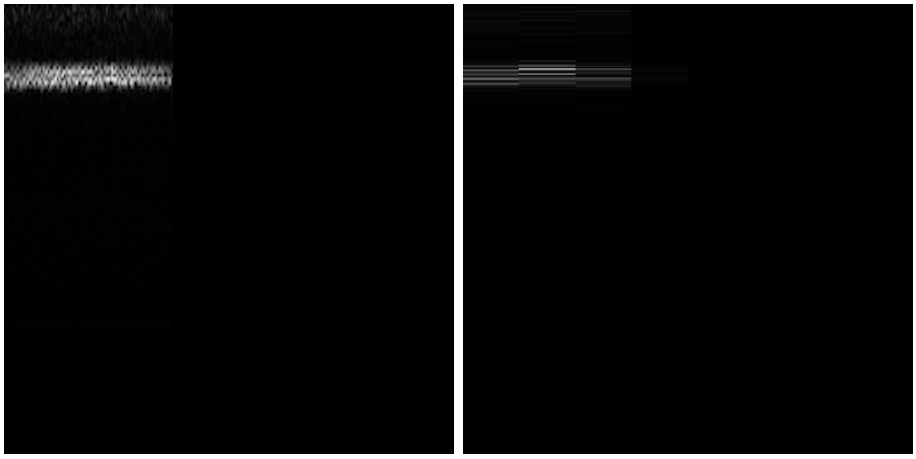
5.3 Spektrogram

Här beskrivs hur spektrogram används som indata till nätverket. Stavelserna omvandlas alltså till spektrogram som används när nätverken tränas.

Spektrogrammen representeras som matriser. Absolutbeloppet av alla element i matrisen normaliseras så att alla de får ett värde mellan noll och ett. Sedan sparas matriserna som bilder i png-format, där varje element är en pixel. Det är dessa bilder som CNN tränas på.

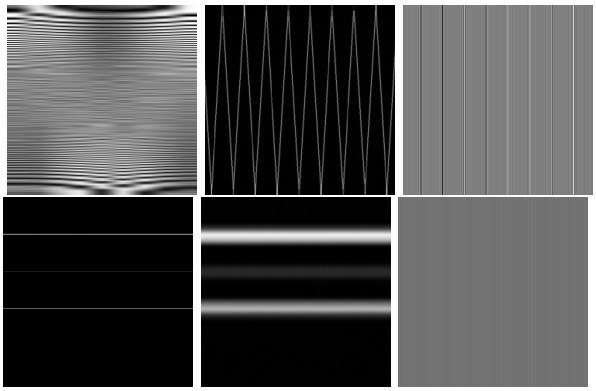
Spektrogrammen har olika upplösning. Dessa skickas in till separata CNN. De två olika spektrogrammen som provas har följande upplösning: Dels LTS med 8x8193 och HTS med 129x256 där siffrorna är $tid \times frekvens$ i antal pixlar. Det första spektrogrammet (LTS) skapades med fönsterlängd 8418 och 50 % överlapp

(8 fönster). Det andra spektrogrammet (HTS) skapades med fönsterlängd 294 och 50 % överlapp (256 fönster). Anledningen till att dessa storlekar väljs är för att ljudvektorn endast har upplösning i tid. Därför väljs ett spektrogram väldigt hög upplösning i frekvens (LTS) och ett spektrogram med ungefär lika hög upplösning i tid som frekvens (HTS). På båda spektrogrammen används Hammingfönster. Detta leder till att antalet datapunkter i respektive spektrogram är 65 544 och 33 024, vilket kan jämföras med de zero-paddade stavelserna på lite under 40 000 samples. I Figur 3 ses de två olika spektrogrammen från en stavelse från en bergfink. Man kan se att bilden till vänster har bättre tidsupplösning medans den till höger har bättre frekvensupplösning. Den helt svarta delen till höger i varje bild är till följd av zero-padding. Bilderna är i svart-vit för att det användes i programmet.



Figur 3: Bilder på spektrogram för en stavelse av en bergfink. Till vänster är HTS och till höger LTS. På x-axeln är det tid och på y-axeln är det frekvens. Frekvenstoppen som ses är vid 6924 Hz.

Samma fönster användes för att skapa spektrogram för analys av de simulerade signalerna. Dessa var då av storlek 1025x9 för LTS och 33x255 för HTS. Detta resulterar i att antalet datapunkter i spektrogrammen är 9225 respektive 8415, vilket kan jämföras med ljudfilens längd som är 8192 datapunkter. Dessa är alltså av motsvarande storlek och därför kan mängden information antas vara jämförbar mellan de olika representationerna av ljudet. Nedan i figur 4 följer bilden av de två spektrogramtyperna samt ljudvektorn datapunkter representerade som bilder. Dessa representationer har skalats om till kvadratiska bilder för att underlätta för läsaren, se Figur 4.



Figur 4: Grafisk representation av simulerade signaler där x- och y-axeln representerar tid respektive frekvens. Överst: Kvitterljud (Klass 1), Nedre: Sinusoid (Klass 3). Från vänster: Spektrogram med hög frekvensupplösning, spektrogram med hög tidsupplösning, den rena ljudsignalen.

5.4 Simulering av fågelsång

För att undersöka om det är rimligt att analysera fågelsång med neurala nätverk undersöks hur väl nätverket kan skilja på olika simulerade ljud. Simuleringsmodellerna följer sex av de åtta sångklasser som definieras i artikeln [1]. Bland dessa sex sångklasser finns två ljud med varierande frekvenser (Kvitter) och fyra ljud med konstanta frekvenser (Sinusoid). Nedan följer ekvationerna för en kvittersignal respektive en sinusoidsignal.

$$y_{kvitter}[n] = e^{(i2\pi(fn+\frac{df}{2}n^2)+\phi)} + \epsilon_{0.5}[n]$$

$$y_{sinusoid}[n] = \sum_{k=1}^{K} a_k e^{(i2\pi f_k n+\phi)} + \epsilon_{0.5}[n]$$

Varje parameter i en simulering är en ny realisering av $N(\mu, \sigma)$ enligt Tabell 1 förutom ϕ som är likformigt fördelat över $[0, 2\pi]$. $\epsilon_{0.5}$ är ett normalfördelat brus, $N(0, 0.5)$, som adderas till de simulerade signalerna.

Tabell 1: Parametervärden för respektive klass av simulerade värden.

Klass	Komponenter	Parametrar
1	1 Kvitter	$f \in N(0.2, 0.01)$ $df \in N(0.001, 0.0001)$
2	1 Kvitter	$f \in N(0.2, 0.01)$ $df \in N(0.0012, 0.0001)$
3	3 Sinusoider	$a_1 \in N(3, 0.005)$ $a_2 \in N(0.5, 0.005)$ $a_3 \in N(2, 0.005)$ $f_1 \in N(0.06, 0.01)$ $f_2 \in N(0.12, 0.01)$ $f_3 \in N(0.18, 0.01)$
4	2 Sinusoider	$a_1 \in N(3, 0.005)$ $a_2 \in N(2, 0.005)$ $f_1 \in N(0.06, 0.01)$ $f_2 \in N(0.12, 0.01)$
5	2 Sinusoider	$a_1 \in N(2, 0.005)$ $a_2 \in N(2, 0.005)$ $f_1 \in N(0.05, 0.06)$ $f_2 \in N(0.051, 0.06)$
6	1 Sinusoid	$a_1 \in N(4, 0.005)$ $f_1 \in N(0.0505, 0.06)$

Inom dessa klasser är ljuden relativt lika, men skiljer sig lite. Alla variabler har en varians och har därför en chans att överlappa varandra. Till exempel har Kvitter-ljuden förändringar i frekvensen (df) som är normalfördelade enligt N(0.001, 0.0001) respektive N(0.0012, 0.0001). Väntevärdet för respektive frekvensförändring är alltså endast två standardavvikelser ifrån varandra. Detta leder till att ett kvitter av den första typen har ca 15% risk att matematiskt vara mer likt ett kvitter av andra typen. Det finns alltså ett inneboende och förväntat överlapp mellan sångklasserna.

Sångklasserna analyserades genom tre media, ett spektrogram med hög frekvensupplösning och låg tidsupplösning, ett spektrogram med låg frekvensupplösning och hög tidsupplösning och som en vektor av den rena ljudfilens värden.

5.5 Fågel- och burkmetoden

För att välja vilken data som skickas in till nätverken används två olika tillvägagångssätt. Det första, burkmetoden, styckar upp alla stavelser, zero-paddar, sparas som HTS, LTS eller ljudvektorer. Sen mäts hur många stavelser den fågeln med lägst antal stavelser har, därefter behålls endast så många stavelser för varje fågel. Sen delas stavelserna upp för träning, validering och test i andelen 70%, 15% och 15% respektive.

Den andra metoden som här kallas fågelmetoden är lite mer komplicerad. Den vill undvika att ett nätverk testas på en stavelse från samma ballad som

en av dess träningsfiler. Detta görs genom att

1. först slumpa ut en ballad från varje fågelart. De balladerna läggs åt sidan och kommer användas som testdata senare.

2. De resterande balladerna används som tränings- och valideringsdata. Detta genom att de styckas upp till stavelser, zero-paddas och sparas som LTS, HTS eller ljudvektorer. Av dessa kommer 80% respektive 20% användas som tränings- respektive valideringsdata.

3. Nätverket tränas med hjälp av tränings- och valideringsdatan.

4. Slutligen ska nätverkets prestanda testas. Detta görs genom att de ballader som tidigare lagts åt sidan som testdata styckas upp till stavelser, zero-paddas och sparas som LTS, HTS eller ljudvektorer. Precis på samma sätt som tränings- och valideringsdatan. Genom att titta på hur nätverket klassar alla stavelser för varje ballad så skattas balladens fågelart till den art som mest stavelser klassas till. Detta görs för alla ballader för att se hur bra metoden är på att klassa arterna.

5.6 CNN

Det neurala närverket implementerades med Matlabs Deep learning toolbox. Strukturen bygger på det neurala nätverket på Matlabs sida om CNN för klassifikation [3]. Detta nätverket är ursprungligen skapat för att klassificera bilder av siffror i gråskala med storleken $28 \cdot 28$ pixlar. De olika lagern i nätverket beskrivs i Tabell 2.

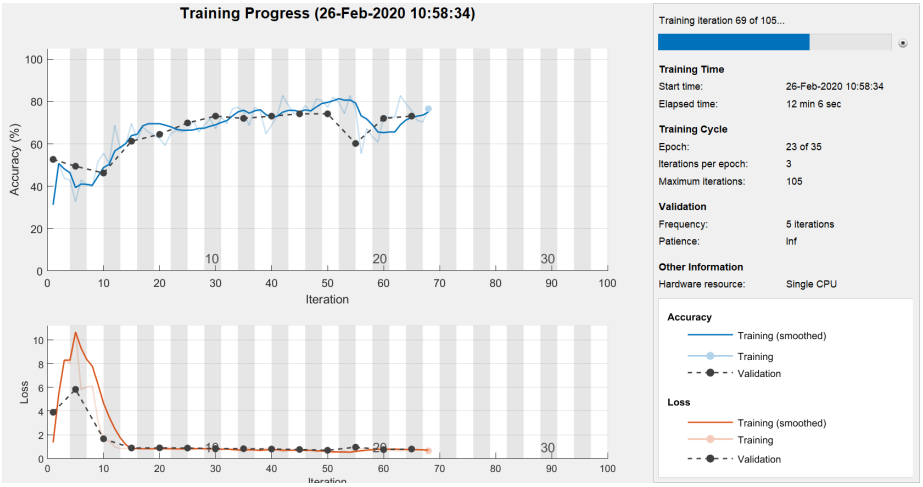
Tabell 2: Lager i det neurala nätverket.

Lager	**Lagerspecifikationer**
imageInputLayer	1 mapp i dimension av input
convolution2dLayer	8 filter av storlek 3x3
batchNormalizationLayer	
reluLayer	
maxPooling2dLayer	poolstorlek 2, stegstorlek 2
convolution2dLayer	16 filter av storlek 3x3
batchNormalizationLayer	
reluLayer	
maxPooling2dLayer	poolstorlek 2, stegstorlek 2
convolution2dLayer	32 filter av storlek 3x3
batchNormalizationLayer	
reluLayer	
fullyConnectedLayer	antal fåglar
softmaxLayer	
classificationLayer	

Det första lagret *imageInputLayer* tar in en matrisrepresentation av bilden som ska analyseras. Därefter följer ett *convolution2dLayer* som är det viktigaste lagret, ett *batchNormalizationLayer* som normaliserar datan och sedan ett *reluLager* som ser till att alla värden är över 0. Lagret *convolution2dLayer* skapar ett antal filter av en viss storlek och faltar resultatet från det tidigare lagret. Detta är lagret som har de flesta parametrarna. Mellan varje grupp av *convolution-normalisation-relu* kommer ett *maxPooling2dLager* som delar upp resultatet från förra lagret i *pooler* av en viss storlek och skickar vidare det största värdet av dessa till nästa lager. Stegstorleken avser hur långa steg den tar mellan varje pool. I vårt fall är poolerna av storlek 2 och det är 2 steg mellan varje pool, alltså är det inget överlapp. Detta är likvärdigt med att dela upp förra lagret i 2x2 rutor och ta maxvärdet av dessa. Efter det kommer ett *fullyConnectedLayer* som kopplar ihop de tidigare filtrena till en vektor som har samma storlek som antalet fåglar. Detta är alltså representation av vilken fågel nätverket anser att indatan hör till. Lagret *softmaxLayer* använder MatLabs *softmax* funktion vilket hjälper matlab att göra en kategorisering av datan. Sist kommer ett *classificationLayer* som returnerar vilken fågel nätverket tror indatan tillhörde.

Nätverket byggde på stokastisk brantaste lutning med momentum (sgdm). Det använde en inlärningshastighet på 0.01, vilket beskriver hur snabbt det ändrar sig. Det validerade med valideringsdatan var femte iteration med valideringsdatan. Det körde alla test i 35 epoker, där varje epok är en genomgång av all träningsdata.

I Figur 5 visas en visuell representation av träning av ett nätverk. I det övre stora fönstret syns hus bra nätverket vid varje tidpunkt gissar på olika stavelser. Den blå linjen är hur bra den gissar på träningsdatan, som den optimerar efter. Den svarta sträckade linjen är hur bra den gissar på valideringsdatan. Det undre fönstret är lite mer abstrakt men visar på hur mycket information nätverket tappar. Där är den röda linjen träningsdata och den sträckade svarta validering. Till höger kan lite allmänna fakta om träningen ses.



Figur 5: Visuell representation av en träning av det neurala nätverket.

6 Resultat

I nedanstående resultat presenteras hur bra träffsäkerhet ett CNN kan ha beroende om den tränar på tre fåglar eller sex fåglar och för olika indata för dessa. De tre fåglarna som tränas på för sig är talgoxe, bofink och gråsparv och de tre resterande fåglar är bergfink, blåmes och pilfink. För alla kategorier tränades 20 stycken CNN, i 35 epoker med learningRate 0.01. Dessutom användes både burk- och fågelmetoden.

6.1 Simuleringar

Resultatet från klassificeringen av de simulerade klasserna presenteras nedan. I Tabell 3 visas medelvärdet för träffsäkerheten, inom respektive klasstyp samt för alla klasser, för de 20 träningarna av nätverket, samt standardavvikelsen för alla klasser. Då resultatet från analysen av stavelserna och spektrogrammet med hög tidsupplösning är så pass lika görs även en analys på skillnaden av resulatet för respektive träning. Denna skillnad har då medelvärdet 0.0183 och standardavvikelsen 0.0978.

Tabell 3: Medelvärdet för träffsäkerheten över 20 träningar av nätverket för de tre olika datatyperna.

Datatyp	**Ljudvektorer**	**LTS**	**HTS**
Kvitterklasser μ	0.8343	0.7683	0.8478
Sinusoidklasser μ	0.9573	0.4059	0.9780
Totalt μ	0.9163	0.5267	0.9346
Totalt σ	0.0980	0.1296	0.0081

6.2 Tre fåglar

I den första delen av resultatet presenteras värden för tre fåglar.

6.2.1 Burkmetoden

Först tränades olika CNN, enligt burkmetoden för tre olika fåglar.

Tabell 4 visar medelvärdet samt minsta och största värde av träffsäkerheten för 20 träningstillfällen av ljudvektorer, LTS och HTS för tre fåglar med burkmetoden.

Tabell 4: Tabellen visar resultatet av 20 träningstillfällen av olika CNN efter 35 epoker, med learningRate=0.01, av tre fåglar med ljudvektorer, LTS och HTS som indata. Här används burkmetoden.

Indata	**n**	**avg (%)**	**max (%)**	**min (%)**
Ljudvektorer	20	51.61	74.19	33.33
LTS	20	89.19	99.77	82.80
HTS	20	96.29	100.00	92.47

6.2.2 Fågelmetoden

Sen tränades olika CNN, enligt fågelmetoden för tre olika fåglar.

Tabell 5 visar medelvärdet samt minsta och största värde av träffsäkerheten för 20 träningstillfällen av ljudvektorer, LTS och HTS för tre fåglar med fågelmetoden.

Tabell 5: Tabellen visar resultatet av 20 träningstillfällen av olika CNN efter 35 epoker, med learningRate=0.01, av tre fåglar med ljudvektorer, LTS och HTS som indata. Här används fågelmetoden.

Indata	**n**	**avg (%)**	**max (%)**	**min (%)**
Ljudvektorer	20	53.67	87.38	23.88
LTS	20	55.88	89.72	23.97
HTS	20	56.77	95.10	10.73

6.3 Sex fåglar

I den andra delen av resultatet presenteras endast värden för sex fåglar.

6.3.1 Burkmetoden

Först tränades olika CNN, enligt burkmetoden för sex olika fåglar.

Tabell 6 visar medelvärdet samt minsta och största värde av träffsäkerheten för 20 träningstillfällen av ljudvektorer, LTS och HTS för sex fåglar med burkmetoden.

Tabell 6: Tabellen visar resultatet av 20 träningstillfällen av olika CNN efter 35 epoker, med learningRate=0.01, av sex fåglar med ljudvektorer, LTS och HTS som indata. Här används burkmetoden.

Indata	**n**	**avg (%)**	**max (%)**	**min (%)**
Ljudvektorer	20	38.61	64.81	16.67
LTS	20	71.39	87.04	59.26
HTS	20	84.44	96.30	66.67

6.3.2 Fågelmetoden

Sen tränades olika CNN, enligt fågelmetoden för sex olika fåglar.

Tabell 7 visar medelvärdet samt minsta och största värde av träffsäkerheten för 20 träningstillfällen av ljudvektorer, LTS och HTS för sex fåglar med fågelmetoden.

Tabell 7: Tabellen visar resultatet av 20 träningstillfällen av olika CNN efter 35 epoker, med learningRate=0.01, av sex fåglar med ljudvektorer, LTS och HTS som indata. Här används fågelmetoden.

Indata	**n**	**avg (%)**	**max (%)**	**min (%)**
Ljudvektorer	20	25.38	54.92	4.34
LTS	20	41.74	77.78	11.80
HTS	20	60.77	79.45	23.62

7 Utvärdering/Diskussion

7.1 Simuleringar

Resultatet från de simulerade ljudfilerna visar att LTS var betydligt sämre än de två andra metoderna. Både ljudvektorerna och HTS gav liknande resultat och vi kan inte avgöra om det existerar en skillnad. Det kan konstateras att båda dessa ligger runt 85% träffsäkerhet om man jämför resultatet mellan de två kvitterklasserna. Som konstaterat i Implementationen är det ett förväntat resultat då dessa två klassers avgörande parameter endast ligger två standardavvikelser

ifrån varandra. Detta leder till att en parameter hos Klass 1 har ca 15% risk att ligga närmare väntevärdet från samma parameter hos Klass 2. Alltså kommer ca 15% av signalerna av Klass 1 vara oskiljaktiga från en signal från Klass 2 och vice versa. En träffsäkerhet på 85% är alltså det teoretiska maximum. Vi ser då att både träningen på ljudvektorer och HTS ger ett resultat som ligger nära detta teoretiska maximum.

7.2 Skillnaden mellan burkmetoden och fågelmetoden

Tabellerna i resultat visar att burkmetoden ger ett bättre resultat än fågelmetoden för alla tester utom ett, ljudvektorer för tre fåglar. Det är värt att påpeka att fågelmetoden alltid ger att största/minsta träffsäkerhetens avvikelse från medel är större än hos burkmetoden. Faktum är att fågelmetoden kan ge resultat som är sämre än slumpen, vilket inte är önskvärt. En anledning till detta kan vara att varje art enbart har ca 10 individer att träna på. Resultatet kan då vara mycket beroende på vilken individ som inte tillhör träningsdata i fågelmetoden.

Anledningen till att burkmetoden presterade så bra är troligen att nätverket då lärde sig om bakgrundsljud i de olika ljudfilerna. Då den tränar på fåglar från samma inspelning som den testar på så verkar det troligt att den lär sig om inspelningsutrustningen istället för själva stavelserna. När man faktiskt testar prestanda bör man bara testa på helt ny data, vilket motsvarar det vi gjorde i fågelmetoden. Alltså visar detta på att man bör tänka sig för hur man faktiskt väljer ut sin data.

7.3 Skillnad mellan ljudvektorer och spektrogram

Resultaten visar genomgående att det är sämre att skicka in ljudvektorer som resultat i jämförelse med HTS. I simuleringarna presterar dock ljudvektorn mycket bättre än LTS och är bara några procentenheter sämre än HTS. För fåglarna är det lite mer spritt men för alla metoder utom fågelmetoden för tre fåglar presterar ljudvektorerna mycket sämre än båda spektrogrammen. Alltså verkar det finnas någon inneboende aspekt av de riktiga stavelserna respektive de simulerade som gör att resultaten blir så olika. Kanske är det att de simulerade ljudfilerna var mycket renare och enklare än de riktiga balladerna.

Dessutom hörde vi från vår handledare Maria Sandsten att det är vanligt att bara skicka in obehandlad data, som ljudvektorn till nätverk, då det förväntas ge bättre resultat. Här visas det ganska tydligt att det finns anledning att omvandla data till spektrogram i vissa tillfällen när man jobbar med neurala nätverk.

7.4 Skillnaden mellan höga/låg- upplösta spektrogram

Vi ser i tabellerna i resultatet att vi får bättre träffsäkerhet för HTS än för LTS. Anledningen för detta är att man får mer information ur HTS. Vad vi menar är att i det spektrogrammet finns det fortfarande en bra frekvensupplösning,

129x256. Så CNN:et har information i både tids- och frekvensdomänen jämfört med det andra spektrogrammet som bara har bra frekvensupplösning, 8x8193.

7.5 Skillnaden mellan olika antal fåglar

Att resultatet för tre fåglar ska vara bättre än det för sex fåglar är väntat, vilket stämmer i alla fall förutom ett, HTS i fågelmetoden (56.77% mot 60.77%). Varför detta är fallet vet vi inte, slumpen är en möjlig förklaring.

7.6 Slutsats

Den metod som fungerade bäst, i alla kategorier var HTS. Denna metod fungerade även bra för den simulerade signalen. Det intressanta är att ljudvektorn fungerade bra och LTS fungerade dåligt för den simulerade signalen. Det var dock tvärtom då en verklig ballad skulle analyseras. Detta kan vara på grund av att det fanns fler störningar och bakgrundsljud i balladen som inte fanns i den simulerade signalen. Dessa störningar syns väl i ljudfilen men syns mindre bra i spektrogrammet. Detta kan vara eftersom spektrogrammet normaliseras linjärt till ett värde mellan 0 och 1. Då ljudstyrka sker exponentiellt kommer därför starka ljud synas tydligt medan svaga toner antar värden nära 0. Detta skulle vara en rimlig förklaring till varför LTS fungerar väl till de riktiga balladerna medan ljudvektorn fungerar bättre för simulerade signalen.

7.7 Hur man skulle kunna fortsätta arbetet, och felkällor

I detta arbete tränades de neurala nätverken endast på enskilda stavelser. Det hade varit intressant att testa samma metoder på hela ballader och se om resultaten skiljer sig. Eller att välja andra sätt att stycka upp ljudfilen, eller andra metoder för frekvensanalys.

Ett annat sätt att arbeta vidare på det som gjorts här är att vidareutveckla våra metoder. Bland annat kan metoden *strophecut* göras mer träffsäker. Med det menas att den inte alltid klipper ut det som man som människa hade sagt var enskilda stavelser. Om det exempelvis finns flera väldigt korta stavelser nära varandra kan funktionen ibland tro att det är en stavelse, det syns exempelvis i Figur 2 på sista bilden, där flera stavelser har klassats som en. Sen finns även svårigheten åt andra hållet också. Om en stavelse är väldigt lång så kan ibland inte hela plockas ut för att det stora filtret också täcker hela stavelsen, vilket gör att differensen inte blir signifikant. Det hade varit intressant att implementera *strophecut* på ett annat sätt än att ha en fast längd för filtrena. Svårigheten är att det är svårt att ta fram en träffsäker definition med siffror på vad en stavelse är.

Arbetet kan även utvecklas vidare genom att ta fram ett mer specifikt anpassat neuralt nätverk för dessa problemen. Det som används här är anpassat för mycket mindre bilder så det borde gå att anpassa det bättre för våra omständigheter. Det kan bland annat vara en anledning till att nätverket är så

dåligt på att klassificera stavelserna som ljudvektorer, då nätverket är byggda för bilder och inte vektorer.

Dessutom har detta arbete använt sig av relativt lite data. Då huvudsyftet inte är att få perfekta uppskattningar, utan snarare att jämföra metoder är det av mindre vikt. Men om bättre resultat vill uppnås hade det varit intressant att söka upp mer data. Exempelvis har bergfinken endast 56 stavelser, så trots att talgoxe och blåmes båda har över 500 stavelser kan vi endast använda 56 av dem. Detta var eftersom man måste använda lika många stavelser från varje art. Det hade därför hjälpt att öka mängden stavelser, speciellt från bergfink då det är den art som har minst antal stavelser.

Referenser

[1] J. Brynolfsson, M. Sandsten, 2017, *Classification of One-Dimensional Non-Stationary Signals using the Wigner-Ville Distribution in Convolutional Neural Networks*, EUSIPCO.

[2] M. Große Ruse, D. Hasselquist, B. Hansson, M. Tarka och M. Sandsten, *Automated analysis of song structure incomplex birdsongs*, Animal Behaviour 112, 2016.

[3] MathWorks, *Create Simple Deep Learning Network for Classification*, se.mathworks.com/help/deeplearning/examples/create-simple-deep-learning-network-for-classification, hämtad februari 2020.

[4] MathWorks, *Convolutional Neural Network*, https://se.mathworks.com/solutions/deep-learning/convolutional-neural-network, hämtad mars 2020.

[5] MathWokds, *Spectrogram*, se.mathworks.com/help/signal/ref/spectrogram.html, hämtad februari 2020

Distribution av arbete

Vår handledare, Maria Sandsten, vägledde oss genom att ge oss många exempel på hur man kan lösa liknande problemet. När vi sedan valde en metod hjälpte hon oss vidare. Inom gruppen antog vi olika arbetsuppgifter. Emil testade CNN, och anpassade olika variabler så att den bättre passade vårt syfte. Maja läste bakgrundslitteratur som behandlade samma problem om fågeligenkänning och arbetade med rapporten. Kåre utvecklade metoden *strophecut* och skapade den slutgiltiga koden för att träna nätverken. Oskar jobbade med skapandet av och träningen på simulerade ljudfiler samt parametrarna hos spektrogrammen. Alla har även arbetat med rapporten. Allt som beskrivits ovan har inte strikt gjorts enskilt, alla i gruppen har hjälpts åt för att komma framåt i arbetet.