KØBENHAVNS UNIVERSITET

Genkendelse af talt sprog med convolutional neural network

af

Aslak Niclasen | rql599

Jonathan Kurish | vgp479

Bachelor i datalogi

Datalogisk Institut

13. Juni 2016

Resume

Genkendelsen af talt sprog via lydfiler er et kendt problem, og der findes mange forskellige måder at løse dette problem på. Det er stadig uklart hvordan problemet løses bedst. Vi har valgt at implementere et Convolutional Neural Network (CNN) ved hjælp af et open source framework kaldet Tensorflow, og benyttet en bestemt type feature extraction, kaldet Mel Frequency Cepstral Coefficients (MFCC). Heraf ønsker vi at undersøge, hvilke parametre, der har betydning for algoritmens evne til at opnå den højest mulige accuracy. Resultaterne viser, at et CNN med øget antal lag og MFCC-features opnår højere accuracy. Ved at benytte denne viden lykkedes det os at opnå accuracy på op til 75% ud af 24 sprog på et datasæt udgivet af TopCoder.

In English

The recognition of speech using audio files is a known problem and many different approaches to solve this problem exist. It is unclear which approach performs best. We have chosen to implement a convolutional neural network (CNN) using an open source framework called Tensorflow as well as using a particular type of feature extration, called Mel Frequency Cepstral Coefficients (MFCC). We wish to investigate which parameters influence the effectiveness of the algorithms ability to achieve the highest possible accuracy. Results show that a CNN with an increased number of layers and MFCC-features achieve a higher accuracy. By applying this knowledge, we achieved accuracies of nearly 75% on a set of 24 languages on a dataset released by TopCoder.

Indhold

Figurliste

Tabelliste

Kapitel 1

Introduktion

Vi forsøger at løse et talegenkendelsesproblem, helt præcist genkendelsen af talt sprog via lydfiler. Maskinlæring er en metode til at løse dette problem. Med brugen af et open source framework kaldet Tensorflow udviklet af Google, implementerer vi et Convolutional Neural Network (CNN) til at løse talegenkendelsesproblemet. Input til den foreslåede løsning benytter elementer af en lydfil, kaldet mel frequency cepstrum coefficients (MFCC), der er en modificeret repræsentation af et effektspektrum, mens output er en sprogklassificering af den originale lydfil. Nærmere betegnet er input en vektor af reelle tal og output et reelt tal mellem 0 og 1.
Netværket bliver trænet og derefter testet med et tilpasset datasæt, udgivet af organisationen TopCoder. Det fulde datasæt består af i alt 176 sprog, hvor vi benytter 3, 6 eller 24 sprog, bl.a. vietnamesisk, polsk og hollandsk. Der benyttes forskellige lydfiler til henholdsvis træning og testning af netværket, og dermed er testresultaterne udelukkende baseret på uset data. Netværket bliver ligeledes tilpasset i de forskellige test, der udføres. Både i netværkets kompleksitet og gennem andre parametre, som f.eks. batch- og pooling-størrelse, forsøger vi at opnå den bedst mulige accuracy indenfor de begrænsninger, der sættes for netværket. Accuracy måles som antallet af korrekte klassificeringer divideret med det totale antal lydfiler i testsættet. Desuden forsøger vi med løsningen at undgå en kendt faldgrube ved neurale netværk, kaldet overfitting. Til dette bruges to teknikker, den ene kaldet dropout-teknikken og den anden kaldet early stopping.

1.1 Relateret arbejde

Det er ikke første gang, at et problem af lignende karakter er ønsket løst. Vi har vha. Google Scholar [1] søgt på kriterierne *spoken language identification*, *neural network* eller

en kombination heraf og fundet nedenstående fire lignende problemer. Alle problemerne omhandler klassificering af lyd og er udvalgt på baggrund af deres relevans. Både med hensyn til problemstillingen og de foreslåede løsninger. Tre af de nedenstående fire forsøg beskæftiger sig alle med samme type af klassificeringsproblem, klassificering af talt sprog via lydfiler. Det sidste forsøg adskiller sig fra disse. I dette forsøg ønskes musikgenrer klassificeret.

I 2012 udførte [2] et sproggenkendelsesforsøg med forskellige maskinlæringsalgoritmer til genkendelse af ét af to valgte sprog. Der blev testet på seks forskellige sprog med lydklip af varighed på fire til syv sekunder. Resultaterne viste, at brugen af neuralt netværk gav bedst resultat ift. brugen af support vector machines eller gaussian mixture models.

I 2009 implementerede [3] et time delayed neural network (TDNN) til at klassificere engelsk, tysk og fransk fra hinanden. Det blev implementeret i to forskellige versioner, hvoraf den ene havde flere skjulte lag. De to implementeringer blev testet mod hinanden på to forskellige datasæt. Det ene datasæt bestod af lydfiler af 5 sekunders varighed, med varierende kvalitet. Det andet datasæt bestod af forskellige radiooptagelser, af høj kvalitet. Resultaterne viste, at et TDNN med en implementering af flere skjulte lag var bedre til at klassificere talt sprog.

I 2012 udførte [4] et forsøg med convolutional neural network til klassificering af musikgenrer. Datasættet bestod af 100 lydfiler af sange inden for 10 musikgenrer, alle med en varighed af 30 sekunder. De fandt deres implementering af et convolutional neural network til at være bedre end andre typer af neurale netværk, der forsøgte at gøre det samme.

I 2014 udførte [5] et studie om brugen af deep neural networks (DNN) til automatisk sprogklassifikation. Resultaterne blev sammenlignet med i-vector systemer på to forskellige datasæt, Google 5M LID Corpus og NIST LRE 2009. Google-datasættet bestående af lydsekvenser fra Googles sproggenkendelsesservices med gennemsnitslængde 2,1 sekunder fra 25 sprog i 9 dialekter og datasættet fra NIST bestående af over 200 timers radioudsendt tale fra hvert af 8 forskellige sprog, resulterende i et samlet datasæt på omkring 25000 lydsekvenser. Resultaterne fra studiet viste, at DNN'er med enten 2 eller 8 skjulte lag gav bedre resultater end i-vector systemer, hvorimod et DNN med 8 skjulte lag gav de bedste resultater, dog uden at være markant bedre ifølge forfatterne.

1.2 Forventninger til læseren

Der stilles forventninger til læseren om kendskab til diskret matematik, lineær algebra [6] og grundlæggende integral- og differentialregning [7]. Endvidere stilles der forventninger til læseren om kendskab til dataanalyse [8]. Der lægges især vægt på kendskab til convolutional neural networks [9] (kapitel 6), hvilket løsningen baserer sig på.

Kapitel 2

Maskinlæring

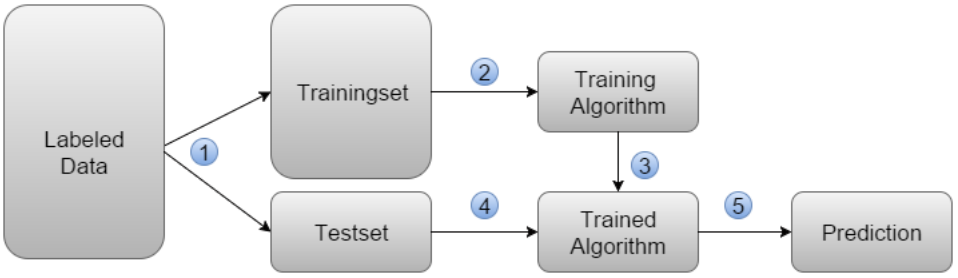
Maskinlæring er studiet om algoritmer, som lærer at genkende og forudsige mønstre vha. data [10]. En maskinlæringsalgoritme lærer ved gentagne gange at bearbejde ny data og derefter forbedre sin præstation. Maskinlæring anvendes inden for mange områder fra simple opgaver som klassificering af billeder af håndskrevne cifre [11] til mere avancerede opgaver, som styring af biler uden en fører bag rattet. [12]. Der eksisterer forskellige typer af maskinlæring, hvoraf et udpluk af disse vil blive introduceret kort herunder.

2.1 Supervised learning

Supervised learning er en type maskinlæring, hvor en mængde af data med kendt input og output anvendes til at lære en algoritme at forudsige output for uset data [8]. Outputtet kaldes labels. Genkendelsen af håndskrevne cifre er et eksempel på supervised learning, fordi data ønskes placeret inden for endeligt mange kendte kategorier. Dette kaldes et klassificeringsproblem. Der findes også andre former for supervised learning, eksempelvis regression [13].

For alle de forskellige metoder af klassificering deles dataen op fra start. En måde at gøre dette på er via krydsvalidering. Krydsvalidering bruges ofte til at træne og evaluere algoritmen i maskinlæring. Det er et vigtigt indledende punkt i en klassificeringsprocess, da dataen direkte deles op i hhv. træningssæt og testsæt. Et træningssæt er den mængde af data, der bruges til træning af algoritmen. Et testsæt er den resterende mængde og benyttes til at teste algoritmens evne til at klassificere korrekt. Følgende er to af de standardiserede måder at krydsvalidere på. Procentvis opdeling af datasættet, eksempelvis 70% til træning, og 30% til test og S-fold metoden. I S-fold metoden deles dataen op i S lige store grupper. Den S'te gruppe vil fungere som testsæt, mens alle

andre grupper fungerer som træningssæt. Dette gøres for alle S, og der opnås deraf S forskellige scorer for hver klassificering. Herefter tages den gennemsnitlige score for S, og datasættet siges at være klassificeret. I figur 2.1 ses ideen bag supervised learning ved brug af procentvis opdelt krydsvalidering.



FIGUR 2.1:

Illustration af hvordan supervised learning fungerer. Data deles op i et trænings- og testsæt. Træningssættet bruges til at lære. Herefter bruges testsættet til at forudsige på baggrund af træningssættets læren.

2.2 Unsupervised learning

Unsupervised learning er en type maskinlæring, hvor en mængde af data med kendt input og ukendt output anvendes til at lære en algoritme at forudsige output for uset data [8]. Den adskiller sig fra supervised learning ved, at der ikke er nogen kendte outputs (eller labels), i stedet vil dataen blive forsøgt opdelt i eksempelvis klynger, såkaldte clusters. En klynge grupperer data med ensartede karaktertræk [8]. En konsekvens af de manglende labels i træningssættet medfører, at effektiviteten ikke kan evalueres af en given algoritmes resultater på samme måde som ved supervised learning.

2.3 Neural network

Et neural network, oversat neuralt netværk, er en orienteret graf af sammenhængende knuder, placeret i en række lag. Det første lag af knuder svarer til algoritmens input, mens det sidste lag af knuder svarer til output. Lagene af knuder imellem kaldes skjulte lag. Disse lag ændres løbende via træningen, således at algoritmen bliver bedre til at klassificere. Det sker, fordi knuderne er forbundet igennem grafen med vægtede kanter. Det er i praksis disse kanters værdier som ændres, når algoritmen lærer at klassificere. Neural networks har vist sig at være gode til at løse klassificeringsproblemer inden for emner som talegenkendelse [11]. Ideen til denne maskinlæringsteknik kommer fra menneskets hjernes struktur. En måde at forestille sig dette på er, at grafen repræsenterer et

sammenhængende system af neuroner, der alle arbejder sammen om at løse et specifikt problem via en fælles indsats. Neuronerne hjælper hinanden ved at være forbundet af synapser i hjernen, hvilket er årsagen til, at grafens vægtede kanter siges at repræsentere synapser. Et neural network lærer vha. eksempler i modsætning til konventionelle computerprogrammer, der kun kan følge en specifik algoritme og dermed udføre en allerede kendt sekvens af instruktioner. Dette giver et neural network mulighed for at løse problemer, som det ikke allerede kender. Figur 2.2 er en illustration af, hvordan et neural network kan se ud.

Langt de fleste neural networks lærer gennem en teknik kaldet backwards propagation. Algoritmen, som benyttes til denne teknik, er beskrevet ved formel 2.1 - 2.4 [9]. Overordnet set er formel 2.1 dermed en udregning af fejlen i outputlaget, hvor fejlen er et reelt tal, som har indflydelse på, hvor godt og hvor hurtigt netværket lærer. Formel 2.2 er en udregning af fejlen i det l'te lag ift. det næste lags fejl. Formel 2.3 er en udregning for ændringen af fejlen af funktionen der forsøges minimeret, ift ethvert bias og formel 2.4 en udregning for ændringen af fejlen af funktionen der forsøges minimeret, ift enhver vægt.

$$\delta^L = \bigtriangledown_a C \odot \sigma'(z^L) \tag{2.1}$$

Hvor δ^L betegner fejlen i output laget, C er den funktion vi ønsker at minimere, $\bigtriangledown_a$ er en vektor med partielt afledte af C, ift. aktiveringsfunktionen a, $\sigma'(z^L)$ er en faktor der måler aktiveringsfunktionen ædrning ift. Z, hvor Z er det vægtede input i lag l.

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{2.2}$$

Hvor δ^l er fejlen i det pågældende lag, w^{l+1T} er vægtmatricen for det næste input lag, δ^{l+1} er fejlen i det næste input lag, $\sigma(z^l)$ er faktoren der måler aktiveringsfunktionens ændring i det nuværende lag, og $\odot$ betegner vektoroperationen for Hadamardproduktet A.

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{2.3}$$

Hvor $\frac{\partial C}{\partial b^l_j}$ er den partielt afledte af en funktion, C der ønskes at minimeret ift. et bias b, og δ^l_j er fejlen i den pågældende knude j i det samme lag l.

$$\frac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1}\delta_j^l \tag{2.4}$$

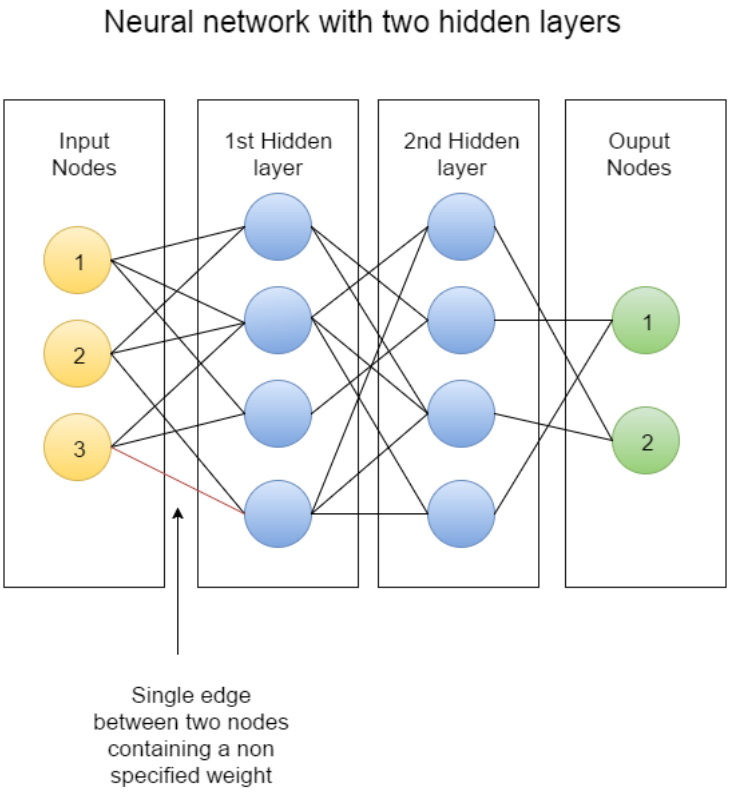
Hvor $\frac{\partial C}{\partial w_{jk}^{l}}$ er den partielt afledte af en funktion C der ønskes minimeret ift. til en vægt w, a_k^{l-1} er aktiveringsfunktionen i det tidligere lag, og δ_j^l er fejlen i det nuværende lag i den nuværende knude.

Denne process forsøger dermed at formindske den endelige fejl mest muligt, hvilket dermed også betyder, at der skal være et kendskab til outputtet. Backwards propagation 'sender' fejlen for den enkelte knude i det næste lag tilbage til det forrige lag. Herved kan de vægtede kanter imellem knuderne opdateres forskelligt, sådan at fejlraten bliver mindre. Med backwards propagation ændres en vægt mere mellem de knuder, hvor fejlraten er størst. Til at beregne fejlen for backwards propagation benyttes en metode kaldet gradient descent. Gradient descent udnytter, at den afledte funktions globale minimum findes. Ved brugen af denne minimeres fejlen derfor også mest muligt. Efter backwards propagation med gradient descent opnås en klassificering af dataene, som vores CNN også baseres på [11].

2.4 Deep learning

Deep learning er en maskinlæringsteknik, som tager afsæt i neural networks. Teknikken involverer, at der laves abstrakte repræsentationer af den data, der ønskes klassificeret [10]. Dette skyldes, at en anden repræsentation af dataen kan gøre det nemmere for algoritmen at lære alt efter det problem, der skal løses. En ofte benyttet måde til at skabe abstraktion er vha. autoencoders. En autoencoder har til formål at rekonstruere dataen, men i en anden repræsentation. Denne anden repræsentation har ofte en lavere dimensionalitet end den oprindelige data. Det betyder, at der i den nye repræsentation tabes information. Autoencoderen har til gengæld den styrke, at dataene nu ikke behøver være labeled. Da den forsøger at lave en rekonstruktion af de oprindelige data, behøver den ikke kende til deres labels og kan derfor benyttes til unsupervised learning.

Deep learning er derfor også en teknik, som bruger neural networks til at klassificere data, men gennem mere abstrakte datarepræsentationer. Navnet Deep learning kommer af, at der ofte er mange skjulte lag i netværket, så mange at dataen skal repræsenteres anderledes for at kunne klassificere med en lav fejlrate.



FIGUR 2.2:

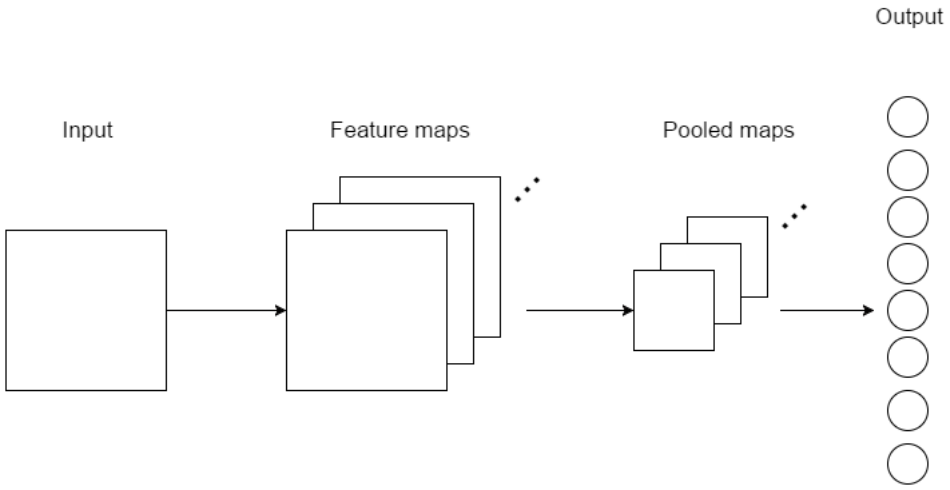
Illustration af hvordan et neural network fungerer. Der findes i starten et inputlag af knuder, herefter følger et eller flere skjulte lag og til sidst findes et output lag, som er resultatet af netværkets klassificering. Kanterne imellem knuderne kaldes ofte for synapser. Alle kanter har en vægt tilknyttet. Disse vægte justeres løbende, for at opnå den bedste klassificering af dataen. Opdateringen af vægtene sker via backwards propagation.

2.5 Convolutional neural network

Et convolutional neural network (CNN) indeholder ligesom andre typer af neural networks et antal lag. Forskellen er, at et eller flere af disse lag er convolution lag.

Et convolutionlag benytter en funktion, som kan fortolkes som et filter. Dette filter transformerer inputtet og uddrager information i dataen, der benyttes til at udnytte inputets struktur. Et input af spatial struktur medfører, at netværket kan have flere lag, uden at antallet af parametre i træningsprocessen forøges markant som f.eks. ved et feed forward neural network. Dette gøres, fordi det formodes, at en relevant feature for inputtet i en specifik spatial position også er vigtig i andre spatiale positioner [14]. Det kunne f.eks. være en skarp kort tone i et bestemt frekvensniveau i en lydfil. Komplekse features i dataen kan derfor identificeres ved at kombinere informationen fra flere convolutionlag. Denne transformation returnerer et feature map. Derefter påføres ofte

en ekstra transformation, som formindsker mængden af data, der arbejdes med, inden outputlaget, via en teknik kaldet subsampling. Dette lag kaldes ofte et pooling lag. En illustration af et CNN kan findes i figur 2.3. Et CNN lærer på samme måde som et ordinært neural network, via gradient descent og opdaterer løbende sine vægte via backpropagation.



FIGUR 2.3:

Illustration af hvordan et Convolutional neural network fungerer. Der findes i starten et inputlag af data, en mængde af knuder, herefter følger et eller flere tranformerede lag af knuder. Enten via feature maps eller pooling lag. Til sidst findes et outputlag. Fra poolinglaget til output laget er alle knuder forbundet til alle outputknuder, hvorimod der er en, én til én korrespondance imellem de to transformerede lags knuder. Illustrationen baserer sig på en lignende figur fra [9], kapitel 6.

2.6 Pooling

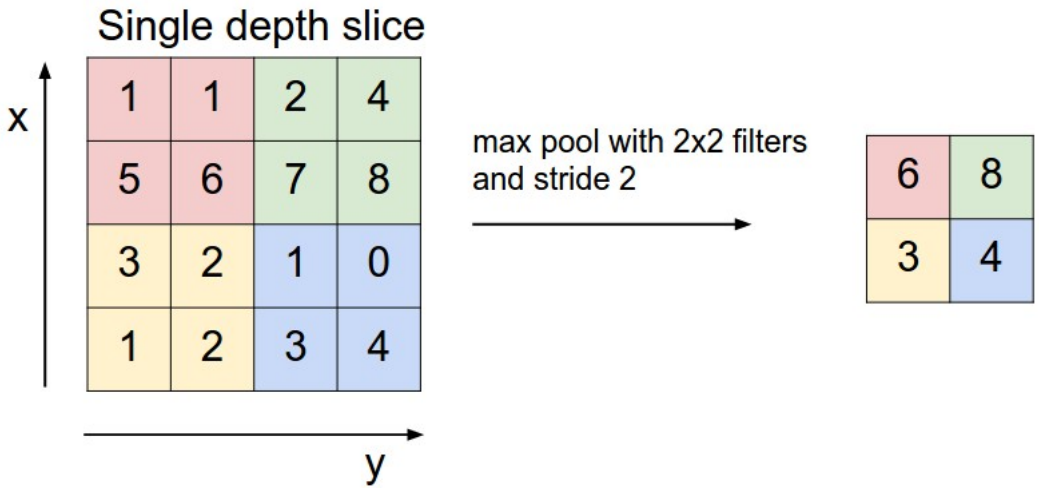
Pooling, også kaldet subsampling, er en fremgangsmåde, som reducerer mængden af data-dimensionaliteten, ved at opdele inputtet i mindre regioner langs hver dimension og afbilde inputtet med en funktion. Denne funktion tager som input en liste af tallene i hver region af billedet og returnerer en enkelt værdi. Ved at reducere dimensionaliteten formindskes den krævne regnekraft til træning i det benyttede neural network. Derudover medfører pooling, at features bliver mindre sensitive over for den præcise struktur i input-billedet og dermed reduceres overfitting, som beskrevet længere nede. Det kan siges, at billedet udglattes [15].

Ofte bruges enten gennemsnittet af inputværdierne, kaldet mean-pooling, eller den

største af inputværdierne, kaldet max-pooling. I figur 2.4 ses et eksempel på MAX-pooling. Definitionen for MAX-Pooling er:

$$s_j = \max_{i \in R_j} a_i \tag{2.5}$$

Her betegner a_i input, R_j er den lokale region, hvori den i'te værdi befinder sig, og s_j er afbildningen for den pågældende værdi i i regionen R_j. Et eksempel på MAX-pooling kan ses i figur 2.4.



FIGUR 2.4:

Eksempel på MAX-pooling. En simpel afbildning fra en mængde af 4x4 inputværdier til en mængde af 2x2 outputværdier. Her er der 4 lokale regioner. Den højeste værdi fra hver region vælges. Taget fra [14]. Besøgt 15-05-2016.

2.7 Softmax regression

Softmax regression som gives et input x og returnerer et reelt tal i intervallet [0;1]. Givet en softmaxfunktion, s, vil $s : x \in \mathbb{R} \rightarrow [0; 1]$ [9]. Før normalisering af dataen er mulig med en softmax funktion, udregnes en værdi kaldet evidensen. Evidensen er en sum af vores inputs individuelle værdier, ganget med en vægt for en klasse samt en bias.

Evidensen er defineret ved:

$$\text{Evidens}_i(x) = \sum_j W_{i,j} x_j + b_i \qquad W, b \in \mathbb{R} \tag{2.6}$$

Her betegner W_i vægtene og b_i er bias for den mulige klasse i, x er inputtet, som vi summerer over med j [16]. Store værdier af evidensen for en klasse i er dermed et udtryk for at det pågældende input har meget tilfælles med klassen. Efter evidensen er

udregnet påføres en softmax-funktion, der transformerer disse værdier til tal i intervallet [0;1]. Softmax funktionen er defineret ved:

$$\text{softmax}(x) = \text{normalize}(exp(x)) \tag{2.7}$$

, Her betegner x input til funktionen, den tidligere udregnede evidens, en vektor af reelle tal. `exp()` er eksponentialfunktionen og `normalize()` er en normaliseringsfunktion, som transformerer `exp()` til et reelt tal mellem 0 og 1.

2.8 Cross entropy

Cross entropy måler en afstand mellem to sandsynlighedsfordelinger. Denne afstand benyttes bl.a. ved loss-funktioner, som er betegnelsen for en funktion, der udtrykker, hvor meget en forudset fordeling afviger fra testsættets faktiske klassificeringer [17]. I vores tilfælde ønsker vi at minimere denne loss-funktion således, at vi bedst klassificerer vores testsæt. Definitionen af Cross Entropy er:

$$H_{y'}(y) = -\sum_i y_i' log(y_i) \tag{2.8}$$

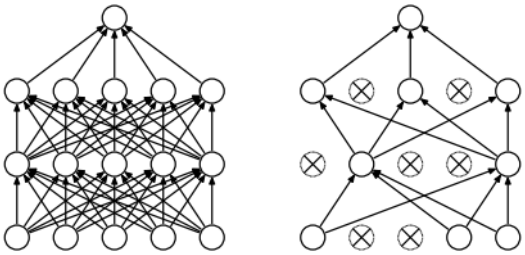
Her betegner y den forudsete fordeling, y_i' er den faktiske sandsynlighed for hver klassificering for hvert element (altså tallet 1 ved den korrekte klassificering og 0 ved alle andre mulige klassificeringer) og y_i er den trænede sandsynlighed for, at hver klassificering for hvert element er klassificeret korrekt.

2.9 Overfitting

Overfitting er et udtryk, der benyttes til at beskrive, at en funktion i for høj grad tilpasser sig et begrænset sæt af datapunkter [18]. Dette kan eksempelvis forekomme i MAX-pooling.

Dropout er en teknik, der fungerer ved, at der ved hvert nyt batch, som betegner en delmængde af datasættet brugt for hvert skridt i træningen, anvendes en udgave af neural network, hvor nogle af knuderne og deres vægte udelades. En af de typiske situationer er, at hver knude beholdes med en valgt sandsynlighed, kaldet p, eksempelvis $\frac{1}{2}$. Alternativt kan adaptiv dropout anvendes, hvor knuder beholdes med en varierende sandsynlighed, baseret på forskellige parametre [19]. De udeladte knuder og deres vægte vil dermed helt udgå fra netværket i dette skridt af træningen. Herefter vil alle andre tilbageværende

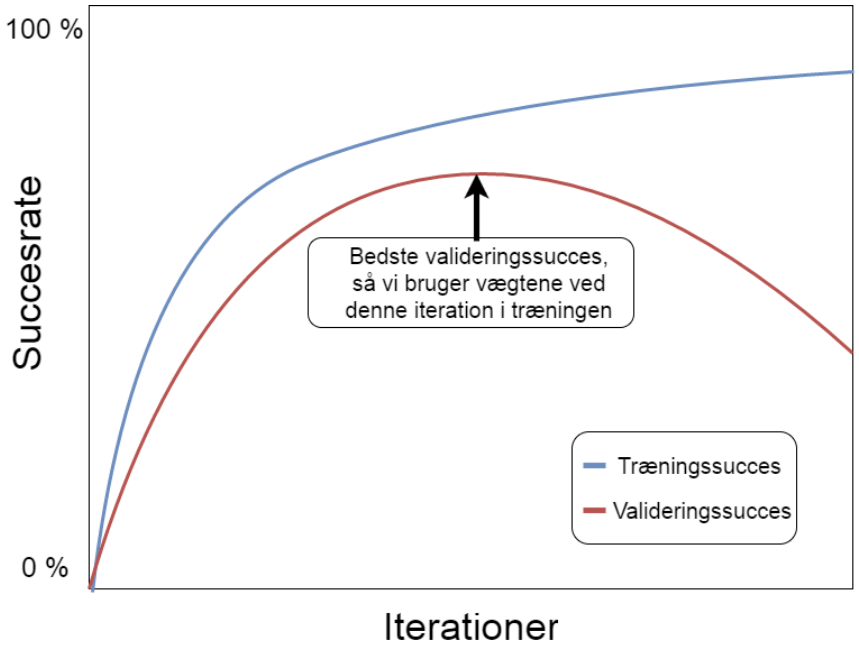
knuder blive skaleret med denne mangel, dvs. de ganges med $\frac{1}{p}$. Til højre i figur 2.5 ses et eksempel på et netværk, hvor dropout anvendes.



FIGUR 2.5:

Til venstre ses et neural network i sin fulde størrelse og til højre ses et skridt i træningen, hvor nogle af knuderne og deres vægte er fjernet ved brug af dropout [20]. Besøgt 31. maj 2016.

Early Stopping er en anden metode til at håndtere overfitting [21]. Dette gøres ved at observere algoritmens præstation på valideringssættet. Et valideringssæt er ligesom et testsæt, men det kan benyttes under træningen, fordi det ikke er en del af testsættet. Undervejs i træningen vil algoritmens præstation på både trænings- og valideringssættet blive testet, og hvis valideringssuccessen pludselig begynder at forværres igen, vil den stoppe med at optimere, det er tegn på, at overfitting finder sted. På figur 2.6 ses et eksempel, hvor overfitting finder sted og valideringssuccessen dermed forværres (oftes arbejdes der med valideringssuccessen i stedet for testsuccessen). Desuden er det optimale punkt for early stopping markeret.



FIGUR 2.6:

Eksempel på hvordan løbende observation af accuracy indikerer, hvornår der forekommer overfitting, og træning af algoritmen derfor bør stoppes.

Kapitel 3

Vores løsning

Vores løsning benytter nogle specifikke tilgange under forskellige faser af det implementerede convolutional neural network. Disse bliver beskrevet herunder. Herudover har vi benyttet os af et datasæt udgivet i forbindelse med en konkurrence om klassificering af sprog og beskrevet hvordan vi udleder information fra lydfilerne samt hvilken software, vi anvender.
Vi har valgt at benytte et convolutional neural network til implementering af vores løsning. Netværket benytter 2x2 filtre til hvert convolution lag, samt 2x2 eller 3x3 MAX-pooling med en stride størrelse på 2. Dette medfører en grad af overlap i MAX-poolingen, som vi ønsker at undersøge effekten af. Til sidst påføres et fully connected layer, som påføres softmax for at evaluere accuracy. Til at optimere den udarbejdede løsning benyttes en specifik optimeringsalgoritme, kaldet Adam. Adam, kort for adaptive moment estimation, er en algoritme, der udfører stokastisk optimering baseret på gradient descent [22]. Denne gradient descent er baseret på at opdatere 2 vektorer med henholdsvis gradientens middelværdi og dens varians, kaldet 1. og 2. momentet. Dertil pålægges i opdateringen en biaskorrektion, da gradientens middelværdi og varians vektorer initialiseres som 0-vektorer fra start. Input til Adam-algoritmen er cross-entropy værdien. Pseudokode for Adam-algoritmen kan ses i figur 3.1.

Vi har valgt at benytte Adam fordi algoritmen har vist sig at være bedst til at klassificere i en række eksperimenter med forskellige datasæt [22].

Vi ønsker ligeledes at håndtere overfitting vha. både dropout og early stopping. Dropout kan implementeres både som en fast sandsynlighed gennem netværket eller en varierende baseret på flere forskellige kriterier undervejs i vores CNN. Vi har valgt at fastsætte dropout-sandsynligheden på enten 0,2, 0,5 eller 0,8, da vi ønsker at undersøge hvilke konsekvenser, dette har for netværket.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

FIGUR 3.1:

Pseudokode for Adam algoritmen [22].

Ligesom dropout kan early stopping implementeres med fokus på forskellige kriterier. Vi har i vores løsning valgt at implementere early stopping således, at vægte og valideringssucces gemmes for hver 500. iteration. Vi gemmer 5 af disse, hvilket betyder, at vi har vægte fra 5 punkter i læringen fordelt over 2000 iterationer. Hvis ikke valideringssuccessen i nogen af disse punkter er stigende, vælges de vægte, som har den største valideringssucces, og læringen afsluttes.

3.1 TopCoder datasættet

For at træne algoritmen til at kunne genkende sprog fra lydoptagelser, har vi benyttet et datasæt udgivet af TopCoder. TopCoder er et firma, som administrerer programmeringskonkurrencer [23]. TopCoder har udgivet det benyttede datasæt i forbindelse med en konkurrence, hvor den religiøse organisationen Faith Comes by Hearing fra 1972 [24], med et ønske om at sprede Bibelens budskab i deres ord over hele verden, søgte en algoritme til at genkende sprog fra lydoptagelser. Datasættet er specifikt blevet indsamlet til dette formål, og derfor anser vi det for et datasæt med høj kvalitet og et lavt niveau af støj [25]. Datasættet er på forhånd delt op i et træningssæt og et testsæt, og da datasættet er udgivet som del af en konkurrence, så er testsættet uden en liste med deres reelle klassificering. Dermed kan vi ikke teste vores egen løsnings effektivitet på det udgivede testsæt. Dette medfører, at vi for at evaluere vores egne resultater bliver nødt til at opsplitte træningssættet yderligere, således at vi kan have et testsæt, som vi ved indeholder den korrekte klassificering. Kendte detaljer om datasættet:

- Træningssættet består af 66176 filer med cirka 10 sekunders tale optaget i 1 af 176 mulige sprog, samt en liste med det talte sprog i hver af filerne.
- Testsættet består af 12320 filer med cirka 10 sekunders tale optaget i 1 af de 176 mulige sprog, men uden en liste over det talte sprog for hver af lydfilerne. Altså har testsættet ingen labels.

Vi spurgte TopCoder og Faith Comes By Hearing [26] om yderligere information vedr. lydfilerne i datasættet, da vi ikke kender til antallet af filer på hvert sprog eller om ordene i filerne er tilfældige sætninger eller ej. Her fik vi intet svar på TopCoders forum, mens Faith Comes By Hearing svarede, at de ikke kunne oplyse om detaljer vedr. datasættet, fordi projektet stadig er i udvikling. Udover manglen på detaljer i vores viden om datasættet, så har vi at gøre med et datasæt indeholdende mange forskellige sprog, som betyder, at vi har brug for meget regnekraft for at træne vores algoritme. Her vælger vi kun at bruge nogle af sprogene for at reducere den krævne regnekraft.

3.2 Feature extraction

Lydfiler indeholder et sæt af målt data, og denne data ønskes delt op i mindre brugbare dele, som regel via tidsopdeling, der indeholder relevant information. Data kan ofte være så støjfyldt, at den ikke direkte kan benyttes til at lave brugbare modeller, hvilket opdelingen hermed også er med til at formindske. Feature extraction går dermed ud på at udlede brugbar information fra den originale data, således at filer med ensartede karaktertræk kan blive grupperet effektivt og automatisk [27]. Der findes mange forskellige feature extractions, og ikke alle disse er lige relevante for det problem, der ønskes løst. Vi har valgt at anvende Mel frekvens Cepstral koefficienter, som tilsammen udgør et Mel frekvens Cepstrum, forkortet hhv. MFCC og MFC [28].

MFC er en repræsentation af et effektspektrum af lyd baseret på en lineær cosinustransformation af et logaritmisk effektspektrum på en ikke-lineær frekvensskala, hvor MFC bruger Melskalaen.

Mel frekvens Cepstral koefficienter findes vha. 3 grundlæggende skridt.

- Først skabes et Effektspektrum af dataen, der arbejdes med.
- Dernæst påføres Melskalaen til det skabte effektspektrum, og logaritmen til værdierne tages heraf.

- Til slut påføres en lineær cosinustranformation på det tidligere resultat

Herunder følger en nærmere beskrivelse af de 3 dele, og hvorfor de benyttes.

Et Effektspektrum , kaldet energy-spectral-density på engelsk, er defineret ved:

$$PS = \int_{-\infty}^{\infty} |x(t)|^2 dt \tag{3.1}$$

Her betegner $x(t)$ en funktion af et signal over tid t. Dette spektrum angiver en repræsentation af den data, vi arbejder med. For et givet signal giver effektspektret et plot over den del af signalets energi pr. tidsenhed, som falder inden for et givent frekvensinterval. Dermed er det muligt at observere hvor i dataen, der er den største mængde af energi [29]. Effektspektret skabes oftest vha. en diskret Fourier Transformation.

Melskalaen er defineret ved:

$$m = 1125 \ log\left(1 + \frac{f}{700}\right) \tag{3.2}$$

Her betegner f den faktiske frekvens målt i Hz i lydfilen og m betegner frekvensen på melskalaen.

Årsagen til at melskalaen bruges er, at mennesket ikke opfatter lyd lineært. Melskalaen er en model, som er lavet til at beskrive menneskets auditive system, som står for høresansen, lineært. Frekvenser fra 0-1000hz opfattes lineært, mens frekvenser over 1000hz opfattes logaritmisk [30].

En lineær cosinustransformation er defineret som:

$$X_k = \sum_{n=0}^{N-1} x_n cos[\frac{\pi}{N} n(k + \frac{1}{2})] \qquad k = 0, ..., N-1 \tag{3.3}$$

Her betegner N størrelsen af inputtet og x_n betegner den n'te inputværdi.

En lineær cosinustransformation, ofte kaldet discrete cosine transformation (DCT), er en transformation af et signal over tid [31]. Den adskiller sig fra en diskret fourier transformation ved kun at benytte sig af cosinus-funktionen, hvorimod en diskret fourier transformation f.eks. også kan benytte sinus-funktionen.
En DCT bruges i dette tilfælde til at dekorrelere sammenhængen mellem de komponenter

i en dannet filterbank, eftersom disse er overlappende. En filterbank inddeler et signal i mindre dele, hvor de enkelte dele repræsenterer et effektspektrum. [32].

3.3 Implementering

Til at løse det introducerede problem benyttes et open source bibliotek kaldet Tensorflow. Tensorflow er udviklet af Google og er bygget til at løse forskellige problemer inden for maskinlæring. Det baserer sig på numerisk beregning via dataflowgrafer. Grafer, hvis knuder repræsenterer en matematisk operator, og kanter, som repræsenterer multidimensionelle arrays, kaldet en tensor. Denne opbygning er med til at gøre Tensorflow til et fleksibelt værktøj [33].

> "It is machine learning software being used for various kinds of perceptual and language understanding tasks" — *Jeffrey Dean, minute 0:47 / 2:17 - TensorFlow: Open source machine learning, Youtube video*

Tensorflow er indtil videre af Google implementeret til brug i programmeringssprogene Python og C++. Vi har valgt at benytte Tensorflow i Python. Da vi skal forsøge at genkende tale ud fra lydfiler, har vi ligeledes brug for et bibliotek til at uddrage MFCC features, som kan benyttes i Tensorflow. Til dette har vi valgt at bruge biblioteket LibROSA [34]. Der eksisterer mange forskellige biblioteker til feature-uddragning, men vi valgte at benytte LibROSA, fordi vi havde færrest problemer med installationen af dette bibliotek.

I det følgende beskrives noget af den vigtigste kode, som benyttes til feature extraction, pooling, optimering mm. af den samlede løsning. Ønskes en brugervejledning, så findes denne i kapitel 4.

Til at træne og teste vores CNN kræves MFCC-data. Denne uddrages gennem LibROSA vha. linjerne herunder, som loader hver lydfil og uddrager features:

```
y, sr = librosa.load(filename,sr=sampling_rate)
mfcc_array = librosa.feature.mfcc(y=y,sr=sr,n_mfcc=number_of_mfccs)
```

`sampling_rate` er antallet af lydprøver taget pr. sekund og `number_of_mfccs` er det ønskede MFCC-format. MFCC-dataen gemmes herefter i filer, og nu kan den benyttes i netværket. Indledningsvis initialiseres grafen og dens vægte. Dernæst udføres convolution og MAX-Pooling.

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool(h_conv1)
```

`x_image` er vores input, og `W_conv1` og `b_conv` er vægte og bias. `tf.nn.relu` er en aktiveringsfunktion benyttet, som har vi sig at være god i forbindelse med andre neural networks med flere skjulte lag [35]. Herefter oprettes et varierende antal lag og en dropout-teknik implementeres. Til sidst tilføjes et softmax-lag. Herunder ses koden for dropout og softmax-laget.

```
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

`h_fc1` er et fully-connected lag og `keep_prob` er sandsynligheden for at beholde hver node i netværket. Dette netværk inklusiv dropout benyttes så til at finde softmax-vægtene vha. netværkets vægte og bias-værdier. Efter ovenstående er netværket initialiseret og træningen kan påbegyndes. Adam-algoritmen tilføres på hvert enkelt batch, og en procentvis fejlrate for valideringssættet udregnes efter træningen i hver iteration. Dette skridt er det vigtigste skridt til løsningen af klassificeringsproblemet, da det er her netværket opdateres og dermed lærer. Nedenunder findes den kode, der bruges til at træne netværket.

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

Vi ser, at træningsskridtet benytter Adam-algoritmen på de minimerede `cross_entropy` værdier. Hvert enkelt `train_step` bliver herefter udført for alle batches, sådan at alle inputværdier bliver brugt. Det sker i nedenstående linie indtil early stopping kriteriet er nået eller efter det satte maks antal iterationer.

```
train_step.run(feed_dict={x: batch[0],
                          y_: batch[1],
                          keep_prob: 1 - dropout_prob})
```

`batch[0]` og `batch[1]` er hhv. data og labels for vores batch. `keep_prob` er sandsynligheden for at beholde en knude. Dermed trækker vi vores indstillede `dropout_prob` fra 1 for at finde sandsynligheden. Til slut evalueres netværket på testsættet, og accuracy printes med linjen:

```
print("test accuracy %g"%accuracy.eval(feed_dict={x: test_data,
                                                   y_: test_labels,
                                                   keep_prob: 1.0}))
```

Her ser vi, at dropout ikke længere benyttes, da vi ønsker at beholde hele netværket, fordi vi skal udnyttes dets fulde kapacitet til at klassificere testsættet.

Kapitel 4

Brugervejledning

For at kunne køre programmet kræves tre dele udført:

1. Downloade og udtrække de 24 valgte sprog fra datasættet
2. Udføre MFCC feature extraction
3. Kørsel af det neurale netværk

1) For at hente datasættet skal hhv. lydfilerne, som er zippet og kan hentes via linket her, og listen med labels, som er gemt i en .csv fil og kan hentes her, bruges. Dernæst skal mappen `trainingdata` og filen `trainingData.csv` gemmes i samme mappe. Hernæst skal filen `make_dataset.py` køres for at udtrykke de nødvendige mapper og .csv filer til de forskellige mulige antal sprog. Mulige antal sprog er 3, 6 og 24. Når `make_dataset.py` eksekveres, skal der specificeres hvilke sprog, der benyttes. Der er frihed til at vælge alle sprog, men de første 3 specificerede sprog vil blive benyttet til testning af netop 3 sprog. Det samme er gældende for 6 sprog.

2) For at udføre feature extraction skal filen `make_mfcc.py` eksekveres. Inde i filen skal antallet af sprog samt det ønskede MFCC-featureformat specificeres. MFCC-formatet er en matrix af NxN reelle tal. mulige MFCC-featureformater er 8x8, 16x16 og 32x32. Som standard er disse sat til 3 sprog og 8x8 MFCC'er. Dette resulterer i en mappe med .txt filer, som sammen med en labelliste i .csv er inputtet til netværket.

3) Til slut skal filen `make_cnn.py` eksekveres i et linux-baseret miljø, som f.eks. Ubuntu for at køre vores CNN i Tensorflow. Her kan de forskellige parametre specificeres. Som standard er disse sat til de værdier, der svarer til det øverste venstre resultat i tabel 6.1.

Kapitel 5

Teststrategi

Vi ønsker at teste hvilke parametre, der opnår bedste accuracy på det anvendte datasæt. Algoritmen bruger softmax regression til at finde de forskellige klassers vægte. Den klasse, der opnår den største vægt klassificeres som bud på den enkelte fil. Den samlede accuracy måles ved at sammenligne den trænede algoritmes bud på klassificering af hver fil i testsættet over for listen med alle testfilerne og deres korresponderende korrekte label. Accuracy måles her ved:

$$\texttt{Accuracy} = \frac{\texttt{Correct classifications}}{N} \tag{5.1}$$

hvor `Correct Classifications` er antallet af korrekte klassificeringer i datasættet og N er det samlede antal filer, der klassificeres. Løsningen testes med 3, 6 og 24 sprog. i tabel 5.1 ses listen over de 24 sprog der benyttes. De første 3 sprog i øverste række benyttes, når der testes med 3 sprog eftersom disse er forskellige. Selvom sprogene ikke behøver være forståelige kan der med sikkerhed høres forskel på disse. Hele den øverste række benyttes, når der testes med 6 sprog.

Dutch	Polish	Vietnamese	Arabic	Bench	Hindi
Chipaya	Eleme	Fai South	Garhwali	Ignaciano	Jingpho
Kuman	Lobi	Malay	Napu	Oromo Eastern	Quechua Southern Pastaza
Romanian	Sango	Tol	Waiwai	Yawa	Zapotec Isthmus

Tabel 5.1:

24 sprog, som benyttes til test. Sprogene er betegnet på engelsk, da det ikke har været muligt at finde danske oversættelser for mange af disse sprog, som vi ikke har hørt om før.

Årsagen til valget af netop disse sprog er primært tilfældig. TopCoders datasæt indeholder 176 forskellige sprog, og vi valgte de første tre sprog (hollandsk, polsk og vietnamesisk), fordi vi var overbeviste om, at vi ville kunne kende forskel på disse, hvilket ville hjælpe os i det indledende arbejde med evaluering af algoritmen. Herefter blev de resterende sprog valgt baseret på deres startbogstav, for at undgå at komme til at blande rundt i sprogene med arbejdet senere hen.

5.1 Testede parametre

- Dropout-sandsynlighed
- MFCC-format
- Batch-størrelse
- Antal lag
- Pooling-størrelse med fast stride-størrelse

Dropout-sandsynlighed: I artiklen *Adaptive dropout for training deep neural networks* fra University of Toronto blev det vist, at en variation i datasættets størrelse havde inflydelse på, hvor godt dropout virkede med fastsat dropout-sandsynlighed [19]. Vi har valgt at undersøge om en lav, middel eller høj dropout-sandsynlighed opnår de bedste resultater. De valgte værdier, er $p = 0.2$, $p = 0.5$ og $p = 0.8$.

MFCC-format: Forskellige MFCC-formater gør, at mængden af information fra lydfilerne er varierende. Vi ønsker at undersøge om en større mængde information giver bedre generalisering og dermed accuracy på uset data eller i stedet resulterer i overfitting og dermed dårligere accuracy. Vi har valgt formaterne 8x8, 16x16 og 32x32.

Batch-størrelse: Et batch er et delsæt af træningsfilerne, som benyttes til at udføre et skridt i træningen. Et batch anvendes for at undgå at skulle bruge hele datasættet i hver iteration af træningen og dermed rigtig meget regnekraft. Vi ønsker at undersøge om valget af batch-størrelse har en effekt på netværkets evne til at klassificere. De valgte batch-størrelser er, 10, 50 og 250.

Antal lag: Ved at justere antallet af lag i vores CNN, vil evnen til at repræsentere komplekse løsninger varieres. Et problem ved øget kompleksitet opstår, når for mange lag resulterer i overfitting i stedet for god evne til at generalisere. Det forventes, at flere lag giver bedre resultater, da vi har implementeret mekanismer til at håndtere overfitting, mere præcist dropout og early stopping. Vi har valgt at teste på 1, 2 og 3 convolutionlag i netværket.

Pooling-størrelse med fast stride-størrelse: Ved at bruge MAX-pooling ønskes at mindske dimensionaliteten og sløre billedet uden at miste vigtig information. Vi har valgt at gøre dette med 2 pooling-størrelser: 2x2 og 3x3, hvor 3x3 slører billedet mere end 2x2. Heraf har vi fastsat stride-størrelsen på 2, således at dimensionaliteten er ens hvadenten vi bruger 2x2 eller 3x3 MAX-pooling.

5.2 Faste parametre

- Antallet af iterationer
- Antal outputfeatures
- Læringsraten

Antallet af iterationer: Det maksimale antal af iterationer er i netværket fastsat til 50.000. Ud fra de udførte tests har vi observeret, at den højeste valideringssucces ofte findes mellem 5.000 og 10.000 iterationer. Det højeste antal af iterationer observeret inden early stopping træder i kraft viste sig at være omkring 25.000. Hvis det på uset data er tilfældet at early stopping ikke træder i kraft, så er det vurderet, at grænsen vil være høj nok til at opnå næsten optimale værdier, eftersom vi forventer, at det er yderst sjældent at valideringsuccessraten stiger yderligere.

Antal output features: Antallet af outputfeatures fordobles fra forrige lag med undtagelse af inputlaget, da inputlaget kun tager én feature ind og returnerer MFCC formatet ud.

Læringsraten: Vi har valgt at fastsætte læringsraten på 0,0001, da vi benytter Adam-algoritmen. Læringsraten i Adam-algoritmen opdateres nemlig løbende ud fra første- og andetmomenterne for hvert skridt i et forsøg på at opnå optimal læringsrate undervejs [22].

Kapitel 6

Resultater

Med de tidigere nævnte parametre kan accuracy for testsættet findes. Række 1 i tabellerne viser MFCC-formatet. Række 2 indeholder forkortelser for de parametre, der varieres, og kolonne 1, 2 og 3 viser, hvor meget den pågældende parameter er blevet ændret. Forkortelserne er således:

- L = Antal lag
- B = Batch-størrelse
- P = Pooling-størrelse
- p = Dropout sandsynlighed
- NA = Ikke tilgængelig grundet netværkets implementering

Tabel 6.1 viser resultaterne for 3 sprog, tabel 6.2 viser resultaterne for 6 sprog og tabel 6.3 viser resultaterne for 24 sprog.

Accuracy på 3 sprog

			8x8			16x16			32x32		
L	B	P	$p = 0.2$	$p = 0.5$	$p = 0.8$	$p = 0.2$	$p = 0.5$	$p = 0.8$	$p = 0.2$	$p = 0.5$	$p = 0.8$
1	10	2x2	89,47	90,43	87,55	93,77	94,25	91,38	37,32	89,00	89,00
		3x3	93,78	92,82	91,39	93,78	94,74	93,78	61,72	31,10	70,81
	50	2x2	90,43	91,38	88,03	88,99	91,86	**96,66**	34,93	**97,13**	91,39
		3x3	**94,74**	90,91	92,82	92,34	93,30	87,08	93,78	94,26	94,26
	250	2x2	86,12	89,47	90,43	89,95	91,38	91,38	90,43	93,78	94,74
		3x3	90,43	90,43	91,39	91,39	94,26	95,69	88,04	96,17	95,22
2	10	2x2	88,51	89,47	92,34	93,77	95,21	93,30	34,93	93,78	94,74
		3x3	84,69	91,87	92,34	94,26	93,30	93,30	34,45	94,74	93,78
	50	2x2	91,38	90,90	88,99	94,25	92,82	92,34	66,03	93,30	95,69
		3x3	92,28	90,43	91,39	91,87	92,34	90,43	89,47	94,74	96,17
	250	2x2	91,38	88,99	90,90	92,34	92,84	**95,69**	**96,65**	92,82	94,74
		3x3	91,87	**92,82**	91,87	94,74	94,26	93,78	96,17	94,26	97,61
3	10	2x2	89,47	88,51	90,90	90,43	**97,12**	90,43	89,00	93,30	94,26
		3x3	NA	NA	NA	94,26	91,39	91,87	94,74	**97,61**	93,78
	50	2x2	88,51	**90,90**	88,03	93,47	94,73	93,47	94,74	95,22	97,13
		3x3	NA	NA	NA	92,82	94,26	92,82	89,95	94,74	95,69
	250	2x2	88,99	88,99	89,95	94,25	94,73	93,30	96,17	94,26	92,82
		3x3	NA	NA	NA	90,91	94,74	92,82	33,49	92,82	94,26

Tabel 6.1:

Denne tabel viser resultatet af de samlede test udført på 3 sprog. Værdier markeret med fed, er den højeste accuracy opnået indenfor det pågældende lag, for det pågældende MFCC-format. Tallet 94,74 er f.eks. den højest opnåede accuracy for et nætværk med ét lag, og 8x8 MFCC-format.

Accuracy på 6 sprog

			8x8			16x16			32x32		
L	B	P	$p = 0.2$	$p = 0.5$	$p = 0.8$	$p = 0.2$	$p = 0.5$	$p = 0.8$	$p = 0.2$	$p = 0.5$	$p = 0.8$
1	10	2x2	74,68	71,87	78,43	79,68	81,87	60,00	18,75	16,88	41,56
		3x3	78,12	74,69	78,44	80,31	80,00	78,13	30,94	15,94	18,75
	50	2x2	79,37	79,37	75,00	78,75	77,50	82,18	84,38	**90,94**	86,86
		3x3	77,81	**80,00**	77,19	**82,19**	80,63	80,94	19,38	15,00	86,56
	250	2x2	74,68	**80.00**	75,62	76,87	78,75	78,12	84,06	82,50	87,50
		3x3	76,25	77,81	77,50	81,25	78,13	79,69	87,81	89,06	87,19
2	10	2x2	80.00	76,56	79,68	77,81	**85,93**	81,56	86,56	90,94	89,69
		3x3	73,13	73,75	71,88	80,00	82,81	77,19	15,63	87,19	88,44
	50	2x2	76,87	**81,25**	74,06	80,62	82,18	82,50	86,88	90,94	91,56
		3x3	74,69	75,63	71,88	82,81	85,31	79,69	87,81	89,06	90,63
	250	2x2	76,87	78,12	76,87	83,43	85,62	83,75	88,13	90,63	90,00
		3x3	76,56	73,75	72,50	82,19	84,06	84,06	85,94	86,88	**91,88**
3	10	2x2	76,87	78,43	70,00	80,93	82,50	75,93	83,44	88,44	86,88
		3x3	NA	NA	NA	73,75	**87,19**	79,06	84,06	85,63	85,31
	50	2x2	80,62	80,31	77,50	82,50	85,31	81,56	85,00	89,69	90,31
		3x3	NA	NA	NA	83,13	85,00	82,19	84,38	88,13	88,75
	250	2x2	77,81	**81,25**	77,50	76,56	85,62	83,75	86,88	89,06	91,56
		3x3	NA	NA	NA	79,38	80,94	85,00	86,25	89,06	**91,88**

Tabel 6.2:

Denne tabel viser resultatet af de samlede test udført på 6 sprog. Værdier markeret med fed, er den højeste accuracy opnået indenfor det pågældende lag, for det pågældende antal MFCC-features. Tallet 80.00 er f.eks. den højest opnåede accuracy for et nætværk med ét lag, og 8x8 MFCC-format.

Accuracy på 24 sprog

			8x8			16x16			32x32		
L	B	P	$p = 0.2$	$p = 0.5$	$p = 0.8$	$p = 0.2$	$p = 0.5$	$p = 0.8$	$p = 0.2$	$p = 0.5$	$p = 0.8$
1	10	2x2	46,58	45,51	37,67	58,16	55,75	7,35	3,97	3,90	4,88
		3x3	51,27	44,89	36,24	58,17	51,92	9,63	4,62	4,36	4,10
	50	2x2	51,65	47,49	45,67	55,43	57,51	13,92	4,62	11,00	13,79
		3x3	52,11	51,01	34,35	59,60	**61,09**	52,18	4,16	4,16	3,12
	250	2x2	**54,19**	50,55	47,36	57,12	59,79	57,12	63,70	62,52	26,61
		3x3	54,13	51,98	43,20	59,92	59,73	58,43	11,26	**65,13**	19,71
2	10	2x2	49,31	47,03	42,29	61,28	65,32	49,57	4,75	25,70	7,55
		3x3	47,50	47,76	41,12	56,80	61,09	54,72	4,16	55,76	4,55
	50	2x2	52,63	49,51	44,95	63,89	61,93	62,19	65,58	70,00	69,88
		3x3	53,87	48,73	43,72	62,85	60,70	57,97	4,49	66,62	65,84
	250	2x2	55,10	49,64	46,84	65,90	**65,97**	63,04	65,91	63,89	68,84
		3x3	**54,91**	51,27	42,94	62,39	65,58	63,37	4,03	68,84	**70,72**
3	10	2x2	47,82	50,48	41,05	62,71	65,77	58,16	64,09	65,78	14,51
		3x3	NA	NA	NA	60,05	53,74	54,26	4,29	3,77	4,23
	50	2x2	53,09	51,39	45,02	62,65	64,60	63,24	65,06	70,92	74,37
		3x3	NA	NA	NA	62,98	**66,95**	64,28	65,39	64,74	65,84
	250	2x2	**53,28**	50,87	47,56	64,08	64,15	67,72	66,04	70,53	**74,76**
		3x3	NA	NA	NA	63,96	60,64	63,37	69,16	67,85	71,83

TABEL 6.3:

Denne tabel viser resultatet af de samlede test udført på 24 sprog. Værdier markeret med fed, er den højeste accuracy opnået indenfor det pågældende lag, for det pågældende MFCC-format. Tallet 54,19 er f.eks. den højest opnåede accuracy for et nætværk med ét lag, og 8x8 MFCC-format.

Den bedste accuracy for 3 sprog er **97,61**. Den bedste accuracy for 6 sprog er **91,88**. Den bedste accuracy for 24 sprog er **74,76**.

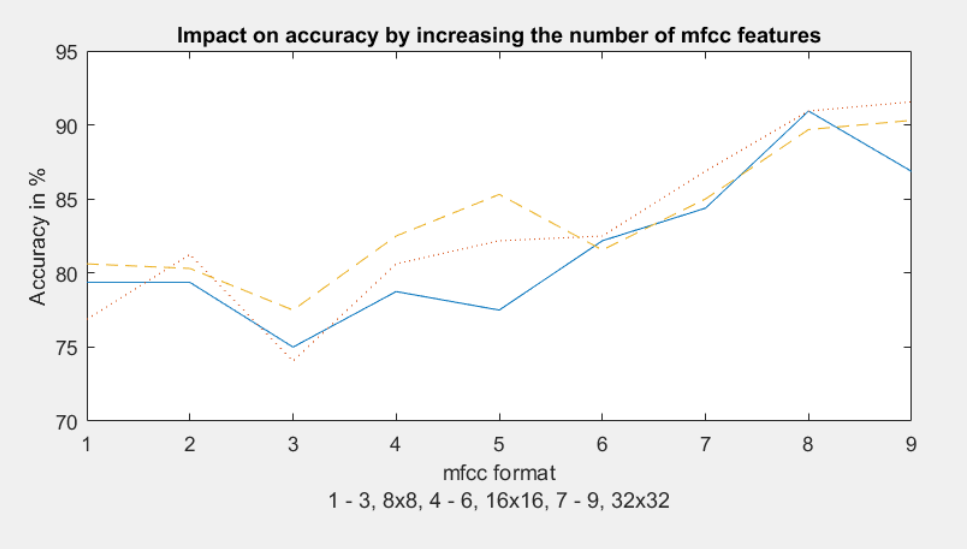
Ud fra tabel 6.1, 6.2, 6.3 og grafen i figur 6.1 ses det, de bedste resultater opnås ved 3 skjulte lag og en MFCC-format på 32x32. Der er en generel tendens til, at hver gang MFCC-formatet forøges, stiger accuracy ligeledes for både 3, 6 og 24 sprog. Der ses derimod i en forøgelse i antallet af lag ikke den samme markante forbedring, som ved en forøgelse i MFCC-formatet. Ved MFCC-format 8x8 bliver accuracy dårligere ved flere lag i nogle af tilfældende.

Ud fra tabel 6.2 og grafen i 6.2 ses det, at dropoutraten har en indflydelse på resultatet. Det lader til, at den mest stabile værdi er $\mu = 0.5$, hvilket stemmer godt overens med bl.a. [19], der vælger at fixere parameteren til 0.5 på baggrund af tidligere viden. Her skal stabil forstås som et udtryk for, at accuracy ikke har en markant stor ændring, når antallet af lag forøges.
I [19] vises det, at en $\mu = 0.5$ værdi er brugbar til at forøge accuracy som også er tilfældet for os.

Det ses, at accuracy for varierende batch-størrelse generelt er meget svingende. Ud fra tabel 6.2 og fra graferne i figur 6.3 ses det, at alle tre benyttede størrelser har været med

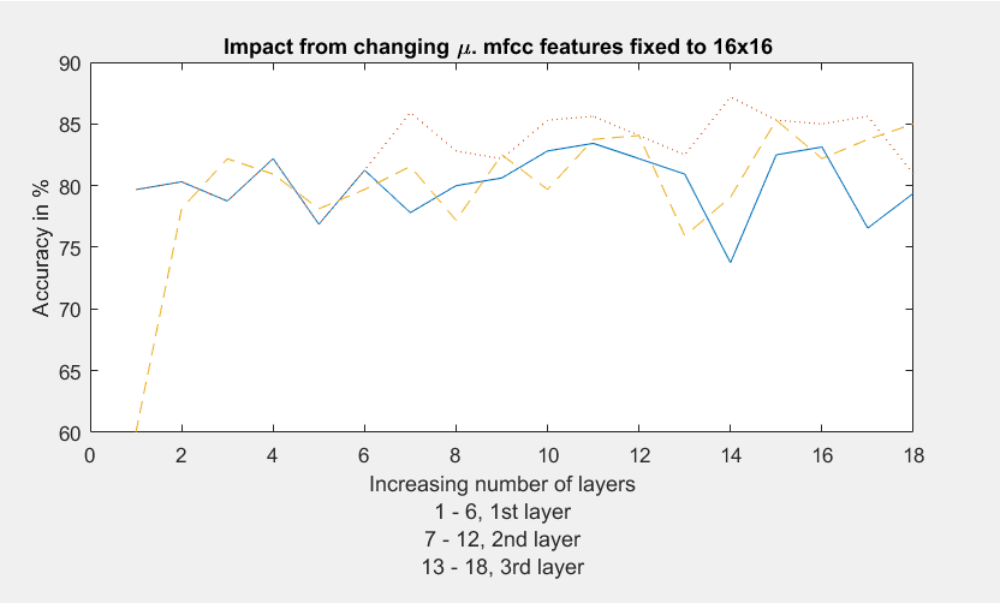
til at give den bedste accuracy på et tidspunkt. En batch-størrelse på 10 giver f.eks. dårligere accuracy end en batch-størrelse på 50 ved 8x8 (MFCC-format) og ved 32x32 i andet lag. Dertil er accuracy bedre ved 16x16. Samtidigt er en forøgelse i batch-størrelse i andet lag ved 16x16 ikke grunden til faldende accuracy, da en batch-størrelse på 250 er stort set identisk med en batch-størrelse på 10. Ved 24 sprog ses en tendens til, at algoritmen fejler ved 32x32 og batch-størrelse 10 eller 50, fordi resultaterne svarer til en tilfældig klassificering af sprogene. Det kan derfor ikke konkluderes, at batch-størrelsen alene har en direkte indvirkning på accuracy ud fra resultaterne.

Ved testning af pooling-størrelsen, kan det bemærkes, at en størrelse på både 2x2 og 3x3 har sine fordele. En pooling-størrelse på 2x2 ser ud til at være bedre end 3x3 i to af graferne i figur 6.4, netop ved kun ét lag og stigende MFCC-format. Ved to lag ses det derimod, at en pooling-størrelse på 3x3 giver bedste resultat jo højere MFCC-format, der bruges. Bemærk, at en pooling-størrelse på 2x2 ikke giver markant dårligere resultat end 3x3, og derfor også er med til at øge accuracy ved en stigning i MFCC-formatet. Dermed ser det ud som om, at en pooling-størrelse på 3x3 er stabil i et højere MFCC-featureformat.



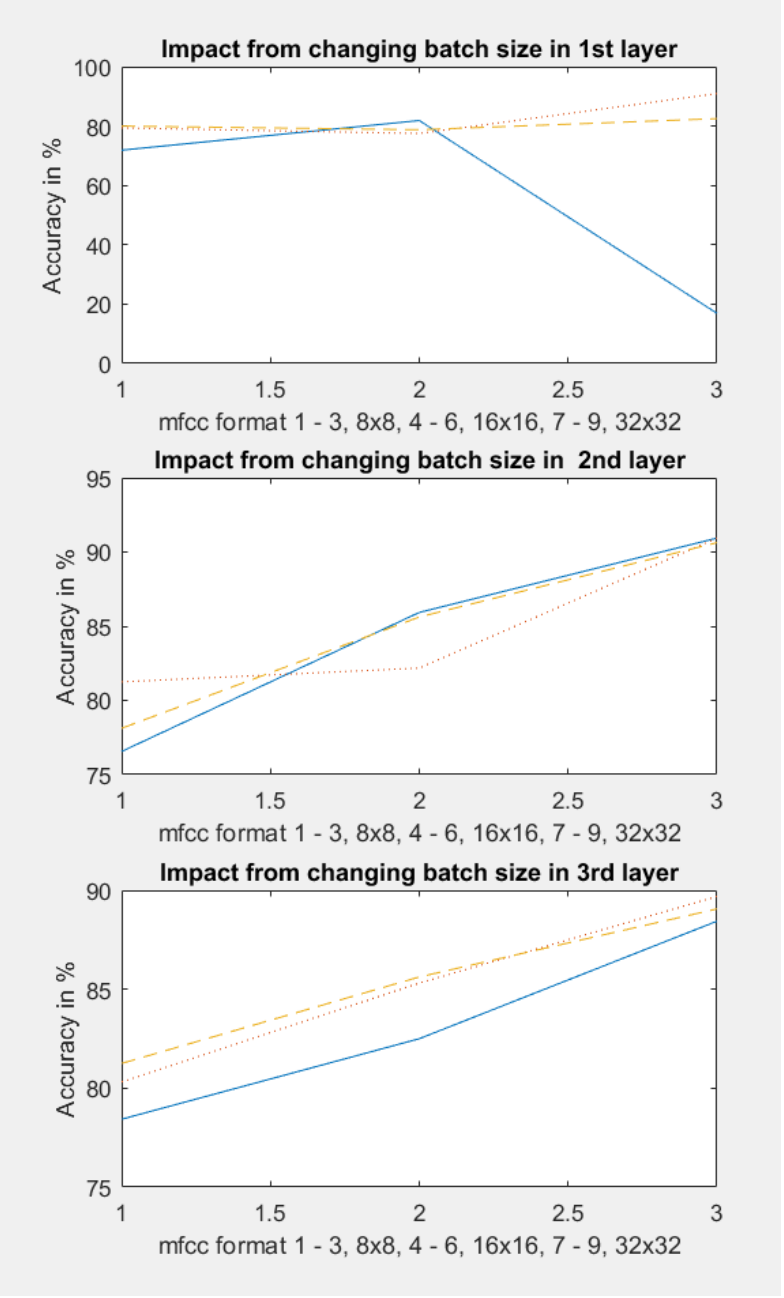
FIGUR 6.1:

Graf over resultater ved ændring i MFCC-format. De tre linier repræsenterer antallet af lag der testes på. Blå = 1 lag, rød = 2 lag, gul = 3 lag. Pooling-størrelse og batch-størrelse, er fastholdt til 2x2 og 50. Her ses det, at en ændring i MFCC-formatet, forøger accuracy i alle lag.



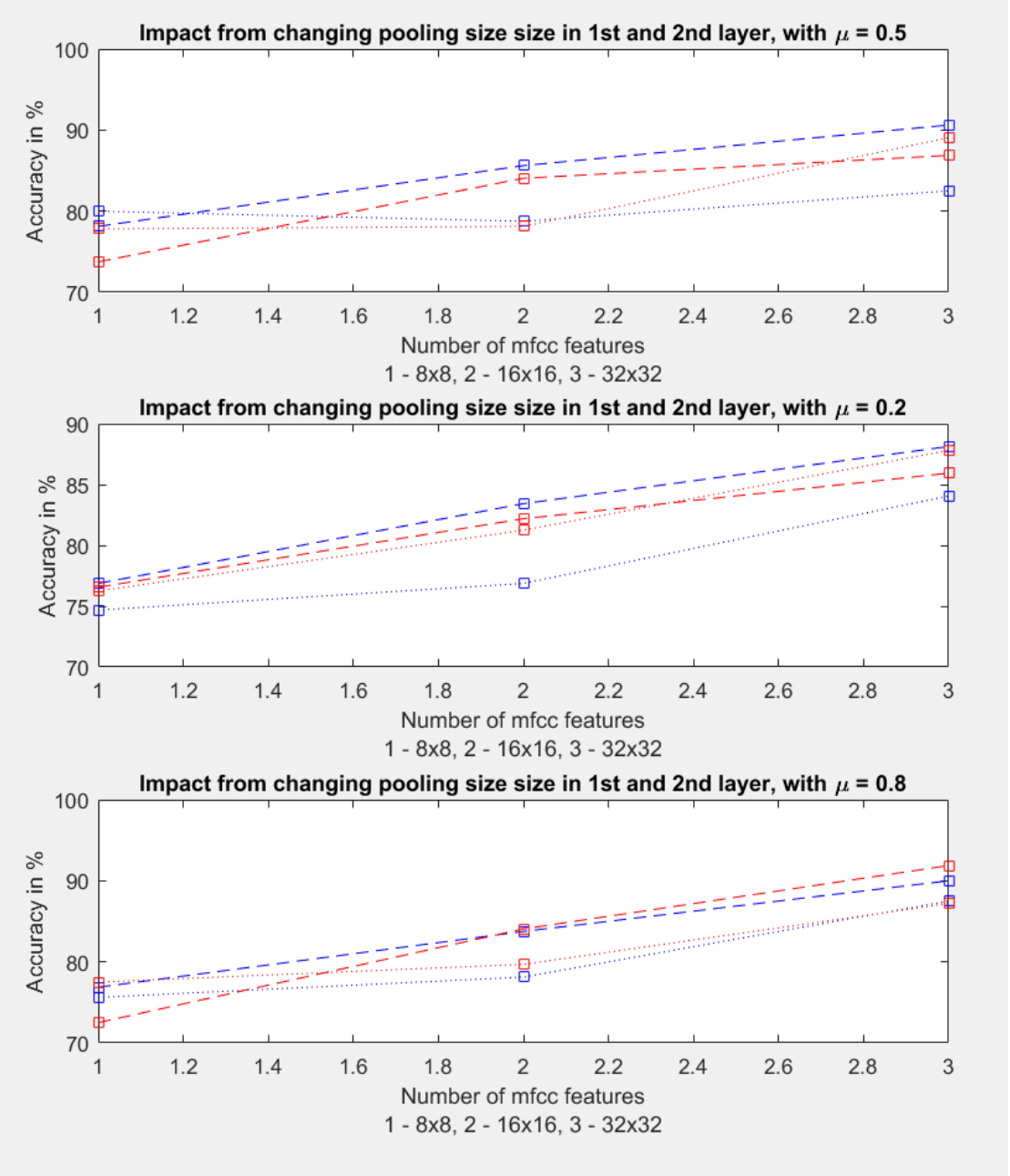
FIGUR 6.2:

Graf over resultater ved ændring i dropoutraten μ. De tre linier repræsenterer de tre forskellige μ værdier igennem de tre lag. Blå = 0.2 rød = 0.5, gul = 0.8. Det ses at en μ værdi på 0.5 generelt egner sig bedst i de højere lag. Accuracy er højere og udsvinget er mindre. Bemærk, I det første lag er accuracy ens for $\mu = 0.2$ og 0.5



FIGUR 6.3:

Grafer over resultater ved forskellige batch-størrelser. De forskellige linier repræsenterer hver batch-størrelse i hver graf. Blå = 10, rød = 50, gul = 250. Pooling-størrelse og dropout rate er fastholdt til 2x2 og 0.5. Det ses af graferne, at accuracy ikke er konsekvent højere ved en identisk batchstørrelse i nogle af lagene. Dermed kan det ikke med sikkerhed siges, at en ændring i batch-størrelsen har en indvirkning direkte på accuracy. Det er tilfældigt hvilken værdi, der egner sig bedst i situationen.



FIGUR 6.4:

Grafer over resultater ved skiftende pooling-størrelse, på 2x2 og 3x3. Farven blå repræsenterer en pooling-størrelse på 2x2, mens rød repræsenterer 3x3. liniens form viser antallet af lag der testes for. Prikkede linier = 1 lag, Stiplede linier = 2 lag. Batch-størrelsen er fikseret til 250 i alle tilfælde. For hver graf ændres μ for 0.5, 0.2, og 0.8. Det ses at en pooling-størrelse på 3x3 i første lag giver bedre resultater. Omvendt ses det i andet lag, at en pooling-størrelse på 2x2 fører til bedre resultater.

Kapitel 7

Diskussion

I dette kapitel reflekteres over de fundne resultater i arbejdet med vores løsning. Problemer med datasættet, det implementerede netværk mm. bliver forsøgt analyseret og beskrevet, og netværkets resultater sammenlignes med det relaterede arbejde beskrevet i introduktionen. Sammenlignet med det relaterede arbejde har vores resultater vist sig at være gode efter vores mening.

Datasættet

Et problem, der kan påvirke accuracy for det testede datasæt er størrelsen på træningssættet. Da det er relativt småt, sammenlignet med f.eks. [5], der bestod af over 200 timers tale for hvert af 8 sprog. I dette forsøg opnåede de med 2 lag en accuracy på 88,39%, som kan sammenlignes med 91,88%, som vi opnåede ved 2 lag med MFCC-format 32x32 og 6 sprog. Havde vi eksempelvis haft over 200 timers tale for hvert af vores 6 sprog forventer vi at kunne have opnået endnu bedre resultater, fordi vi dermed ville kunne generalisere bedre. Vores datasæt bestod for hvert sprog af cirka 350 lydfiler af 10 sekunders varighed, hvoraf vi benyttede os af cirka 70% til træning og resten til test. Dette giver os et datasæt bestående af cirka 2450 sekunders data for hvert sprog, hvilket svarer til cirka 40 minutter, som er markant lavere end 200 timer. Et andet muligt problem ved datasættet anser vi for at være ensartede sprog. Nogle af sprogene i datasættet havde ens karaktertræk i udtale, eftersom de i større eller mindre grad har samme oprindelse. Dette har sandsynligvis haft en betydning for resultaterne ved især 24 sprog, hvor vægtene i softmax-laget forventes at have været nærtliggende. Dette er desuden grunden til, at vi ved test af 3 sprog kun benyttede sprog, som var nemmere at skelne fra hinanden ud fra udtale. Dette ses dog ikke som en indvirkende faktor i vurderingen af hvilke parameterværdier, der bedst klassificerer, da alle resultaterne anskues for at være lige afhængige af sprogenes oprindelser. Et tredje muligt problem

vedr. datasættet, der har relevans for den samlede accuracy, er brugen af sprog, som har været svære at indsamle data for. Til dataindsamlingen af sprog som kun snakkes i meget begrænset omfang kan resultere i, at klassificeringen af sproget ender med at være en klassificering af speakeren. Hvis i stedet testsættet indeholder filer af en anden speaker vil disse med større sandsynlighed ende med at blive klassificeret forkert.

Implementering af early stopping

I implementeringen af vores convolutional neural network, har vi valgt at sætte en grænse for hvornår early stopping indtræffer. Det kan blive et problem, da vi under læringsprocessen kan ende i et lokalt minimum. Dette problem mener vi at have observeret ved test for 24 sprog, hvor værdier som 3,97%, 4,10% og 4,29% eksisterer ved de små batch-størrelser på hhv. 10 og 50. Værdierne er alle markant lavere end resten af de omkringliggende tests og på niveau med tilfældig klassificering for hver fil. Vi mener, at vi ved første iteration måske har opnået et godt, men tilfældigt gæt, som derefter ikke er blevet forbedret inden early stopping kriteriet er nået, fordi de små batch-størrelser har medført en mangel på træning på repræsentative batches. Dog ses flere steder i tabellen, at selvom batch-størrelsen er 10, så får vi brugbare resultater. Årsagen hertil formodes derfor at være grundet mere repræsentative batches undervejs i træningen. Skal problemet undgås fremadrettet, kan grænsen for hvornår early stopping indtræffer øges. Denne grænse er baseret på, at vi ikke har observeret en større ændring i accuracy, i mange iterationer efter, hvis vi befinder os i et lokalt minimum. Samtidig kunne en mulig løsning være at ændre early stopping kriteriet til at omhandle antallet af behandlede inputs i stedet for at omhandle antallet af iterationer. På den måde vil en ændring i batch-størrelsen ikke være afgørende for hvordan early stopping opfører sig.

Brugen af ReLU

Da implementeringen af netværket benytter rectified linear Units, som aktiveringsfunktion, kan dette have betydning for accuracy. Hvis en tilpas stor mængde af neuroner tillægges værdien 0 via aktiveringsfunktionen, vil neuronerne være ubrugelige. Disse neuroner kaldes ofte døde neuroner, da de ikke bidrager til netværket med deres output vægt mere. Som regel sker dette hvis learning rate er sat for højt. Derfor forventes det ikke at dette problem er til stede i netværket i høj grad, da vi benytter Adam algoritmen, der tilpasser learning rate undervejs.

Netværkets kompleksitet

Ud fra de observerede resultater så vi generelt, at en øget kompleksitet i netværket med større MFCC-format gav højere accuracy. Vi forventer derfor, at dette vil fortsætte, hvis netværkets kompleksitet øges yderligere, og MFCC-formatet forøges. Dette er ikke tilfældet, hvis MFCC-formatet derimod er for småt til netværkets kompleksitet. Dette formoder vi fører til overfitting, og kan være en af grundene til at vi ser en lavere accuracy for 8x8 MFCC-format ved det tredje lag (53,28) end det første lag (54,19).

Dropout-sandsynligheden

Dropout-sandsynligheden havde indvirkning på den opnåede accuracy. Vi fandt, at en sandsynlighed på 0.5 gav det mest stabile resultat i kørsler med flere skjulte lag i et komplekst netværk. Vi formoder, at en dropout-sandsynlighed på 0.5 indikerer, at der findes en balance mellem evnen til generalisering og mængden af information netværket kan lære af. Hvis dropout-sandsynligheden er sat for højt, da formodes det, at vi mister for meget information fra netværket til at generalisere. Hvis dropout-sandsynligheden derimod er sat for lavt, da formodes det, at vi ikke benytter nok information fra netværket til at generalisere.

Forskel i pooling-størrelsen

Vi observerede at der ikke var en markant stor forskel på forskellige pooling-størrelser jo højere MFCC-formatet og netværkets kompleksitet blev. I begge tilfælde med 2x2 og 3x3 pooling sker der en transformation af dataen, som forsøger at beholde væsentlig information. Det er dermed en balancegang for, hvor fin dataen der stilles til rådighed er, kontra den beregningskraft, der behøves for at håndtere dataen, der vil give den bedste accuracy, når vi ændrer på pooling-størrelsen.

Sammenligning med relateret arbejde

Det implementerede CNN havde ved tunede parametre en maksimal accuracy på 97.61, 91,88 og 74,76 for hhv. 3, 6 og 24 sprog. Dermed har det implementerede netværk opnået højere resultater end hvad [2], [4] og [3] opnåede (91, 72, 83). Det skal nævnes at vores data ikke som i [2] har været påvirket af yderligere støj fra optagelsen af lydfilerne. Dette er en væsentlig faktor, som med stor sandsynlighed ville resultere i lavere accuracy for vores implementering, hvis ikke der var taget højde for dette. På grund af manglende detajler er det ikke kendt om dette også er tilfældet for [4].

Derudover er det vigtigt at nævne, at der er en væsentlig forskel i inputtet, der benyttes. I [3] bruges spektrogrammer (billeder, hvor pixel intensiteten kan give information om dataen). hvor der derimod i vores løsning bruges MFCC format som i [4] og [2]. [3] kom ved test af 3 sprog på et time-delayed neural network frem til, at et netværk med flere lag fungerer bedre end et netværk med få. Disse resultater stemmer overens med vores løsning, der ligeledes indikerer, at flere lag giver bedre resultater. Dette er et tegn på at en forøgelse i antallet af lag giver potentiale for bedre resultater uanset typen af det anvendte neural network.

Kapitel 8

Konklusion

Vi præsenterede en løsning på et problem omhandlende talegenkendelse. Vi implementerede et convolutional neural network vha. frameworket Tensorflow og benyttede MFCC-data, der er en modificeret repræsentation af et effektspektrum udledt fra et datasæt af lydfiler udgivet af TopCoder. Vi udvalgte fem parametre, som blev varieret i et forsøg på at finde ud af hvilke af disse, der var med til at opnå den bedste accuracy. Vi justerede på MFCC-dataens format, antallet af convolutionlag, batch- og pooling-størrelsen samt dropout-sandsynligheden. Vores resultater viste, at vi med et CNN relativt nemt kunne løse talegenkendelseproblemet. Dertil viste resultaterne, at brugen af flere lag og mere detaljeret information om lydfilerne gav de bedste resultater. Det skal dog bemærkes at et CNN med et ektremt højt antal lag, eksempelvis over 100, måske ikke ville give gode resultater pga. at de features netværket ville finde ikke er mulige at generalisere over. Derudover fandt vi, at selvom dropout-sandsynligheden på 0,5 ikke altid gav det bedste resultat, så viste den sig at være den mest stabile. Her forstås stabil som værende et udtryk for, at accuracy ikke har en markant stor ændring, når antallet af lag forøges. Generelt havde batch-størrelsen ikke indvirkning på accuracy, men vi så, at en lille batch-størrelse i et komplekst netværk med mange sprog medførte, at early stopping indtraf og reducerede accuracy markant, fordi early stopping kriteriet er afhængigt af antallet af iterationer og ikke antallet af inputs, der trænes på. Dertil fandt vi, at en variering i pooling-størrelsen var et trade-off mellem netværkets kompleksitet og øget MFCC-format, hvilket resulterede i at begge pooling-størrelser var nyttige. Vi opnåede på 3, 6 og 24 sprog en accuracy på hhv. 97,61%, 91,88% og 74,76%, hvilket vi mener er gode nok, til at løsningen kan anvendes.

Fremtidigt arbejde

Ud fra de forskellige testresultater, var det meget tydeligt, at en stigende kompleksitet i netværket, i samspil med et forøget MFCC format, gav højeste resultat. Det er derfor oplagt at tilpasse netværket med yderligere lag for at se, om accuracy kan forbedres yderligere heraf.

Da der benyttes et relativt lille datasæt, giver det mening at teste det nuværende netværk med andre typer af datasæt. Det kunne f.eks. være dem benyttet i [5], da datasættet indeholder flere lydfiler fra samme sprog. En anden faktor er også mængden af støj i filerne. Ved at teste på flere forskellige datasæt fås en indikation af om løsningen er robust til klassificering af lydfiler.

Vi har desuden valgt at benytte MFCC-data i vores løsning. I [36] vises, at MFCC-delta-delta'er, som er MFCC's dobbelte afledte, giver bedre resultater. En undersøgelse af hvilken indflydelse brugen af disse eller andre features ville have i vores netværk kan formodentligt øge accuracy yderligere for vores løsning.

I implementeringen af vores CNN benyttede vi Rectified linear units (ReLU) som aktiveringsfunktion. En af grundene til dette, var at ReLU i andre studier har været brugt, samt at den ikke er dyr at beregne i læringsprocessen. Med en forøget mængde af kraft ønsker vi at se om andre aktiveringsfunktioner som Sigmoid og eller Tanh funktionen, der begge benytter eksponentialfunktionen, ville give et bedre resultat end det opnåede.

Under testfasen af vores CNN, blev der benyttet MAX-pooling. En implementering af en mere sofistikeret poolingstrategi, som foreslået i [15] ønskes testet. Den foreslåede metode viser sig at virke bedre på de benyttede datasæt. Dermed kan accuracy for vores løsning måske forøges ved at benytte samme pooling strategi.

Appendix A

Hadamardproduktet

Hadamardproduktet er en velkendt vektoroperation indenfor lineær algebra. Operationen benyttes ofte til at addere et antal vektorer eller multiplicere vektorer med matricer. Dele af litteraturen kalder også denne operationen for Schurproduktet. Symbolet, der benyttes for Hadamards produkt, er $\odot$. Nedenfor findes et simpelt eksempel for brugen af operatoren.

Lad a og b være to vektorer defineret ved:

$$a = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \qquad b = \begin{bmatrix} 2 \\ 6 \end{bmatrix} \qquad \text{Da er } a \odot b \text{ lig:}$$

$$\begin{bmatrix} 3 \\ 4 \end{bmatrix} \odot \begin{bmatrix} 2 \\ 6 \end{bmatrix} = \begin{bmatrix} 3 \cdot 2 \\ 4 \cdot 6 \end{bmatrix} = \begin{bmatrix} 6 \\ 24 \end{bmatrix}$$

Litteratur

[1] Google: Google Scholar
`https://scholar.google.com`

[2] De Mori, Julien, et al.
`"Spoken Language Classification CS 229 - Machine Learning."`
2012.

[3] Montavon, Grégoire.
`"Deep learning for spoken language identification."`
NIPS Workshop on deep learning for speech recognition and related applications, 2009.

[4] Nakashika, Toru, Christophe Garcia, and Tetsuya Takiguchi.
`"Local-feature-map Integration Using Convolutional Neural Networks for Music Genre Classification."`
INTERSPEECH, 2012.

[5] Lopez-Moreno, Gonzalez-Dominguez, et al.
`"Automatic Language Identification Using Deep Neural Networks"`
IEEE, 2014

[6] Kenneth Hardy
`Linear Algebra for engineers and scientists using MATLAB` (Kapitel 2.1)
Pearson Education, 2005

[7] Paul Dawkins
`Common Derivatives and Integrals, 2005`
Besøgt 7. Juni 2016

[8] Abu-Mostafa et al.
`Learning From Data`.
AMLbook.com, 2012.

[9] Neural Networks and Deep Learning, Michael A. Nielsen, Determination Press 2015
`http://neuralnetworksanddeeplearning.com/chap3.htmlsoftmax`
Opdateret Januar 2016, besøgt 10. april 2016

[10] Department of Computer Science DIKU: Machine Learning
`http://www.diku.dk/english/research/imagesection/machine-learning/`
Udgivet 2015, besøgt 21. februar 2016.

[11] Christoper M. Bishop.
`Pattern Recognition and Machine Learning`.
Springer Science + Business Media, LLC, 2006.

[12] Google: Google Self-Driving Car Project
`https://static.googleusercontent.com/media/www.google.com/en//selfdrivingcar`
Udgivet Januar 2016, besøgt 21 Februar 2016

[13] Hastie et al.
`The Elements of Statistical Learning (2nd Edition).`
Springer-Verlag, 2009.

[14] CS231n Convolutional Neural Networks for Visual Recognition.
`http://cs231n.github.io/convolutional-networks/`
Besøgt 15. maj 2016

[15] Matthew D. Zeiler and Rob Fergus,
`"Stochastic Pooling for Regularization of Deep Convolutional Neural Networks."`
ICLR, 2013.

[16] Softmax Regressions
`https://www.tensorflow.org/versions/r0.7/tutorials/mnist`
Besøgt 10. april 2016

[17] Visual Studio Magazine: Neural Network Cross Entropy Error
`https://visualstudiomagazine.com/articles/`
Udgivet april 2014, besøgt 10. april 2016

[18] Hawkins, Douglas M.
`"The problem of overfitting."`
Journal of chemical information and computer sciences 44.1, 2004

[19] Lei Jimmy Ba and Brendan Frey
`"Adaptive dropout for training deep neural networks"`
Advances in Neural Information Processing Systems, 2013.

[20] Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov,
"Dropout: A Simple Way to Prevent Neural Networks from Overfitting."
Journal of Machine Learning Research, 2014.

[21] Prechelt, Lutz
"Early Stopping - but when?"
Springer Berlin Heidelberg, 1998.

[22] Diederik P. Kingma, Jimmy Lei Ba
"Adam: A Method For Stochastic Optimization."
ICLR, 2015.

[23] TopCoder: How It Works
https://www.topcoder.com/community/how-it-works/
Udgivet 2015, besøgt 20. februar 2016.

[24] Faith Comes By Hearing: God's Word For Every Person
https://www.faithcomesbyhearing.com/
Udgivet marts 2000, besøgt 26. februar 2016.

[25] TopCoder: Marathon Match
https://community.topcoder.com/longcontest/
Udgivet 2015, besøgt 20. februar 2016.

[26] Mail, Forum. Personlig kommunikation. Februar 2016

[27] Machine Learning Mastery: Discover Feature Engineering, How to Engineer Features and How to Get Good at It
http://machinelearningmastery.com/discover-feature-engineering
Udgivet september 2014, besøgt 9. april 2016

[28] Practical Cryptography: Mel Frequency Cepstral Coefficient (MFCC) tutorial
http://practicalcryptography.com/miscellaneous/machine-learning
Udgivet 2012, besøgt 9. april 2016

[29] Vaseghi, Saeed V.
"Power Spectrum and Correlation."
John Wiley Sons Ltd, 2000.

[30] Logan, Beth.
"Mel Frequency Cepstral Coefficients for Music Modeling."
ISMIR, 2000.

[31] Khayam, Syed Ali.
"The discrete cosine transform (dct): theory and application."
Michigan State University 114 (2003).

[32] Cassidy, R., and J. Smith III.
"Auditory filter bank lab."
(2005).

[33] Tensorflow: About Tensorflow
https://www.tensorflow.org/
Besøgt d. 9 April 2016

[34] LibROSA
https://bmcfee.github.io/librosa/
Besøgt 10. April 2016

[35] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio.
"Deep Sparse Rectifier Neural Networks."
International Conference on Artificial Intelligence and Statistics. 2011.

[36] Md. Afzal Hossan, Sheeraz Memon, Mark A Gregory
"A Novel Approach for MFCC Feature Extraction."
Signal Processing and Communication Systems (ICSPCS), 2010 4th International Conference on. IEEE, 2010.