Kan man klone en stemme?

Navn:	Alexander Ziegler Simonsen
Skole:	EASJ / Zealand Roskilde
Fag:	Machine Learning
Vejleder:	Michael Claudius
Antal tegn:	23.691
antal sider:	23

2.400 tegn pr. side. Max 10 sider (24.000)

Indledning

Jeg valgt emnet “kloning af stemmer”, da jeg syntes at det er et super interessant spørgsmål og emne. Udover interessen, er der mange eksempler på digitale stemmer, eksempelvis Amazon Alexa og Google Assistant, som bliver brugt dagligt i mange forskellige scenarier.
Derfor er der helt klart et markedet for stemmer, som kan kopieres ved en god del optagelse, eller genereres alene ved man bare skrive noget tekst. Der er mange mulighederne for et sådan produkt, nogle få eksempler kan være: figurer i spil, chatbots som kan tale, bedre upersonlige telefonrådgivning, osv.

Motivation

Jeg har hørte om software som er i stand til at kopiere stemmer, som gør det muligt at sige hvad som helst med den stemme, ud fra kun 5 sekunders optagelse. Men det er en lidt mere advanced måde at gøre det på. Det her er et utroligt stort projekt, hvor jeg kommer til at møde rigtig mange udfordringer undervejs. Jeg ved at dette projekt er noget jeg kan lære rigtigt meget af, derudover bliver det ret sjovt at være i stand til at, få figuren **Geralt of Rivia** fra witcher spil serien, til at sige alle mulige useriøse ting.

Problemstilling

Jeg vil gerne undersøg om, det er muligt at lave et text-to-speech program, som kan tale med en bestemt stemme.
Mit **hovedspørgsmål** er **“kan man klone en stemme?”**, hvilke jeg vil prøve at besvare gennem mine delspørgsmål. Jeg har i alt 5 delspørgsmål:

- Hvilke datasæt er der til rådighed med lydoptagelser?
- Hvordan håndterer vi dataformatet af lydoptagelserne?
- Hvordan håndtere man lyd i Deep Learning?
    - Hvilke værktøjer findes der til TTS?
- hvad er CNN?
    - er CNN velegnet til TTS?
- Er det muligt at udarbejde et TTS ML program på 5 uger?

Metodisk tilgang

Jeg ved fra de forskellige løsninger der findes på markedet at, der skal bruges et datasæt med optagelser, bare for at få modellen til at lærer hvordan den taler engelsk. Derfor vil jeg bruge nogle af de eksisterende datasæt, som allerede er opbygget til lige dette formål, men det kommer jeg ind på lidt senere.

Eksperiment

Jeg har tænkt mig at arbejde med forskellige teknologier, for at finde frem til den der bedst opnår mit hoved mål.

Implementering

Der er mange dele der skal til for at genskabe den menneskelige stemme, så jeg får nok ikke nået at dykke ned i hver eneste del gennem dette projekt. Det system jeg kommer til at arbejde mest med er CNN (Convolutional Neural Network), som har vist sig at være rigtig god til at arbejde med TTS (Text to Speech).

Planlægning

Før jeg virkelig startede på projektet, lagde jeg en plan for alle 5 uger. Dette er dog kun en optimistisk vurdering, da jeg arbejder med vildt fremmede teknologier og jeg har aldrig arbejdet med lyd før. Derudover skriver jeg dagbog for hver dag, så det er nemmere at huske hvad jeg har foretaget mig gennem hele projektet. synopsen skriver jeg på løbende.

Ugeplan

uge	plan
0 (16-17)	Find datasæt og rense data
1 (18)	Opstilling af datasættet. Undersøg teori bag de forskellige teknologier.
2 (19)	Arbejde med de forskellige teknologier. Mere forståelse for teorien.
3 (20)	Opstilling af et MVP (minimum viable product), eller enkelte prototyper af de forskellige dele.
4 (21)	Sammenlign de forskellige teknologier med hinanden, for at finde frem til det bedste resultat
5 (22)	Finpudse synopsen og produktet

Forskning

Hvilke datasæt er der til rådighed med lydoptagelser?

Det mest kendte datasæt er kaldt **LibriSpeech ASR corpus**. Det er en samling af 1000 timers optagelse, fra forskellige engelske taler (både mænd og kvinder). Lyden er indsamlet fra public domain lydbøger, hvilke de har fra Librivox. Desværre er kvaliteten af de forskellige optagelser ikke den bedste, samt generelt meget forskellige i forhold til hinanden, siden hver taler fra Librivox er frivillige.

Et andet godt datasæt jeg fandt er CSTR VCTK Corpus, som er et højere kvalitet datasæt af 109 native engelsk taler, med rimelige forskellige accenter. Hver taler siger omkring 400 sætninger.

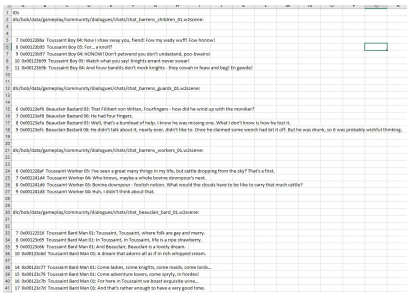
En tredje en som jeg fandt gennem min forskning var UrbanSound8K, hvilke faktisk ikke havde noget med stemmer at gøre. Jeg brugte den mest til at lære hvordan man arbejder med lyd i DL (deep learning), da der kun var tale om 10 typer af lydfiler. Det skal også siges at, dette var det eneste datasæt, som havde en brugbar fil over deres datasæt.

Til sidst har vi mit *uofficielle* datasæt, som jeg har anskaffet mig fra spillet witcher 3, da jeg indså at deres optagelser er af en meget høj professionel kvalitet. Heldigvis var der nogle som har lavet noget software, som trækker lydfilerne ud af spillet, da de filer ikke normalt er tilgængelig. Samtidige er der også genereret et Excel ark. Dette excel ark har 3 former af information:
hvilken spilfigur der taler lyd filens navn hvad der bliver sagt

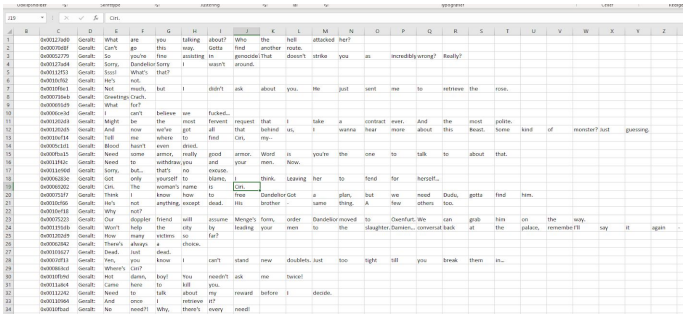
Dette er utroligt vigtigt for datasættets helhed, da det ellers ville være næsten ubrugeligt uden det. Sammenlagt er der lidt under 76 timer af optagelse fra alle figurer i spillet. Dog er det kun omkring lidt under 17 timer hvor hovedpersonen siger noget, så resten er jeg ikke interesseret i at arbejde med.

Hvordan håndterer vi dataformatet af lydoptagelserne?

Tæt på alle de datasæt jeg kunne finde, havde ikke en brugbar data fil, som gav overblik på alt dens data. Hvis jeg skal være i stand til at arbejde med et datasæt, skal jeg bruge en enkelt fil, der giver overblik over hele datasættet. Siden jeg ikke havde disse filer, endte jeg med at skrive flere former for små scripts, som kiggede igennem et datasæts mapper og produceret en datafil.



I tilfældet af mit *uofficielle* datasæt fra witcher spillet, var der utrolig meget arbejde jeg skulle igennem først. Excel arket med hver linje var ikke sat ret pænt op, så jeg skulle lave noget oprydning før det var brugbart. Først trak jeg alle de linjer ud jeg skulle bruge, for derefter at opdele hver celle ud fra mellemrummet (alt teksten var lagt i en celle). Efter jeg havde alt dataen opdelt, skrev jeg et python script der kunne samle alle cellerne efter de 2 første columns, så jeg kun havde et felt for hele replikken udtalt.



Dette gav mig en datafil over mit datasæt, som indeholdt 2 værdier, nemlig navnet på filen og dens tilsvarende udtalte linje. Da jeg nu vidste hvilke filer jeg skulle bruge, ud fra de sammenlagte 63.871, skrev jeg et python script, som rykkede filerne ind i en anden mappe. Her oplevede jeg nogle få problemer, der betyde at jeg måtte omskrive mit script lidt hver gang. Først kunne jeg ikke overføre en fil, da filen lå i mappen i forvejen, dette var grundet at Excel arket genbrugte nogle lydfiler flere gange. Her fjernet jeg bare alle gentagelser fra den datafil jeg arbejde med.

Det næste problem var at, jeg prøvede at rykke på filer, som faktisk ikke fandtes lokalt.
Her satte jeg bare scriptet til kun at rykke på filer der fandtes, men samtidige også holde styr på de filer som ikke fandtes, for derefter at skrive dem til en fil.
Da det var overstået havde jeg kun faldet fra 20.156 til 19.367 filer. Nu havde jeg en datafil der gav klart overblik over mit uofficielle datasæt, som bestod af 3 columns:
navn på filen replikken i tekst længden af replikken

Billederne på den næste side fremviser koden for det script, som rykket på alle filerne og lavet en datafil over mit datasæt.

```python
import json
import os

# list of filenames
files = []
# list of missing files
missing = []

# we will need this later in the program
jsonData = []

# lets get the list of files
# this part maybe have to be rewritten, if your json data isn't the same
with open('newLines.json') as lines:
    data = json.load(lines)
    for f in data:
        files.append(f['filename'])
        jsonData.append(
            {"filename": f['filename'],
             "dialog": f['dialog'],
             "length": len(f['dialog'])
             })

obj = json.load(open('lines.json'))

# check that it worked
# for f in files:
#    print(f)

# file end for every file
ending = ".wav.ogg.wav"
```

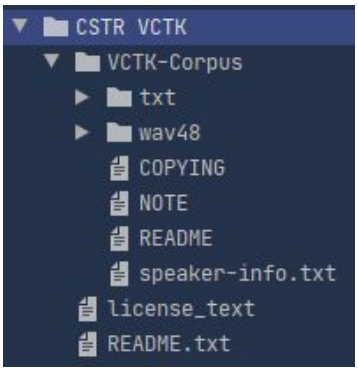
```python
# path
# E:\ML sypnosis\audio\_____sounds\wav\g_lines
path = "E:/ML sypnosis/audio/_____sounds/wav/g_lines/"

# just to see if it works (should always show 1)
# files.append("test")

# checking if the filename have a file connected to them
for f in files:
    if not os.path.exists(path + f + ending):
        # # uncomment the print line when testing
        # print(path + f + ending)
        missing.append(f)

print("there are: ", len(missing), " missing files")

# write the missing files to a txt file
with open('missing.txt', 'w') as outfile:
    json.dump(missing, outfile)

# count before
print("count of json array: ", len(obj))

items = []
# get all the elements as json objs, so we can remove them after
for i in range(len(obj)):
    if obj[i]["filename"] in missing:
        items.append(obj[i])

for i in items:
    obj.remove(i)
```

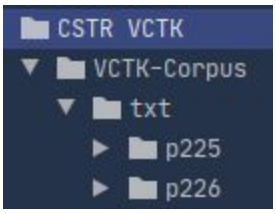
```python
print("count of json array: ", len(obj))

# sort it by the len
print("typeof:", obj)


# make a new json file, without the missing files
with open('newLinesTest.json', 'w') as outfile:
    json.dump(jsonData, outfile)
```

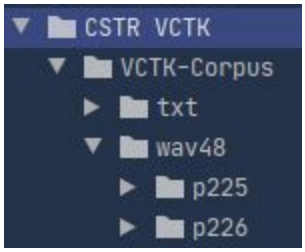
Selv om jeg kun havde tænkt mig at bruge dette script til mit (witcher) datasæt, endte det med at være noget jeg brugte på stort set alle mine datasæt. Der var ikke ret meget af dette script, som blev ændre mellem de forskellige datasæt. Derfor vil jeg ikke dække de andre udgaver i dette dokument.

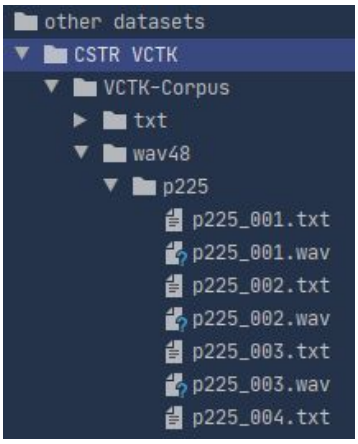
Det næste datasæt jeg gik gang med var **LibriSpeech ASR**, som gav mig utrolig mange problemer. Men jeg vil ikke gå i dybden med det, da det lidt ville være en gentagelse af det jeg har skrevet i forvejen.

Derefter arbejdede jeg med **CSTR VCTK Corpus (fremover CVC)** datasættet. Deres mappe opbygning var generelt nemmere at arbejde med, i forhold til det fra **LibriSpeech ASR** datasættet.







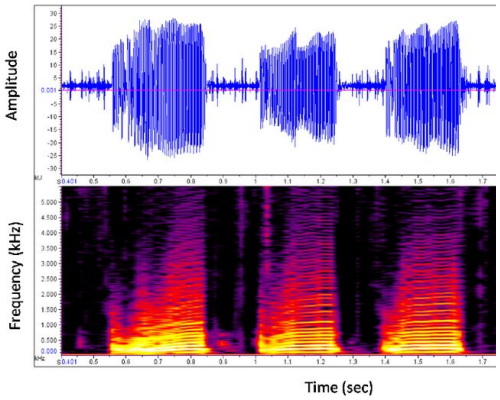


Der var ens mappe struktur i både ***txt*** og ***wav48***, så jeg overskred mapperne fra den ene med den anden. Det vil sige at alle filerne forbundet til hinanden, lå indenfor den samme mappe. Dog gav **CVC** datasættet mig lidt problemer, eksempelvis ved deres dårlig datafil over de forskellige talers accent. Grundet den lidt klodsede måde dataen var stillet op, var det ikke helt nemt at adskille accents og region, der var endda også nogle som ingen region havde. Dette betød først og fremmest at, jeg måtte gå ind og ændre på mellemrummet manuelt i filen. Da der nu var et fikst mellemrum mellem hver element, var jeg i stand til at skrive et script, som håndteret hver linje som en lang tekst string, hvor den så opdelte værdierne ud fra bestemte index punkter.
Nu var jeg i stand til at ligeledes skabe en datafil for dette datasæt.

Til sidst har vi **UrbanSound8K** datasættet, hvor jeg gik i gang med at lave en datafil over datasættet. Dog fandt jeg ud af det ikke var nødvendigt, da det var det eneste datasæt med en fungerende datafil. Kort sagt er UrbanSound8K datasættet en samlinge af lyde, som falder under 10 grupper:

1. air conditioner
2. car horn
3. children playing
4. dog bark
5. drilling
6. engine idling
7. gun shot
8. jackhammer
9. siren
10. street music

Hvordan håndtere man lyd i machine learning?



For at lydoptagelserne kan bruges til noget, skal de transformeres til et format, der kan fortolkes af vores machine learning (ML) modeller. ML arbejder meget bedre med billeder, derfor udnytter vi python biblioteket **libROSA** til at konvertere vores lydfiler til et spektrogram.
Hvis man kigger på billedet til højre, så er toppen hvordan lyd filer normalt er opstillet. For at vi kan arbejde med det, skal vi konvertere det til et spectrogram (bunden af billedet), som er en anden måde at fremvise lydbølger på.

Hvilke værktøjer findes der til Text to Speech?

De fleste TTS (text-to-speech) løsninger der findes på markedet, deler det op i nogle mindre dele, som hver har sit ansvar. De fleste har en 3-dels-opbygning, som består af:
Encoder, decoder og vocoder/converter.

Encoder håndtere det data vi har fra et datasæt, bearbejder dataen og konvertere det til noget vi kan bruge i resten af systemet. Der er mange måder på at gøre det på, men jeg fik ikke dykket dybt ned i dem. Det jeg fik arbejdet med i dette projekt var **libROSA,** som laver lyden om til spektrogrammer.
Decoder er lidt svære at forklare, da det er her man udnytter de forskellige deep learning (DL) teknikker, som lære fra de spektrogrammer man bruger. Her kan stort set alle de store ML/DL biblioteker bruges (TensorFlow, pyTorch, Theano, CNTK). Gennem hele mit projekt, har jeg dog kun arbejdet med TensorFlow (GPU udgaven) og Keras, samt brugt Anaconda til at håndtere alle software pakkerne nødvendige for projektet.
Vocoder tager hvad den har lært fra decoder delen og skaber nye lydfiler. Her findes der flere måder, dog vil jeg nævne de 2 jeg stiftede kendskab med gennem dette projekt. Den første er en løsning google har lavet, kaldet WaveNet, som bliver brugt i forskellige forskningsprojekter, et eksempel er Deep Voice 3. Derudover findes der også en anden vocoder, kaldet WaveRNN som bliver brugt i forbindelse med ForwardTacotron.

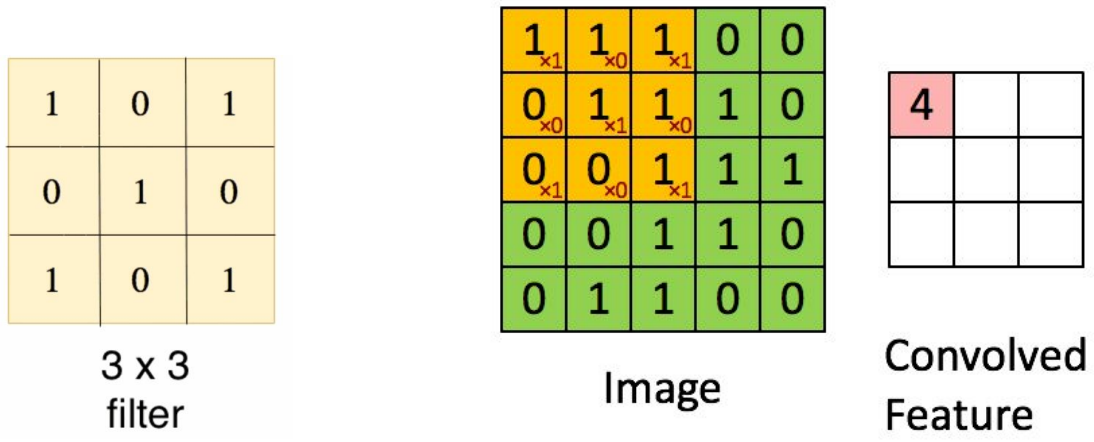
Ud over de nævnte teknologier findes der flere forskningsprojekter, som arbejder med at udarbejde bedre TTS. Disse forsker er meget ivrige for at udgive deres resultater, samt hvordan de er været i stand til at opnå dem. Det giver os muligheden for at skabe skabeloner, ud fra deres forsikringspapirer, på hvordan man skal sætte TTS programmet op.

Hvad er Convolutional neural network?

CNN (Convolutional neural network) er en af de bedste former for DL netværks, med en meget høj nøjagtighed. CNN består af flere lag, eksempler er: Max Pooling, Dense, Dropout, Convolutional, Flatten.

Convolutional layers

Den vigtigste del ved et CNN er det Convolutional layer. Dette layer bruger et filter, der er mindre end vores input billede, som så bliver kørt langs hele billedet.
hvis vi tager *figur 6.1* som et eksempel, tager vi filteret til venstre, starter fra det venstre top hjørne på billedet og går en pixel til højre. Dette fortsætter indtil filteret rammer det højre hjørne. Efter den har gået fra hjørne til hjørne, gør den det igen en pixel længere nede, indtil filteret har gået det hele igennem. Filteret tager værdien fra filterets ene pixel, ganger det med den underliggende værdi fra billedet og til sidst lægger de 9 tal sammen. Det resulterer i en enkelt værdi i vores convolved feature map.
Så vi har et billede på 5x5 pixel, med et filter på 3x3, som så giver os 9 celler i en 3x3 matrix.

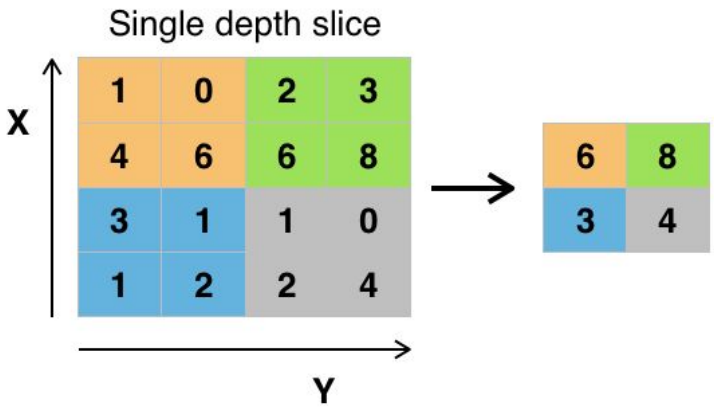


figur 6.1

Resultatet fra dette layer giver os et feature map, som senere bliver håndteret af et andet Convolutional layer (der vil generelt være mere end et Convolutional layer i et CNN).
CNN håndtere dataen i en 2D matrix (ligesom det fremvises på figur 6.1), hvilke er en af de største forskel på CNN i forhold til andre typer af neural networks (NN). Det er også derfor at CNN har opnået et så godt resultat, i forhold til de andre typer af NN.

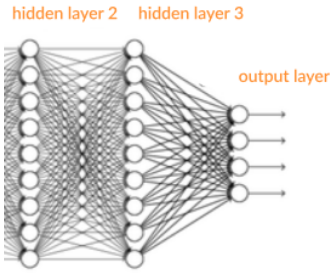
Max Pooling layer

pooling laget bruger et filter, som minder meget om det der bliver brugt i Convolutional laget. Når man bruger max pooling, får man kun det højeste tal, hvilke kan ses på billedet til højre. På den måde kan vores netværk lærer hurtigere, da der nu er mindre værdier den skal holde styr på.
Hvis vi kigger på billedet til højre, kan vi se at filteret er sat til 2 x 2, samt har en stride på 2. Stride kan forstås som offsettet mellem filter checks, hvilke kan sættes til en unik værdi både vandret og lodret.



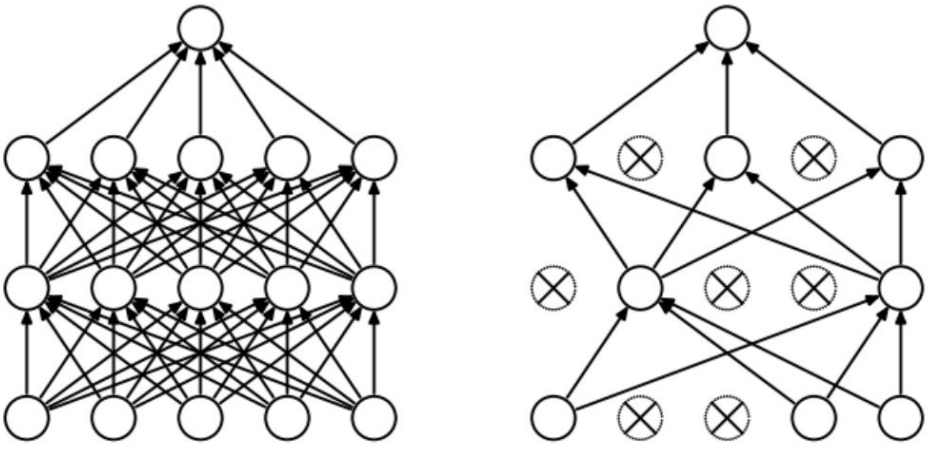
Dense layer

Dense er mere kendt som fully connected layer, dog bliver det kaldt Dense i Keras biblioteket. Når et lag er fully connected betyder det at hver neuron er forbundet til alle neuroner på det næste lag. Normalt er det kun nogle få lag i vores netværk, som er sat til dense.



Dropout layer

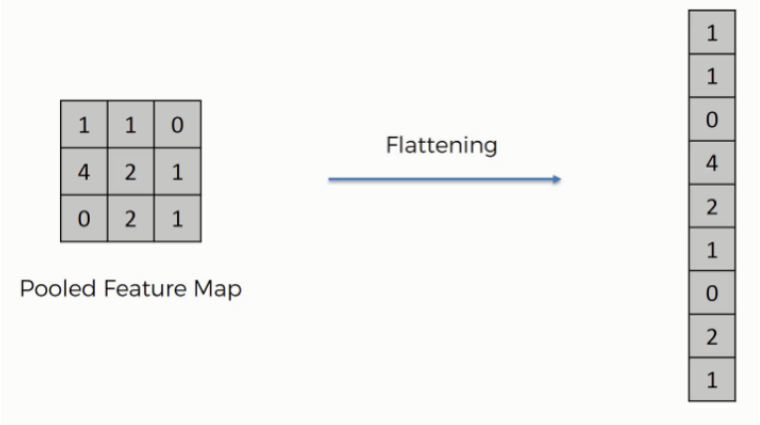
Når vores netværk er sat helt op, har vi et hav af parameter, som alle er med til få modellen til at lærer fra vores data. Dog kan der var nogle af disse parameter, som kan forekomme unødvendig for modellens indlæring. Dette er noget som er umuligt for os at finde frem til, men hvis vi benytter et dropout layer, vil vores netværk finde frem til de parameter, som den vurderer som unødvendig. Billedet her fremviser hvordan det virker:



(a) Standard Neural Net (b) After applying dropout.

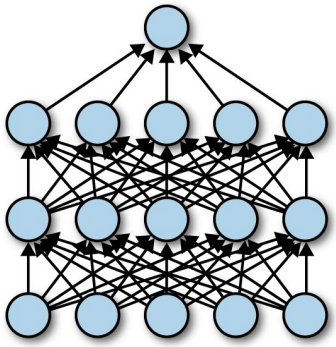
Flatten layer

Hele ideen bag et CNN, er at det kan håndtere vores billeddata i en format, som er tættere på vores input data (32x32x3)(16x16x15). Men når det kommer til de sidste stadier af vores netværk, er vi nødt til at konvertere vores data, ned til en 1 dimensional liste af værdier, for at modellen kan lærer fra dataen.



Er CNN velegnet til TTS?

Normalt ser man ikke CNN blive brugt i forbindelse med lyd, da det generelt kun arbejder med billeder. Men som vi nævnte tidligere ved ***Hvordan håndtere man lyd i Deep Learning?***, kan man konvertere lyd filen til et spektrogram billeder, der så kan håndteres af vores CNN. Gennem denne form for processe, har CNN givet et hurtigere og bedre resultat end andre neural networks, eksempelvis RNN (Recurrent neural network) som blev brugt før i tiden til TTS forskning.
Siden vores lyd filer bliver konverteret til spectrogram billeder, er det ret vigtigt at alle lydfilerne er stor set af samme længde, da det ellers ikke vil være nemt for vores CNN at arbejde med datasættet.
Det er dog muligt at sætte vores model op, på en sådan måde at, det kan håndtere alle former for input længder. Dette kræver dog at stort set hele vores CNN er *fully connected/dense*.


(a) Standard Neural Net

Så umiddelbart vil sige at, det er det bedste NN til TTS forskning indtil videre.

Mine forsøg

En af de første ting jeg skulle få styr på, var at få mit TensorFlow til at arbejde med min GPU. Dette var ikke så svært at sæt op, da de har lavet en rigtig god guide på deres hjemmeside.

Jeg undersøgt de forskellige teknologier til at lave kopier af stemmer, hvor jeg fandt ud af at der var nogle eksisterende løsninger på markedet. Der er googles WaveNet, samt googles Tacotron 2, derudover har vi MelNet og til sidst har vi deep voice 3 (DV3) udarbejdet af det kinesiske firma Baidu. Jeg valgte at arbejde med DV3 (som faktisk bruger WaveNet som en del af deres vocoder). Dette valg er grundet at, DV3 havde den laveste mængde tid krævet for indlæring, samt også havde nogle af de bedste resultater.

Jeg har brugt rigtig meget tid på at undersøge hvordan DV3 fungerer, men jeg var ikke i stand til at genskabe deres løsning. Jeg valgte i stedet for at, arbejde med den ene af de 2 eksisterende løsninger jeg kunne finde på nettet. jeg valgte at arbejde med deepvoice3_pytorch, som var en åben implementering af DV3 i pytorch. På deres github side, kan man finde et eksempel, hvor de fremviser hvordan det virker med en allerede trænet model. Deres eksempel virker dog ikke, uden man først skal ændre på koden først, det skyldes at løsningen er skrevet med TensorFlow (1.X). Jeg fik det til at virke på colab, efter at jeg havde fikset de små problemer, men jeg var ikke helt i stand til at få det til at virke lokalt på min pc. Dette er et absolut krav for hele mit projekt, da de forskellige lyd datasæt jeg har lokalt, fylder mange gigabyte og jeg har ikke rigtig mulighed for at opload til colab, grundet en begrænset mængde harddiskplads. Jeg havde ingen erfaring med pytorch, så jeg gik videre til en anden løsning.

Jeg endte med at kigge på nogle andre løsninger end DV3, da jeg indså det var alt for nyt teknologi, til jeg kunne opnå ret meget med det inde fra min tidsramme. Så jeg kigget på de andre løsninger.

Det sidste jeg prøvet at arbejde med var ForwardTacotron, som er en anden udgave af googles Tacotron2, hvor de har udskiftet WaveNet med WaveRNN, som er en helt åben vocoder. Desværre bruger løsningen et software kaldet **espeak**, som ikke rigtig virker på windows, da det stort set kun findes på linux eller mac systemer. Jeg forsøgte med at dual boot min windows pc med et linux system, men jeg var meget presset på tid på det tidspunkt, så jeg droppede hele denne løsningen efter en dag.

Det fungerende program

Samtidige med at jeg prøvet alle de løsninger jeg lige har beskrevet, arbejdet jeg med et lille program, for at finde ud af hvordan man arbejder med lyd i et CNN. Det er her jeg har brugt **UrbanSound8K** datasættet, da det er meget simpelt og kun har 10 typer af lyde.
Først importere jeg de layer typer jeg skal bruge.

```
import keras
from keras.layers import Activation, Dense, Dropout, Conv2D, \
                         Flatten, MaxPooling2D
from keras.models import Sequential
```

```
from ipywidgets import IntProgress
from IPython.display import display
import time
```

```
import librosa
import librosa.display
import numpy as np
import pandas as pd
import random
import tensorflow as tf
import os
```

Ipywidgets, IPython.display og time er kun noget jeg bruger ved et plugin, som viser en process bar når man loader noget.
Resten af dem behøver ikke en forklaring.

For at få mit grafikkort til at virke med TensorFlow, skal jeg bruge disse 2 linjer, ellers giver det mig problemer (det er vist kun Nvidia RTX korts som har det her problem).

```
# these 2 lines fixes some GPU problems I have had.
physical_devices = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

```
# path to dataset
ds_location = 'E:/ML sypnosis/other datasets/UrbanSound8K/audio/'
```

Derudover skal vi bruge en sti til hvor datasættet ligger lokalt på min pc.

Vores datasæt har 8 columns, men der er kun 4 columns vi skal bruge:

- slice_file_name
- fold
- classID
- class

```
# Read Data
data = pd.read_csv('UrbanSound8K.csv')
data.head(5)
```

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

```
In [6]: data.shape
Out[6]: (8732, 8)

In [7]: # only get data over 3 seconds long                    ("not" limited to 3000)
        valid_data = data[['slice_file_name', 'fold' ,'classID', 'class']][ data['end']-data['start'] >= 3 ] #[:3000]
        valid_data.shape
Out[7]: (7468, 4)
```

vi er dog kun interesseret i de filer som er 3 sekunder lang eller længere, da det vil give et mere konkret datasæt at arbejde med. Derfor sætter vi en anonymous function op, som checker om hvert element er over eller lig med 3 sekunder. Antallet falder faktisk kun fra 8732 til 7468, så det er helt okay. Dette er den anonymous function vi bruger:

```
[ data['end']-data['start'] >= 3 ]
```

Jeg vil gå igennem et eksempel, hvor jeg tager en enkelt lyd fil og konvertere det til et spektrogram. Først loader vi lydfilen og decoder den som en tid serie (**y**), der bliver opbevaret som en 1 dimensions NumPy floating point array. Samtidige får vi også sampling rate af y, som vi gemmer i **sr** variablen.

```
# Example of a Siren spectrogram
y, sr = librosa.load(ds_location + 'fold6/135160-8-0-0.wav', duration=2.97)
```

Som man kan se på billedet til højre, bliver de 2 værdier givet til librosa for at oprette et spektrogram.

```
# this line gives an output of 32x32
ps = librosa.feature.melspectrogram(
    y=y,
    sr=sr,
    S=None,
    hop_length=2048,
    n_mels= 32,
    window='hann',
    center=True,
    pad_mode='reflect',
    power=2.0)
```

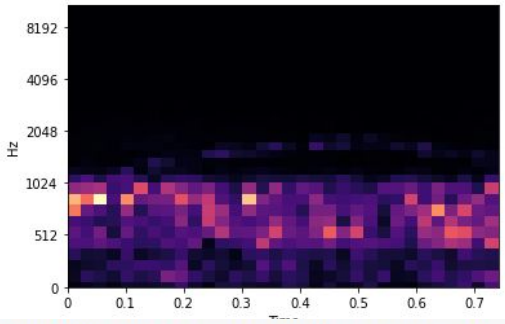
```
print(type(ps))

# the output (32, 32) is (y, x)
print("ps shape:", ps.shape)
# ziegler
print("y, x")
print("y:", len(y))
print("sr:", sr )
```

```
<class 'numpy.ndarray'>
ps shape: (32, 32)
y, x
y: 65489
sr: 22050
```

Hop_length er med til at bestemme mængden af værdier på y aksen, for vores endelige spektrogram.
Talværdien (2048) er et tal som bliver divideret med lydfilens sample rate (**sr**), hvilke resultatere i 32 værdier på y aksen.
n_mels er meget nemmere at arbejde med, da man bare sætter mængden af ønsket værdier på x aksen.
Af en eller anden grund står y værdien først i vores output format (32,32) (y,x).
Resten af parametrene er ikke så vigtige, da det for det meste er deres default værdier.

```
# (log scale mal) spectrogram of siren clip
librosa.display.specshow(ps, y_axis='mel', x_axis='time')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x271707f5388>
```



Derefter kan vi fremvise vores spektrogram, som kan ses på billedet til højre. Det var meget vigtigt at få kvaliteten ned på 32x32 pixel, da spektrogrammerne ellers ville haft opbrugt utroligt meget ram.

Dette er de linjer der er med til at, få vores model til at kunne klassificere dataen senere. Dette bliver forbundet til datasættets column kaldet “classID”.

```
y_train = np.array(keras.utils.to_categorical(y_train, 10))
y_test = np.array(keras.utils.to_categorical(y_test, 10))
```

Jeg har 3 udgaver af min model, som jeg fremviser over de 3 næste sider. Dog er det kun mængden af neuroner der ændres, så strukturen på dem er identisk.

Derfor vil jeg prøve at forklare strukturen her, så jeg ikke behøver at gøre det på de næste sider. Jeg starter med et input format af (32,32,1) (X, Y, Z).
Det første tal på Conv2D og Dense layer'erne, står for længden af Z aksen.
Det er kun MaxPooling, Flatten og Dropout lagene som ikke har en activation funktion forbundet til sig.
Lagene er bygget op i denne rækkefølge:

```
# version 3 - many layers, high amount of neurons # best about 0.96 ac
input_shape=(32, 32, 1)
model1 = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same", input_shape=input_shape),
    keras.layers.MaxPooling2D((2)),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D((2)),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
    ])
```

1. Conv2D
2. MaxPooling2D
3. Conv2D
4. Conv2D
5. MaxPooling2D
6. Conv2D
7. Conv2D
8. MaxPooling2D
9. Flatten
10. Dense
11. Dropout
12. Dense
13. Dropout
14. Dense

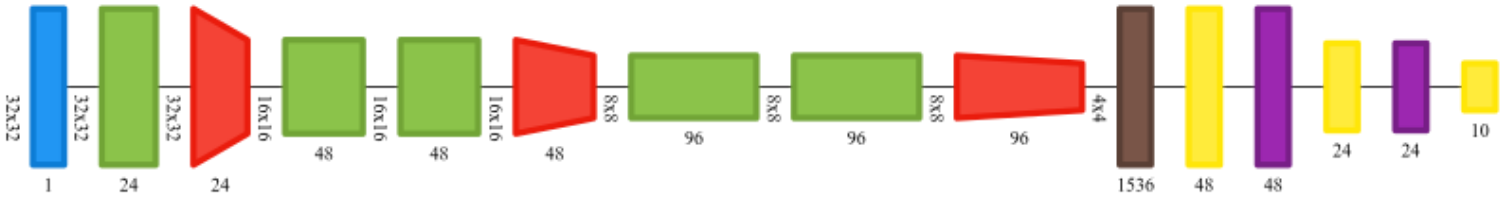
```
model3.compile(
    optimizer="Adam",
    loss="categorical_crossentropy",
    metrics=['accuracy'])

# this model don't really work with epochs at
model3.fit(
    x=X_train,
    y=y_train,
    epochs=80,
    batch_size=32,
    validation_data= (X_test, y_test))

score = model3.evaluate(
    x=X_test,
    y=y_test)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Det sidste Dense lags har en activation function sat til "softmax" typen. Det betyder at netværket prøver at få hvert element til at passe ind i en af de 10 varianter.

Diagrammerne for de forskellige modeller, som jeg fremviser på de 3 næste sider, følger alle denne color-coding:

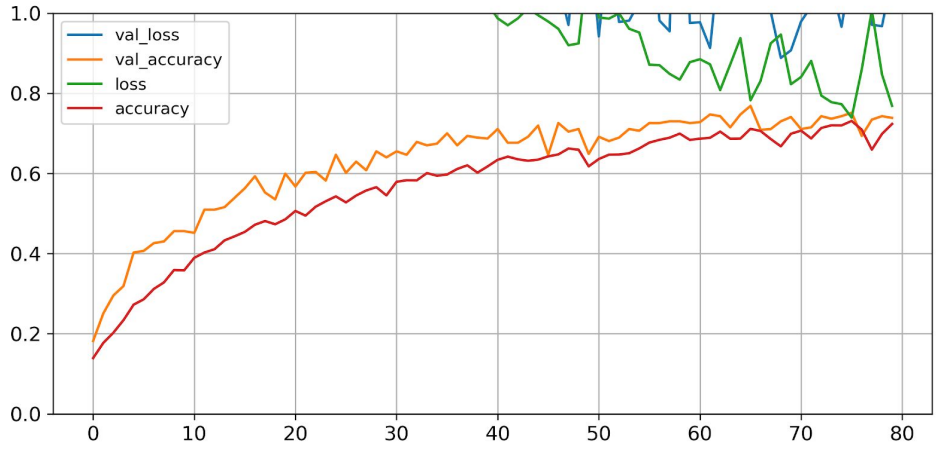
Disse diagrammer er lavet med et online software kaldet Net2Vis.
Alle 3 modeller er sat til 80 epochs.dd

model version 2



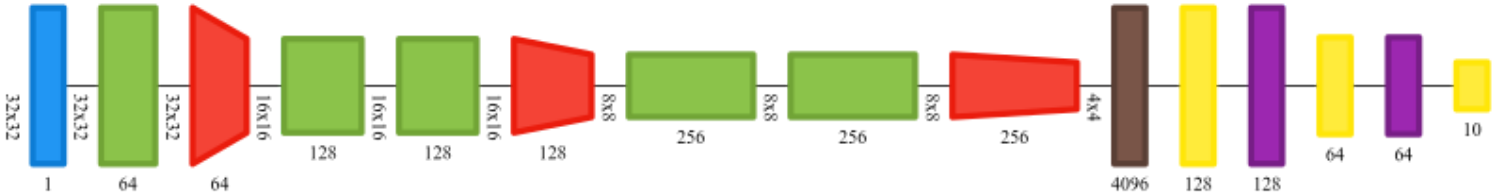

```
In [20]: # version 2 - many layers, low amount of neurons # best about 0.85 ac

input_shape=(32, 32, 1)
model = keras.models.Sequential([
    keras.layers.Conv2D(24, 7, activation="relu", padding="same", input_shape=input_shape),
    keras.layers.MaxPooling2D((2)),
    keras.layers.Conv2D(48, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(48, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D((2)),
    keras.layers.Conv2D(96, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(96, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(48, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(24, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
    ])
```

Hver epoch tog omkring 1.17 sekunder.



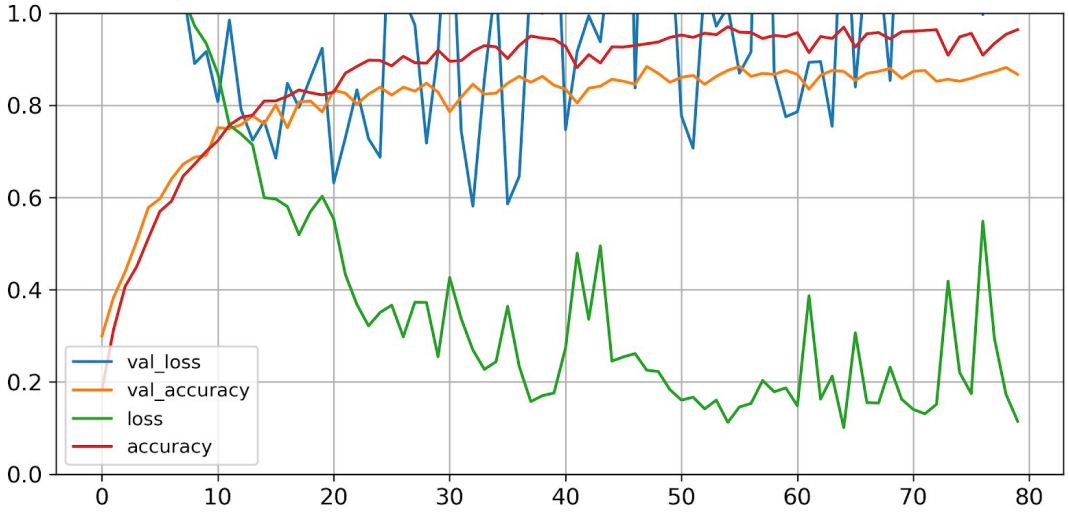
Denne model gav kun omkring en nøjagtighed på omkring 80% - 85%

model version 3 (min bedste model)




```
In [21]: # version 3 - many layers, high amount of neurons # best about 0.96 ac
input_shape=(32, 32, 1)
model1 = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same", input_shape=input_shape),
    keras.layers.MaxPooling2D((2)),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D((2)),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
    ])
```

Hver epoch tog omkring 2.24 sekunder.

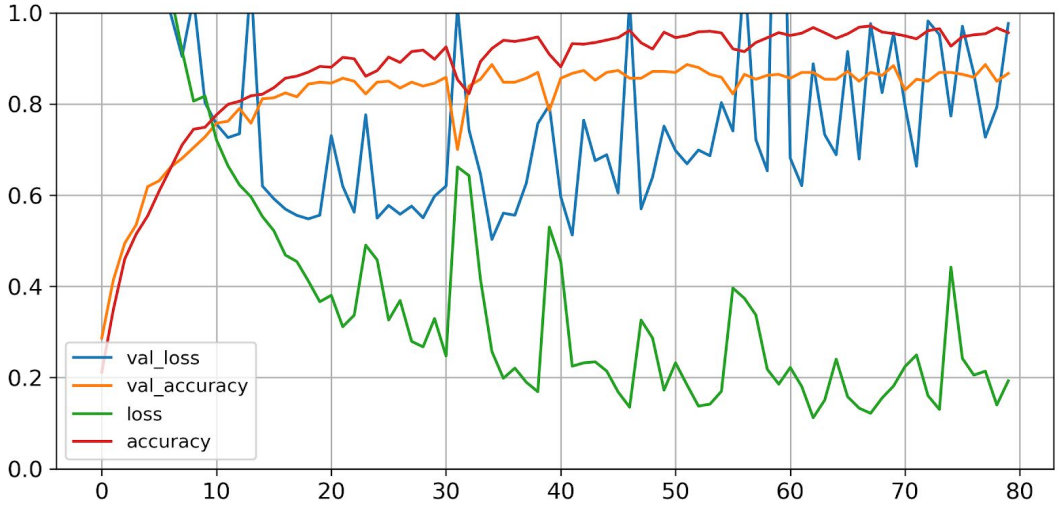


Denne model gav omkring en nøjagtighed på omkring 90% - 96%

model 4 - test forsøg


```
input_shape=(32, 32, 1)
model2 = keras.models.Sequential([
    keras.layers.Conv2D(128, 7, activation="relu", padding="same", input_shape=input_shape),
    keras.layers.MaxPooling2D((2)),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D((2)),
    keras.layers.Conv2D(512, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(512, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(256, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
    ])
```

Hver epoch tog omkring 3.48 sekunder.



Denne model gav omkring en nøjagtighed på omkring 90% - 96%
Så der er virkelige ingen grund til at fordoble mængden af neuroner fra model 3.

Konklusion

Hvilke datasæt er der til rådighed med lydoptagelser?
Der er rigeligt med datasæt til rådighed (LibriSpeech ASR corpus, CSTR VCTK Corpus, UrbanSound8K), gennem dette projekt har jeg skabt mit eget uofficielt datasæt, så det er ikke ret svært at finde noget brugbart.

Hvordan håndterer vi dataformatet af lydoptagelserne?
Gennem dette projekt har jeg oplevet at, stort set ingen datasæt har en enkelt brugbar fil der giver overblik over deres datasæt. De har dog informationen delt over flere filer, hvor det så bliver muligt at lave en datafil for hele datasættet. Så det er helt klart muligt at, fremstille en datafil over de forskellige datasæts.

Hvordan håndtere man lyd i Deep Learning og hvilke værktøjer findes der til TTS?
Hvis vi tager mit program som et eksempel, konverterer man lydfilen til et spektrogram, for at vores CNN kan klassificerer lyden ud fra de 10 typer som datasættet har. I dette tilfælde brugte jeg libROSA som min **encoder**.
Decoder delen består af det netværk man opstiller, ud fra de ML/DL biblioteker der findes i forvejen (TensorFlow, pyTorch, Theano, CNTK). **Vocoder** delen kan opfyldes af WaveNet eller WaveRNN.
Så længe du ved hvad du har behov for, kan man helt klart finde et eksisterende programmer der opfylder ens behov.

Hvad er CNN og er det velegnet til TTS?
Convolutional neural network kan håndtere data i en 2D eller 3D matrix format, hvilke forbedre dens indlærings resultat i forbindelse med billeder. Når dataen bliver bearbejdet på en måde så vores CNN kan forstå det, så er der ikke et andet NN som giver lige så gode resultater.

Er det muligt at udarbejde et TTS ML program på 5 uger?
Jeg vil sige ja, som udgangspunkt så er det muligt, da der er mange kilder af viden på emnet tilgængelige på nettet, men det er ikke lige til. Det vil kræve et godt kendskab til ML/DL på forhånd, for at være i stand til at arbejde med det. Hvis jeg havde en stærkere grundlæggende forståelse for ML/DL, er jeg okay sikker på at, jeg ville haft opnået dette inde for projektets tidsperiode af 5 uger.

Alt er dog ikke tabt, da det arbejde jeg har fået lavet, dannet grundlag for min forståelse og udvikling af et program der anvendes CNN og klassifikation af lyd. Programmet opnået en nøjagtighed på omkring 90% - 96% genkendelse.

Refleksion

Gennem dette projekt, har jeg arbejdet hårdt med emner som har været meget fremmed for mig. Det har gjort mig opmærksom på følgende områder, som jeg kan forbedre til næste gang. Jeg havde ikke lagt en god nok tidsplan for mit arbejde, hvilke har resulteret i at, jeg har brugt alt for meget tid på nogle dele. Derudover havde jeg været alt for fastlåst på DV3 løsningen, som der desværre ikke findes ret mange eksempler på, da det er noget af det nyeste forskning inde for TTS. Det ville haft gavnet mig meget, hvis jeg i stedet for lavet nogle små **Spikes** (kort hurtige *product testing,* kendt fra XP) for de forskellige løsninger, som jeg kunne haft brugt i dette projekt.

Jeg er dog ret stolt over det jeg har nået, da jeg er kommet rigtig langt, jeg er blevet klogere på hvordan man behandler data, samt har lært hvordan man foretager forskning på højt tekniske emner.
Jeg har tænkt mig at arbejde videre med dette projekt efter eksamen, da jeg føler jeg er rigtig tæt på. Jeg opnået at klargøre de datasæt, som jeg skulle bruge i forbindelse med indlæringsprocessen. Jeg har gjort mig bekendt med hvordan TTS fungerer. Jeg er blevet opmærksom på alle de populære eksisterende TTS løsninger, som kunne bruges for at opnå mit mål.

Litteraturliste

datasets

- witcher 3 lyd pakke (indeholder også et Excel ark over filerne og linjerne udtalt fra hver fil) - https://www.nexusmods.com/witcher3/mods/3058
- LibriSpeech ASR corpus - http://www.openslr.org/12
- CSTR VCTK Corpus - https://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html
- URBANSOUND8K - https://urbansounddataset.weebly.com/urbansound8k.html

løsninger

- MelNet - https://sjvasquez.github.io/blog/melnet/
- Deepvoice3_pytorch - https://github.com/r9y9/deepvoice3_pytorch
- google colab eksempel af Deepvoice3_pytorch - https://colab.research.google.com/github/r9y9/Colaboratory/blob/master/DeepVoice3_single_speaker_TTS_en_demo.ipynb
- min fikset udgave af eksemplet af Deepvoice3_pytorch - https://colab.research.google.com/drive/1QYA0281znNzT66fSoRIMo3lHLNRCp2uc?usp=sharing
- tacotron2 - https://github.com/NVIDIA/tacotron2
- ForwardTacotron - https://github.com/as-ideas/ForwardTacotron

forsikringspapirer (søg efter "pdf" på siderne)

- Deep Voice 3: Scaling Text-to-Speech with Convolutional Sequence Learning - https://arxiv.org/abs/1710.07654
- WaveNet: A Generative Model for Raw Audio - https://arxiv.org/abs/1609.03499
- MelNet: A Generative Model for Audio in the Frequency Domain - https://arxiv.org/abs/1906.01083

andet info

- diagram over deep voice 3 - https://miro.medium.com/max/1400/1*XUEIsQJz8HedKpo30HvK8g.png
- (vocoder) WaveRNN - https://github.com/fatchord/WaveRNN
- GPU TensorFlow guide - https://www.tensorflow.org/install/gpu

biblioteker og/eller software

- LibROSA - https://librosa.github.io/librosa/
- TensorFlow - https://www.tensorflow.org/
- pytorch - https://pytorch.org/
- theano - http://deeplearning.net/software/theano/ (aldrig brugt, kun nævnt)
- CNTK - https://github.com/microsoft/CNTK (aldrig brugt, kun nævnt)

software

- anaconda - https://www.anaconda.com/
- Net2Vis (fremstil diagrammer af ML/DL modeller) - https://viscom.net2vis.uni-ulm.de/

andet

- **Geralt of Rivia - https://en.wikipedia.org/wiki/Geralt_of_Rivia**
- (software til at trakke lyden ud af spillet) Extracting-Voice-Over-Audio-from-Witcher-3 - https://github.com/Gizm000/Extracting-Voice-Over-Audio-from-Witcher-3
- guide på hvordan man bruger det - https://youtu.be/YAm2r4wCfSg

Bilag

- Bevis på at jeg er ordblind
- Koden fra mit program