종합설계 최종보고서

- 음성명령으로 제어하는 스마트 홈 -

Rolling Stones

경영정보학과/ 컴퓨터공학과	201311868	고주현
컴퓨터공학과	201514207	김영선
응용통계학과/ 컴퓨터공학과	201511655	문예지
경영학과/ 컴퓨터공학과	201212088	이용주
지도교수	박능수	(인/서명)

종합 설계 최종 보고서
- 음성 명령으로 제어하는 스마트홈

고주현, 김영선, 문예지, 이용주
건국대학교 경영학과/컴퓨터공학부, 건국대학교 컴퓨터공학부, 건국대학교 응용통계학과/
컴퓨터공학부, 건국대학교 경영학과/컴퓨터공학부
jhko2@naver.com, withbbang@gmail.com, yjmbetty@gmail.com,
didix.yong@gmail.com

Graduation Project
- Home controlled by voice command

Joohyeon Ko, Youngsun Kim, Yeji Mun, Yongju Lee
School of Management, Konkuk University
School of Computer Engineering, Konkuk University
School of Applied Statistic, Konkuk University
School of Management, Konkuk University

요 약

딥러닝/머신 러닝의 비약적인 발전으로 음성인식(Speech-to-text), 음성합성(Text-to-Speech), 자연어 처리(Natural Language Processing)을 활용한 음성 AI 비즈니스는 영어회화, 음성주문, 번역기 등 생활과 교육 전반의 다양한 서비스 모델이 개발되고 있다. 본 프로젝트는 머신러닝/딥러닝 기반 기술을 활용하여 음성 AI기반의 생활 편의 및 시설 관리 서비스를 구현하고자 한다. 특히, 구글 데이터셋 내에 존재하는 6가지의 데이터를 머신러닝 알고리즘을 활용하여 학습하고, 그 결과에 따라 GUI를 제어하는 것을 목적으로 한다. 현대시대에 더 많은 편의를 위한 서비스 제공을 목적으로 하였으며 사람의 단순 음성을 하드웨어가 인식할 수 있도록 명령어셋으로 바꾸어 집안의 가전소품을 제어할 수 있는 서비스를 구현한다. 이를 통해, 팀원들의 음성인식, 딥러닝에 대한 기본지식을 배양하고 활용할 수 있는 응용 능력을 향상시키고자 한다.

1. 서 론

딥러닝/머신러닝의 비약적인 발전으로 음성인식(Speech- to-text) 음성 합성(Text-to-Speech), 자연어 처리(Natural Language Processing)을 활용한 음성 AI 비즈니스는 영어회화, 음성주문, 번역기 등 생활과 교육 전반의 다양한 서비스 모델이 개발되고 있다. 그리고 최근 빅데이터, 인공지능 분야가 주목받으면서 컴퓨터와 사용자의 원활한 의사소통을 위하여 음성 인식 기술 분야 연구가 가속화되고 있으며, 2011년 iOS에 탑재된 Siri를 시작으로 다양한 음성 인식 시스템이 상품화되어 출시되고 있다.

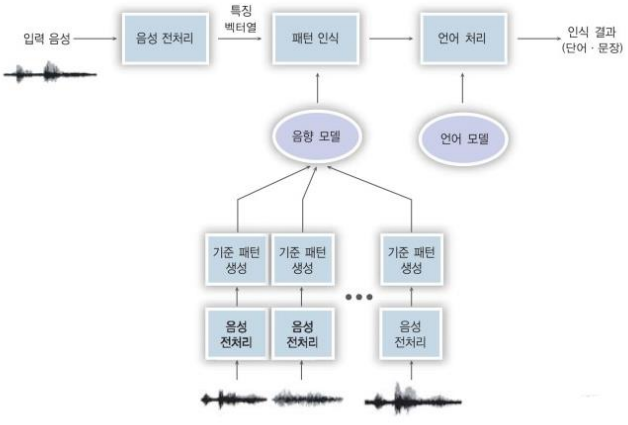


그림 1.1. 음성인식 프로세스

이 논문에서는 다양한 음성 처리방법과 음성 인식 딥러닝 모델 중에서 MFCC모델과 CNN모델을 사용한 프로젝트를 구현해보는 것을 목표로 한다. 먼저, 영어 음성 명령어를 구글에서 다운 받은 후, 음성 명령어 데이터 셋을 MFCC방법으로 전처리한다. 전처리한 음성명령어를 Training set으로 두고, CNN모델과 Python의 Tensorflow 라이브러리를 활용하여 Deep Learning model을 만든다. 저장된 모델을 바탕으로 적중률을 측정하고, 모델이 추론한 음성명령을 GUI에 반영한다. 추가적으로 다른 모듈을 포함하여 다른 서비스를 제공할 수도 있다. 음성 명령을 사용한 간단한 조작에서 더 나아가 IoT 기반으로 한 가전제품 조작 등 다양한 목적으로 상용화 될 수 있다. 최근 각광받는 분야인 머신러닝을 실생활에 접목하여 활용함으로써 on/off와 같은 단순한 명령뿐만 아니라 복잡한 명령어로 폭넓은 서비스를 제공하는 기반이 될 것이다

본 논문의 구성은 다음과 같다. 서론의 두번째 부분에서는 프로젝트에 대해서 간략하게 설명 한 후, 2장에서 프로젝트와 관련된 배경지식과 기술동향에 대해서 알아볼 것이고, 3장에는 프로젝트의 전체적인 개발과정과 세부적인 과정을 설명할 것이다. 그리고 4장에서는 프로젝트의 결과와 성능에 대해서 분석할 것이고, 5장에서 프로젝트에 관한 결론을 맺는다.

2. 프로젝트 배경지식

2.1. 음성인식

2.1.1. 음성인식 개요

전처리는 분석을 위한 기본이다. 쓰레기를 넣으면 쓰레기가 나온다. (garbage in, garbage out)는 것은 머신 러닝의 기본 전제다. 올바른 학습 모델을 얻기 위해서는 올바른 데이터를 입력해야 하므로, 우수한 학습 알고리즘을 설계하는 것만큼이나 충분히 가다듬어진 데이터를 확보하는 것이 중요하다. 고도의 기술과 올바른 절차를 따르더라도 잘 정제된 데이터가 확보되지 않으면 왜곡된 모델이 나올 수 있으며, 이는 학습 결과에 대한 신뢰도를 떨어트림과 동시에 전체 프로젝트의 진행 또한 보장 할 수 없게 된다.

이처럼 머신러닝에서 학습의 전제가 되는 소스 데이터는 굉장히 중요한 역할을 한다. 학습을 할 수 있는 데이터를 만드는 모든 과정을 전처리 라고 하는데 여기에는 이미지 혹은 음성을 분석, 혹은 학습 할 수 있도록 데이터로 바꾸는 과정 또한 포함한다. 그리고 이 과정에서 어떤 방법을 통해 데이터화 하는지가 전체 학습과정에 지대한 영향을 끼칠 정도로 인공지능, 머신러닝에서 데이터의 전처리는 굉장히 중요한 역할을 가진다.

음성인식 머신러닝에서 데이터의 전처리 방법은 데이터를 어떻게 가공하는지에 따라 학습 모델의 능률이 달라지기에, 어떤 종류의 음성인식 기술을 구현할지에 따라 전처리 방법이 달라지는 경우가 많다. 아래에 음성인식의 부류에 관해 기술하였다.

2.1.2. 음성인식 기술

음성인식 기술의 분류

종류	명칭	설명
발성의 형태	고립단어인식 (isolated word recognition)	문장이 아니라 단어 단위로 인식하는 것
	연결단어인식 (connected word recognition)	여러 개의 단어를 연결시켜 발성한 것을 인식하는 형태
	핵심어검출 (keyword spotting)	자연스럽게 발성한 연속된 음성 중에서 인식 대상 단어만을 추출하여 인식
	연속음성인식 (continuous speech recognition)	자연스럽게 발성한 문장형태의 연속된 음성을 인식할 수 있다. 가장 난이도가 높은 단계
화자 특성	화자종속인식 (speaker-dependent speech recognition)	특정 화자 또는 사용자가 자신의 음성으로 미리 인식기를 훈련시키는 방식
	화자독립인식 (speaker-independent speech recognition)	미리 수백 또는 수천 명의 음성에 관한 정보를 추출하여 데이터베이스화함으로써 별도의 훈련 과정 없이 어떤 사용자라도 사용하도록 하는 것이 목적
	화자적응 (speaker adaptation)	화자종속 및 화자독립 방식을 절충한 것으로 특정 화자가 화자독립 인식기를 사용하는 경우 자신의 목소리에 대한 인식률을 높이기 위해 화자독립 인식기를 자신의 목소리에 적응시키는 방식

그림 2.0. 음성인식 기술

음성인식 분야에 대한 연구는 1950년대부터 시작하여 지금까지 짧지 않은 기간 동안 이루어지고 있다. 그러나 최고 선도그룹에서 개발된 인식기조차 기술 수준 면에서 많은 한계를 가지고 있다.

음성인식의 궁극적인 목표는 인간과 마찬가지로 자연스럽게 발성되는 모든 음성을 인식할 수 있는 수준이어야 하지만 기술적인 한계로 인하여 응용 환경에 따라 음성인식기를 여러 단계로 나누어 개발하고 있다. 발성의 형태에 따라 음성의 인식 엔진을 다음과 같이 분류할 수 있다.

1) 고립단어인식(isolated word recognition)

문장이 아니라 단어 단위로 인식하는 것으로 한 단어씩 떨어져 있는 형태 때문에 고립단어인식이라고 한다. 이 방식은 인식대상 단어가 고정되어 있는

고정단어인식과 사용자가 응용분야에 따라 인식 대상 단어를 갱신할 수 있는 가변단어인식의 두가지 형태로 나누어 볼 수 있다.

음성 인식의 가장 초보적인 단계이며 현재 가장 많이 상용화되어 있는 형태이다. 명령어 인식, 단독 숫자음 인식, 또는 영어 알파벳 인식 등을 대표적인 예로 들 수 있다. 수십 단어급에서 수십만 단어급까지 다양한 형태가 있으며, 비교적 조용한 환경에서 인식률은 90% 이상의 수준이다.

2) 연결단어인식(connected word recognition)

여러 개의 단어를 연결시켜 발성한 것을 인식하는 형태이다. 대상 단어가 제한되더라도 단어들의 조합으로 다양한 발성을 다룰수 있으며, 고립단어인식에 비해 유연한 인터페이스를 제공할 수 있다. 고립 단어인식에 비해 난이도가 높으며 인식률이 낮다. 대표적인 예가 연속숫자 인식(connected digit recognition)으로, 현재 부분적으로 상용화되고 있다.

3) 핵심어검출(keyword spotting)

자연스럽게 발성한 연속된 음성 중에서 인식 대상 단어만을 추출하여 인식한다. 문장형태의 발성을 그대로 인식하는 것은 성능면에서 한계가 있으므로 이를 극복하고 특정한 분야로의 적용을 위해 제안된 방식이다. 사용자 입장에서는 자연스럽게 말할 수 있어 더욱 친숙한 인터페이스의 느낌을 가질 수 있다.

열차, 비행기 자동 예약 시스템 등에서 사용자가 발성한 여러 가지 정보 중에 지명에 해당하는 것만을 알고 싶을 경우 이러한 방식을 이용하며, 상품 예약, 자동 콜센터 등에도 이용이 가능하다. 현재 부분적으로 상용화가 이루어지고 있다.

3) 연속음성인식(continuous speech recognition)

자연스럽게 발성한 문장형태의 연속된 음성을 인식할 수 있다. 가장 난이도가 높은 단계이며 음성신호처리 기술뿐만 아니라 언어처리 기술도 요구하고 있어 음성타자기 등 문법형식에 맞는 발성의 인식에 주로 적용되고 있다.

자유 대화 형태의 발성은 언어적 문법을 적용하기가 까다롭고 음향학적 발성형태도 매우 다양하여 인식에 어려움을 주고 있다. 인식 대상 단어의 크기는 수만 단어급에서 수십만 단어 또는 무제한 어휘로 구분할 수 있다.

또 다른 기술적 한계는 화자특성에 따른 음성인식률의 차이이다. 특정화자만을 대상으로 하여 높은 인식 성능을 제공하는지, 비교적 무난한 인식성능을 가지며 임의의 화자 누구나 사용할 수 있는지에 따라 다음과 같이 분류할 수 있다.

5) 화자종속인식

특정 화자 또는 사용자가 자신의 음성으로 미리 인식기를 훈련시키는 과정이 필요하다. 이 경우 인식기는 훈련된 화자의 음성을 다른 화자에 비해 더 정확히 인식할 수 있다. 목소리가 비슷한 사용자의 음성이라도 성능이 저하될 가능성이 있다. 비교적 구현이 간단하여 단말기 등에 탑재되어 응용되고 있으나 사용자가 훈련하는 과정을 거쳐야 하는 불편함이 있으므로 제한된 분야에만 사용된다.

6) 화자독립인식

미리 수백 또는 수천 명의 음성에 관한 정보를 추출하여 데이터베이스화함으로써 별도의 훈련 과정 없이 어떤 사용자라도 사용하도록 하는 것이 목적이다. 화자종속 인식기에 비해 구현이 어려우나 현재 대부분의 상용화 시스템은 이 방식을 이용한다.

7) 화자적응(speaker adaptation)

화자적응 방식은 화자종속 및 화자독립 방식을 절충한 것으로 특정 화자가 화자독립 인식기를 사용하는 경우 자신의 목소리에 대한 인식률을 높이기 위해 화자독립 인식기를 자신의 목소리에 적응시키는 방식이다. 이를 위하여 사용자는 인식기를 사용하기 전에 인식기가 요구하는 소량의 발성을 하는 과정을 거쳐야 한다.

2.2. 음성인식 알고리즘

2.2.1. MFCC

음성신호처리 분야에서 Mel-frequency cepstrum (MFC)은 단구간 신호의 파워스펙트럼을 표현하는 방법 중 하나로, 비선형적인 Mel스케일의 주파수 도메인에서 로그파워스펙트럼에 코사인변환 (cosine transform)을 취함으로써 얻을 수 있다. Mel-frequency cepstral

coefficients (MFCCs)는 여러 MFC들을 모아 놓은 계수들을 의미한다.

MFCC와 일반적인 캡스트럼의 차이는 일반적인 캡스트럼의 경우 주파수 밴드가 균등하게 나누어져 있는 반면 MFCC의 경우 주파수 밴드가 Mel-scale에서 균등하게 나누어진다는 것이다. Mel-scale로의 주파수 워핑은 소리를 더욱 잘 표현할 수 있는 장점이 있다. 따라서 오디오압축 등에서 사용된다.

MFCC는 일반적으로 4단계를 거쳐서 구할 수 있다.

1단계, 소리 신호를 적절한 프레임 길이로 잘라준다.

2단계, Mel-scale Filter Bank를 이용하여 에너지 스펙트럼을 구한다

3단계, 그 값에 로그를 취한다.

4단계, 로그가 취해진 Filter Bank 에너지에 DCT를 계산한다.

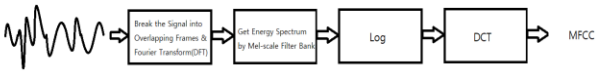


그림 2.1. MFCC의 3단계

1단계는 시간영역에서의 들어온 소리신호를 주파수 분석에 신뢰도를 높이기 위해 적절한 프레임의 길이로 잘라준다. 일반적으로 MFCC에서는 20~40ms로 하고 있다. 거기에 Fourier Transform(DFT)를 취하여 준다. 단일 프레임에 DFT를 취하여 주면 다음 식과 같다.

$$S_i(k) = \sum_{n=1}^{N} s_i(n)h(n)e^{-j2\pi kn/N} \qquad 1 \le k \le K$$

그림 2.2. 단일 프레임에 DFT를 취한 식

Si(n)는 s(n)이라는 시간영역에서의 신호를 n개의 정해진 샘플 수로 자르고, i는 프레임의 번호를 나타낸 것이고, h(n) : n개의 샘플을 가지는 윈도우이며, K는 DFT이 길이다. 이 결과를 이용하여 Periodogram-based power spectral Estimate을 구하면 다음과 같다.

$$P_i(k) = \frac{1}{N}|S_i(k)|^2$$

그림 2.3. Periodogram-based power spectral Estimate

2단계로 위의 결과에 Mel-scale Filter Bank를 이용하여 에너지 스펙트럼을 구한다. 각 백터의 자기 스펙트럼에 해당하는 일부 구간을 제외한 다른 주파수 영역에 대해서는 모두 0의 값을 가지게 된다.

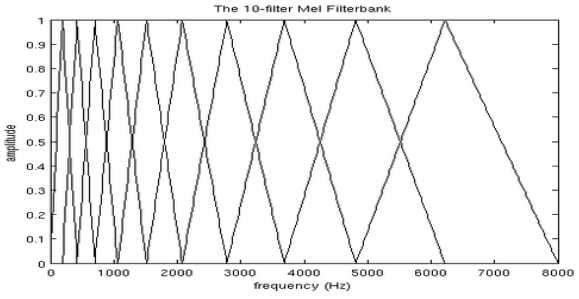


그림 2.4. Mel-Filter Bank

3단계로 위의 에너지가 넘어오면 그 값에 로그를 취한다. 이는 인간이 실제로 듣는 것과 유사하게 소리의 특징 정규화하는 과정이다.

4단계로 로그가 취해진 Filter Bank 에너지에 DCT를 계산하면 원하는 결과를 낼 수 있다.

2.2.2. CNN

CNN은 Convolutional neural network의 줄임 말로 Convolutional Layer(합성곱 계층)을 이용하는 딥러닝 알고리즘이다. CNN은 이미지, 음성인식 분야에서 많이 활용된다. CNN은 신경망을 블록과 같은 형태로 조합하여 만들며 합성곱 계층과 더불어 Pooling Layer(풀링 계층)이 사용된다.

기존의 Affine, ReLu로 이루어진 신경망과 달리 ReLu의 앞 뒤로 Conv layer(합성곱 계층)과 Pooling layer가 붙게 된다. 이러한 신경망 층은 출력 계층에서만 이전과 동일하게 affine 계층과 softmax계층으로 이루어지게 된다.

일반적인 Fully connected layer(완전연결 신경망)은 이미지 데이터의 처리시 3차원의 데이터들 혹은 배치파일 시 4차원의 데이터들을 모두 1차원으로 변경해야 한

다. 그렇기에 정보의 손실이 일어나며 특히 공간적인 개념에 있어서 데이터를 다루기 어렵다는 단점을 갖고 있다. 그에 반해 CNN 즉 합성곱 계층의 경우 형상을 유지한 채로 입력 받고 전달하기 때문에 더 효율성 높고 정확성 있는 학습이 가능하다.

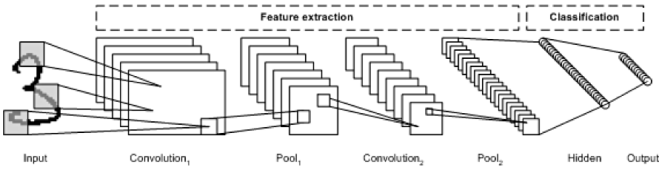


그림 2.5. Convolutional Layer을 이용하는 CNN 알고리즘

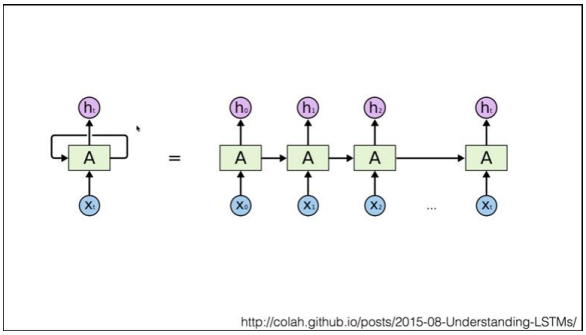
CNN은 합성곱 연산을 처리한다. 이들은 이미지 처리에서 말하는 필터 연산에 해당되는데 이때 필터에 해당되는 값이 완전연결 신경망에서의 가중치 매개변수가 되고 추가적으로 편향도 주어질 수 있다. CNN에 있어서 가장 다른 점은 패딩(Padding)과 스트라이드(stride)이다. 패딩은 합성곱 연산을 수행하기 위해 입력된 값의 주위를 0으로 채워 필터값과의 합성곱 연산 이후 출력 크기를 조정할 목적으로 사용된다. 즉 4,4입력데이터에 3,3필터를 적용할 경우 패딩이 없다면 3,3데이터를 출력하게 됨으로써 출력크기가 달라지는데 이 때 1만큼 패딩을 통해 출력 데이터값을 4,4로 조정하게 된다. 스트라이드는 합성곱 연산 시 필터를 적용하는 위치의 간격을 의미한다. 스트라이드가 1이면 필터 적용 윈도우가 1칸씩 이동하게 되는데 이 스트라이드 역시 패딩과 마찬가지로 출력크기를 조절하게 된다. 즉 Padding, Stride, Pooling, Filter의 요소 값에 따라서 출력 값의 형태가 조정될 수 있다.

CNN은 앞서 말하였듯이 다차원의 합성곱 연산을 지원하는데 이때에는 필터의 채널 수와 입력 데이터의 채널 수가 일치해야 한다. 이러한 다수의 필터의 합성곱을 통해서 완전연결 신경망과는 달리 배치, 블록 형태로 처리가 가능하다.

이미지 데이터의 경우 2-DIRECTION으로 처리하나 음성파일 및 1차원의 데이터의 경우 1-DIRECTION으로 처리하기도 한다. 2D의 경우 컴퓨팅 파워를 더 소모하나 평균적으로 더 좋은 성능을 발휘한다.

2.2.3. RNN

RNN(Recurrent Neural Network, 순환신경망)는 Sequential data를 처리하는 모델로 순서대로 처리하는 것을 뜻한다. RNN이 기존의 뉴럴 네트워크와 다른 점은 '기억' 즉, hidden state를 가지고 있다는 점이다. 네트워크에서의 hidden state는 입력데이터와 결과 값을 요약한 정보이고, 새로운 입력이 들어올 때 hidden state를 수정하게 된다. 입력을 전부 처리했을 때, hidden state는 Sequential data 전체를 요약하는 정보가 되는 것이다. 이 정보를 바탕으로 새로운 입력을



이해하고, 이 과정이 반복되기 때문에 Recurrent하다고 말할 수 있다. 이러한 특성 때문에 길이가 긴 Sequential data도 처리할 수 있게 된다. 음성인식, 자연어처리의 경우 단어 하나하나를 학습한다 해도 전체 문맥을 이해하기 어려운 경우가 많은데 이 경우 앞 뒤 문맥과 함께 학습할 수 있는 것이 RNN이다.

그림 2.6. RNN 학습 다이어그램

위 다이어그램에서 볼 수 있듯이 X는 입력, A는 hidden state, h는 출력을 나타낸다. X0가 들어갔을 때, 첫번째 A가 만들어지고, X1이 들어왔을 때 기존의 A와 X1을 참고하여 새로운 A를 만들게 된다. RNN은 이 과정을 반복하여 요약된 정보를 바탕으로 출력을 만들어 낸다. 이러한 반복되는 처리를 통해 이전 단어들을 보고 다음 단어가 나올 확률을 계산하는 모델에 주로 사용되며 자동번역, 자연어 문장 생성 등의 텍스트생성, 언어

모델링에 적용될 수 있다. 또, 시퀀스 길이에 관계 없이 입력과 출력을 받아들일 수 있어서 필요에 따라서 다양한 구조를 만들어낼 수 있다.

2.2.4. LSTM

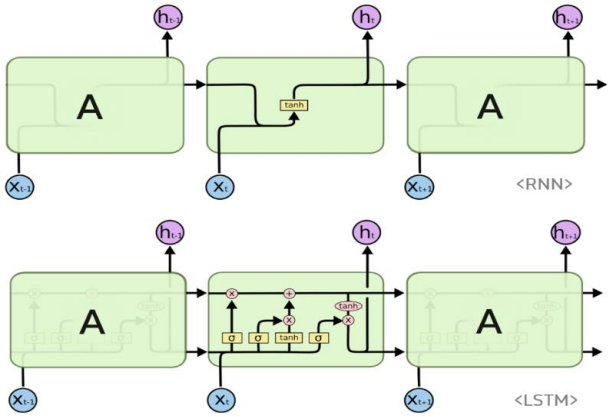


그림 2.7. RNN Hidden State

RNN을 응용한 모델들이 많이 있긴 하지만 LSTM은 그 중 제일 탁월하다. RNN은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 역전파 그레디언트가 줄어들어 학습능력 저하를 보인다. 이 문제를 극복하기 위해서 고안된 것이 바로 LSTM이다. LSTM은 RNN의 히든 state에 cell-state를 추가한 구조이다.

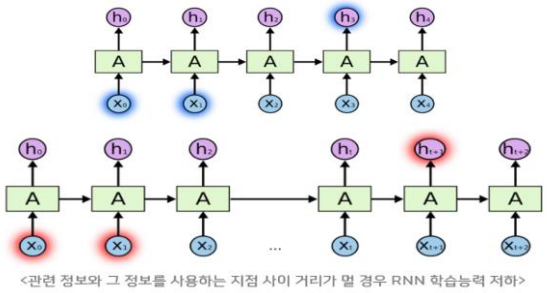


그림 2.8. LSTM의 정보 간 거리가 먼 경우

LSTM의 핵심은 메모리셀과 셀스테이트이다. LSTM은 뉴런 대신 메모리 셀이라는 구조를 이용해서 은닉층은 계산한다.

LSTM의 구성요소로 전체적인 큰 구조인 메모리 셀이 있으며 전반적인 처리과정을 담당하는 셀스테이트 그리고 내부적으로 시그모이드 함수를 통해 흐름을 담당하는 게이트가 존재한다.

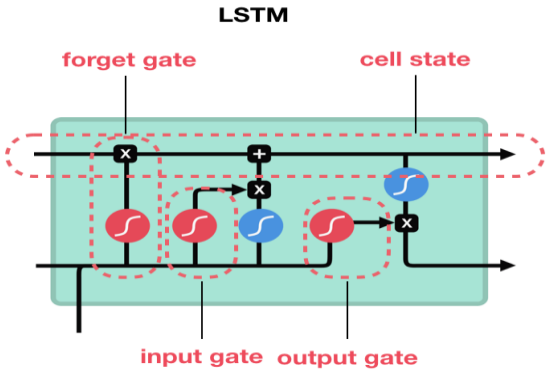


그림 2.9. Cell state 내부의 시그모이드 함수

셀스테이트 내부의 게이트는 기존 정보를 버릴지 말지 유무를 결정한다. 게다가 게이트는 시그모이드를 이용하여 선택적으로 새로운 정보를 추가 및 삭제 기능도 가지고 있다.

LSTM은 이러한 방법으로 기존 정보들을 꾸준히 갱신하며 장기 의존성을 해결했다. LSTM은 가중치를 곱셈으로 처리하지 않고 덧셈으로 처리하기 때문에 Gradient Vanishing문제를 해결할 수 있다.

2.3. IoT 기술 동향

2.3.1. IoT 개요

사물인터넷(Internet of Things, IoT)은 각종 사물에 센서와 통신 기능을 내장하여 인터넷에 연결하는 기술 즉, 무선 통신을 통해 각종 사물을 연결하는 기술을 의미한다. 인터넷으로 연결된 사물들이 데이터를 주고받아 스스로 분석하고 학습한 정보를 사용자에게 제공하거나 사용자가 이를 원격 조정할 수 있는 인공지능 기술이다. 여기서 사물이란 가전제품, 모바일 장비, 웨어러블 디바이스 등 다양한 임베디드 시스템이 된다. 사물인터넷에 연결되는 사물들은 자신을 구별할 수 있는 아이피를 가지고 인터넷으로 연결되어야 하며, 외부 환경으로부터의 데이터 취득을 위해 센서를 내장할 수 있다. 모든 사물이 해킹의 대상이 될 수 있어 사물인터넷의 발달과 보안의 발달은 함께 갈 수밖에 없는 구조이다.

정보 기술 연구 및 자문회사 가트너에 따르면 2009

년까지 사물인터넷 기술을 사용하는 사물의 개수는 9억 여개였으나 2020년까지 이 수가 260억 개에 이를 것으로 예상된다고 한다. 시스코 시스템즈의 조사에 따르면 2013년부터 2022년까지 10년간 사물인터넷이 14조 4천억 달러의 경제적 가치가 있을 것이라고 예상한다

2.3.2. 기술 구성

사물인터넷을 구축하기 위한 기술적으로 필요한 부분은 크게 사물신원확인, 의사소통이 가능한 네트워크 구축, 사물에 감각 부여, 컨트롤 가능성으로 나누어 볼 수 있다.

사물 신원 확인. 사물인터넷에 참여하는 각각의 개체는 다른 개체로 하여금 스스로를 식별할 수 있게 해주는 신원이 필요하다. 넓은 범위의 네트워크 상에서 개별 사물의 신원을 확인하기 위해서는 개별 사물에 IP주소를 부여해야 한다. 이에 따라 IP주소에 대한 수요는 증가하였고 기존에 존재했던 32비트인 IPv4 체계로는 증가하는 사물들의 주소를 모두 할당하는 데 어려움이 따른다는 한계가 나타났다. 이로 인해 128 비트인 IPv6 체계의 필요성이 대두되고 있다. 다만, 이는 각 말단 사물 개체들이 인터넷 망에 직접 접속할 때를 전제로 하는 조건이며, 가정, 기업 등의 로컬 네트워크에 위치한 말단 사물들이 사설 IP주소를 이용하여 로컬 서버를 경유해 인터넷으로 정보를 주고받는 경우에는 수요가 급격히 많아지는 것을 방지할 수 있다. 한편, 인터넷공급사업자인 ISP들이 가입자에게 제공하는 공인 IP주소를 제한하고 있고, 추가 할당을 위해서는 비용을 지불해야 하기 때문에 개별 가입자가 다수의 말단 사물을 공인 IP주소를 사용하여 인터넷에 바로 연결하는 것은 비용상 타당성이 없는 한계가 있음을 감안해야만 한다.

네트워크 구축. 사물들은 스스로가 취합한 정보를 필요에 따라 다른 사물과 교환, 취합함으로써 새로운 정보를 창출할 수 있어야 한다. 사물끼리의 일관된 정보전달 방법을 확립하기 위해 HTTP를 대체할 MQTT 프로토콜이
제시되었고 OASIS(Organization for the Advancement of Structured Information Standards)에서는 MQTT를 사물 인터넷의 표준 규약으로 사용하고 있다.

감각 부여(센서부착). 사물에 청각, 미각, 후각, 촉각, 시각 등을 부여해 주변 환경의 변화를 측정할 수 있도록 한다. 사물에 부여되는 감각은 오감에 한정되지 않고 RFID, 자이로스코프, 가이거 계수기 등을 통한 감각으로 확장될 수 있다. 예컨대 이불의 경우 감압센서와 습도센서를 통해 사용자가 수면 중 몇 번 뒤척였는지, 얼만큼 땀을 흘렸는지 등을 측정할 수 있다.

2.3.3. 기술 동향 및 발전 방향

한국IDC는 전 세계 IoT 지출가이드 보고서에서 사물인터넷 시장규모가 2019년 7,450억 달러에 이를 것으로 전망했다. 이는 2018년 지출액 6,460억 달러보다 15.4% 증가한 수치이다. 또한 전 세계 IoT 시장이 2022년에는 1조 달러를 넘어설 것으로 내다봤다.

IDC 보고서에 따르면 2019년 IoT 솔루션 지출이 가장 클 것으로 예상되는 산업은 조립 제조(1,190억 달러), 공정 제조(780억 달러), 운송(710억 달러), 유틸리티(610억 달러) 순으로 나타났다. 제조업은 제조 운영과 생산자재관리 지원 솔루션 투자에 집중할 전망이다. 운송업은 IoT 지출의 절반 이상을 화물 모니터링에, 유틸리티업은 주로 스마트 그리드에 투자가 예상된다. 5년간 연평균성장률(CAGR)이 가장 빠른 산업은 보험(17.1%), 연방/중앙정부(16.1%), 헬스케어(15.4%) 순으로 나타났다.

IDC의 고객 인사이트 및 분석을 담당하는 마커스 토치아 리서치 디렉터는 “소비자의 IoT 지출은 2019년에 1,080억 달러에 달해 두 번째로 큰 지출을 차지할 전망이다. 주요 소비자 사용 사례는 스마트 홈, 개인 건강 및 커넥티드카 인포테인먼트와 연관될 것"이라며, “예측 기간 동안 스마트 홈에서는 가정 자동화와 스마트 가전이 5년간 연평균성장률 17.8%로 가장 빠른 소비 성장을 보인다.”라고 덧붙였다.

2019년 가장 큰 투자가 예상되는 IoT 활용사례는 제조 운영(1,000억 달러), 생산자재관리(442억 달러), 스마트 홈(441억 달러), 화물 모니터링(417억 달러)부문에서 주도할 것으로 전망된다. 예측 기간 동안 가장 빠른 소비 성장이 예상되는 IoT 활용 사례는 다른 산업에

서 어떻게 IoT 투자를 하는지 보여 준다. 최고 활용 사례에는 공항 시설 자동화(운송), 전기차 충전(유틸리티), 농업 현장 모니터링(자원), 침상 원격측정(헬스케어) 등이 포함된다.

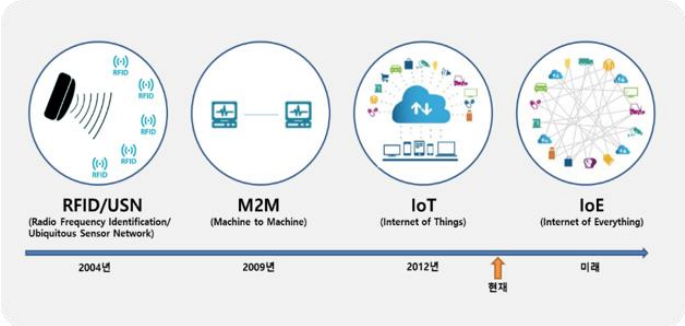


그림 2.10. IoT의 발전 과정

지금까지 인터넷을 이용해 플랫폼에 접속하고 이를 통해 기기간에 소통이 되었다면, 미래에는 기기간의 소통이 주축이 되는 IOE 시대가 본격적으로 열릴 것으로 예상된다.

이를 위해 IOT 시장의 선두주자가 되기 위해 여러 기업에서 기술 개발에 박차를 가하고 있으며, 이종 기기간의 소통이 원활해지는 순간 IOT 산업은 큰 변화를 맞이할 것으로 보인다.

3. 프로젝트 개발

3.1. 프로젝트 개발환경

3.1.1. 하드웨어

CPU	Intel i7 core
RAM	16GB

3.1.2. 소프트웨어

OS	Windows 10, OSX
IDE	Jupyter notebook
Library	Tensorflow 1.12.0 (텐서플로우 내 keras 탑재) Librosa 0.6.2 Numpy 1.15.4 Scipy 1.2.0 Mathplotlib 3.0.2
Language	Python 3.6.5

3.2. 소프트웨어 설계

3.2.1. 시스템 구조도

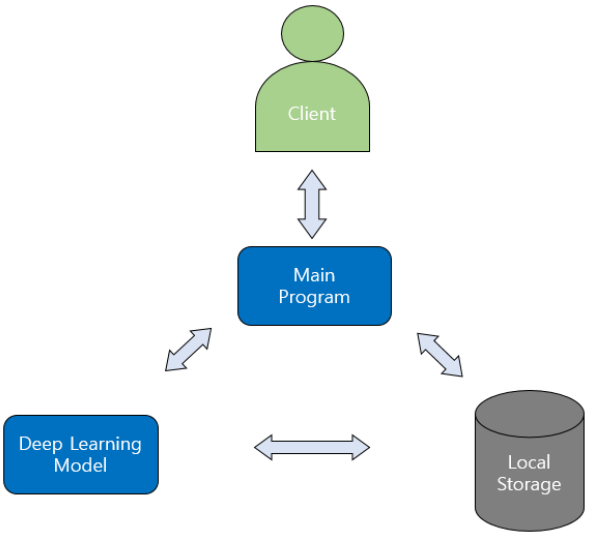


그림 3.1. 소프트웨어 시스템 구조도

3.2.2. Activity DiaGram

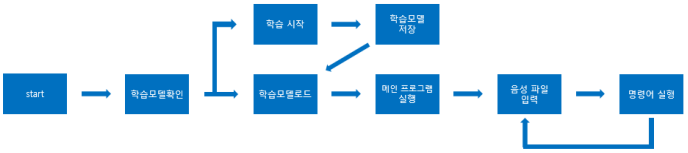


그림 3.2. 소프트웨어 Activity Diagram

3.2.3. 학습 모델 구조도

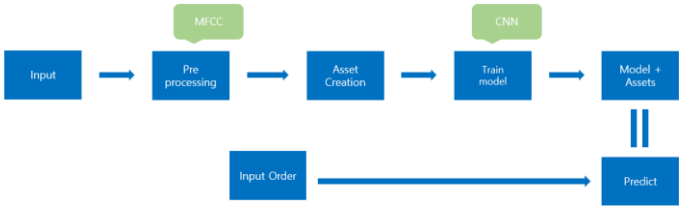


그림 3.3. 소프트웨어 학습 모델 구조도

3.3 소프트웨어 기능

구분	기능	설명
GUI	House	음성인식을 시작
	Stop	모든 명령을 종료
	랜턴 on/off	랜턴이 점등, 소등된 상태를 알 수 있도록 이미지를 통해 구현

	커튼 Up/Down	커튼을 치는지, 걷는지 알 수 있도록 이미지를 통해 구현

3.4. 구현 상세

3.4.1. 음성 전처리

1) 구글 데이터 셋 다운

- 구글에서 딥러닝을 위한 무료 음성 명령 공개 데이터 셋을 다운 받았다.
- 28개의 명령어 중, on/off, open/close, house/stop을 선정하여 약 12196개의 데이터를 확보하였다.

2) 데이터셋 구분

- 총 12196개의 데이터 중에서 10%를 test set, 18%를 validation set, 72%를 learning set으로 구분한다.

3) Mel spectrum

- Python의 librosa 라이브러리를 사용하여 MFCC를 구현한다.
- 받은 파일 소리를 진폭으로 Mel spectrum을 만든다. 주파수로 저장되어 있는 데이터를 데시벨로 변환하여 이를 Mel spectrum으로 만든다.

3.4.2. 딥러닝 학습 모델

1) Audio 데이터 load

```
# MFCC & audio load
import librosa
import subprocess

# DataSize 880
size = 880


def start():

    # DATA PATH SET
    train_path = '.\\trans_data\\train\\'
    print(os.listdir(train_path))
    dirs = [f for f in os.listdir(train_path) if isdir(join(train_path, f))]
    dirs.sort()
    print('Number of labels: ' + str(len(dirs)))
    print(dirs)

    all_wav = []
    unknown_wav = []
    label_all = []
    label_value = {}
    target_list = ['up', 'down',  'on', 'off', 'house', 'stop']
    unknown_list = [d for d in dirs if d not in target_list ]
    print('target_list : ', end='')
    print(target_list)
    print('unknowns_list : ', end='')
    print(unknown_list)
```

- audio의 path를 사용하여 wav파일을 불러온다.
- 여기서 size는 실제 데이터를 입력해 봄으로서 훈련데이터가 대개 어느 정도의 크기를 가지고 있는지 사전 파악 이후에 적당한 크기를 사용자가 정하는 것이다. 사이즈는 바꿔도 상관 없으나, 테스트 과정에도 같은 사이즈의 데이터로 테스트 해야 한다.

2) 데이터 MFCC 적용

```
i = 0
a = 0

hop_length = 512 # 23ms
for direct in dirs[0:]:
    waves = [f for f in os.listdir(join(train_path, direct)) if f.endswith('.wav')]
    label_value[direct] = i
    i = i + 1
    print(str(i) + ":" + str(direct) + " ", end="")
    for wav in waves:
        samples_, sample_rate = librosa.load(join(join(train_path, direct), wav))
        samples = librosa.feature.mfcc(y=samples_, sr=sample_rate, hop_length=hop_length)
        samples = np.array(samples)
        samples = samples.flatten()

        if len(samples) != size:
            a += 1
            continue
        if direct in unknown_list:
            unknown_wav.append(samples)
        else:
            label_all.append(direct)
            all_wav.append([samples, direct])
```

- 파일을 받아서 소리의 진폭으로 Mel-spectrum을 만든다. 주파수로 저장되어 있는 데이터를 데시벨로 변경하여 이를 Mel-spectrum으로 변경한다. 그리고 그 결과를 numpy에 저장한다.
- hop _length는 한 프레임의 단위를 정하는 것으로 512는 약 23ms 정도의 크기를 가진다.
- 먼저, audio 파일을 floating point series로 만든다. Samples_는 audio time series이고, sampling rate는 samples_의 sampling rate이다.Librosa.feature.mfcc의 결과로서, 사이즈가 (20, T)인 numpy.ndarray의 mfcc matrix를 얻게 된다. Ndarray인 samples를 array로 변환한 후, 행 중신 정렬으로 1차원 리스트를 만든다.

3) Data Augmentation

- 훈련데이터와 테스트 데이터를 880으로 통일하여 학습 및 분석 시에 통일성을 부여하였다.

4) 학습 모델 구조

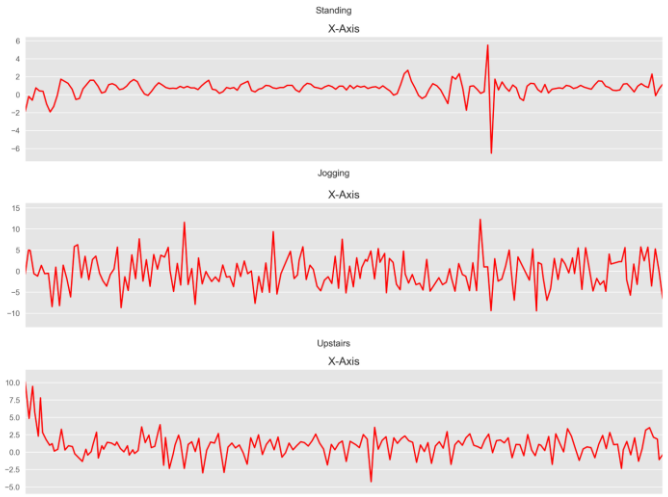
```
lr = 0.001
batch_size = 256
drop_out_rate = 0.5
input_shape = (size, 1)
```

- CNN의 모델 구조는 다음과 같다.
  Input - > (8,11) - > (16,7) -> (32,5) -> (64,5) -> (128,3) -> (256,1) -> (128,1) -> output
- 히든레이어 개수: 7개
- Learning rate : 0.001
- Drop out rate : 0.5
- Batch size : 256
- Train data rate : 0.8
- Test data rate : 0.2
- Activation function : relu
- Output activation function : softmax
- Epoch : 500

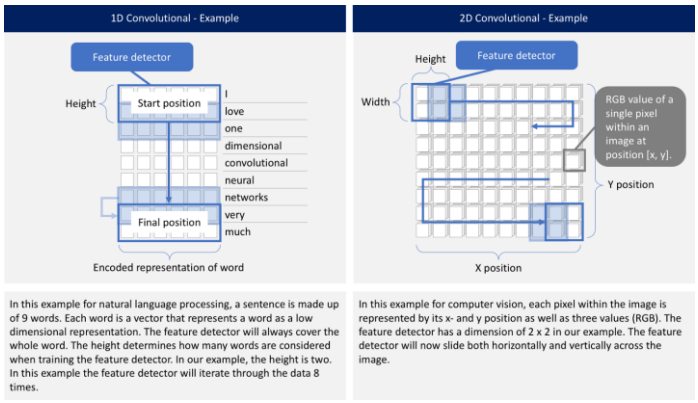
5) Neural Network Model 구축

```
x = layers.Conv1D(8, 11, padding='valid', activation='relu', strides=1)(input_tensor)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(16, 7, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(32, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(64, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(128, 3, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
output_tensor = layers.Dense(len(label_value), activation='softmax')(x)
```

- CNN모델을 Keras를 통해 생성하고, 명령어의 결과를 softmax으로 인코딩했다.
- Conv1D은 CNN알고리즘에서 채널의 사이즈와 activation함수를 정하는 것이다. 여기서 filter와 kernel_size를 정하고, sigmoid보다 성능이 향상된 함수를 이용하기 위해서 activation='relu'로 설정한다. 그리고 stride=1로 설정했다.
- 음성 데이터의 경우 시간을 하나의 축으로 음성 값을 또 다른 하나의 축으로 봐서 2차원으로 분석하기 위해 Conv2D를 사용하는 경우도 있으나 시간에 따른 음성 값의 변화로 1차원의 sequential 데이터로 볼 수 있기에 1D를 사용하는 경우가 많다. 음성 데이터는 아래의 그림처럼 시간의 흐름에 따른 데이터 값(주파수)를 가지고 잇다.



이렇게 시간의 흐름 데이터를 학습하는 과정에서 convolution 1D를 사용 할 경우 convolution 2D에 비해 파라미터의 수가 적기 때문에 훨씬 더 적은 연산을 통해 빠른 학습을 시킬 수 있다. 이 때문에 시간이 부족하고, 고 사양의 CPU 및 GPU가 없는 이번 프로젝트의 여건 상 Conv1D가 적합하다고 생각해 Conv1D를 통해 이번 학습모델을 구성하였다. Conv1D와 Conv2D의 차이점은 아래의 그림과 같다.



위와 같이 Conv1D는 한 방향으로 데이터를 분석하고 Conv2D는 두 개의 방향으로 데이터를 분석하기에

2차원 이상의 데이터가 아닌 음성 데이터는 Conv1D를 사용 했을 시에 학습 효율을 높일 수 있다.
- MaxPooling은 특징들을 subsampling하고 채널을 통과한 Layer를 한번 더 가공하는 것이고 Dropout은 일반화의 오류를 없애기 위한 것으로, 둘 모두 학습의 성능 향상을 위한 작업이다.
- 마지막 단계에서 Flatten과 Dense를 이용해 feature의 크기를 줄이고, 이를 통해서 topology들의 invariance를 지니게 만든다.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 880, 1)	0
conv1d (Conv1D)	(None, 870, 8)	96
max_pooling1d (MaxPooling1D)	(None, 435, 8)	0
dropout (Dropout)	(None, 435, 8)	0
conv1d_1 (Conv1D)	(None, 429, 16)	912
max_pooling1d_1 (MaxPooling1	(None, 214, 16)	0
dropout_1 (Dropout)	(None, 214, 16)	0
conv1d_2 (Conv1D)	(None, 210, 32)	2592
max_pooling1d_2 (MaxPooling1	(None, 105, 32)	0
dropout_2 (Dropout)	(None, 105, 32)	0
conv1d_3 (Conv1D)	(None, 101, 64)	10304
max_pooling1d_3 (MaxPooling1	(None, 50, 64)	0
dropout_3 (Dropout)	(None, 50, 64)	0
conv1d_4 (Conv1D)	(None, 48, 128)	24704
max_pooling1d_4 (MaxPooling1	(None, 24, 128)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 256)	786688
dropout_4 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
dropout_5 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 7)	903

Total params: 859,095
Trainable params: 859,095
Non-trainable params: 0

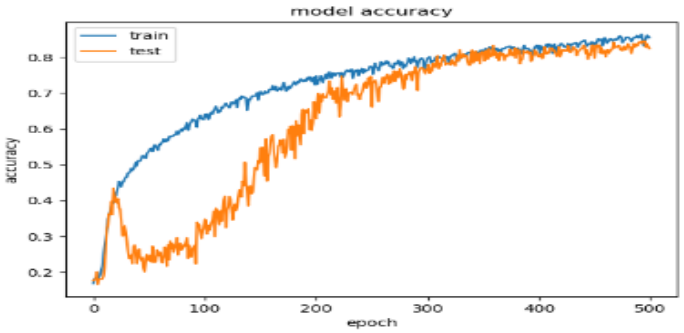
Train on 9756 samples, validate on 2440 samples

- 최종적으로 우리의 모델은 다음과 같이 7개의 계층을 가지게 된다.
-1~5 레이어는 conv1D & maxpooling & Dropout 단계로써 특징을 추출하고 강화하는 단계다.
-6~7 레이어는 데이터를 평활화 하는 단계다.
-최종적인 아웃풋은 128 벡터 크기를 우리가 얻고자 하는 7개 카테고리에 대해서 각각의 확률을 알 수 있도록 soft-max인코딩을 통해 출력한다.

```
history = model.fit(train_wav, train_label, validation_data=[test_wav, test_label],
          batch_size=batch_size,
          epochs=500,
          verbose=1)
```

- train data와 validation data를 설정하여 CNN모델을 학습시킨다.
- 훈련할 때 한꺼번에 가져올 데이터의 양인 batch_size를 정한다. 이 때, batch gradient descent를 사용하려고 했지만 사용하는 데이터가 적지 않았으므로 mini-batch의 방법으로 64, 128, 256, 512로 설정해보았다. 실험해본 결과 accuracy가 256에서 512로 넘어갈 때 하락했기 때문에 batch_size를 256으로 설정했다.

- Epoch는 전체 데이터셋에 대해서 forward와 backward을 한 번 수행 즉, 한번의 학습을 하는 것을 의미한다. Epoch를 주는 것은 최적화하려고 하는 모델이 항상 convex하지는 않고 학습 때마다 초기 위치에 따라 모델의 성능이 달라지기 때문에 Epoch시마다 가중치를 달리하면서 학습하기 위함이다. Sum of loss 그래프를 확인하면서 accuracy를 확인해본 결과, 500보다 더 높게 잡았을 때, overfitting되는 문제가 있었으므로 epoch는 500으로 설정했다.



- Training data 기준 정확도 약 87%, Test data 기준 약 85%

6) 모델 저장

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")
```

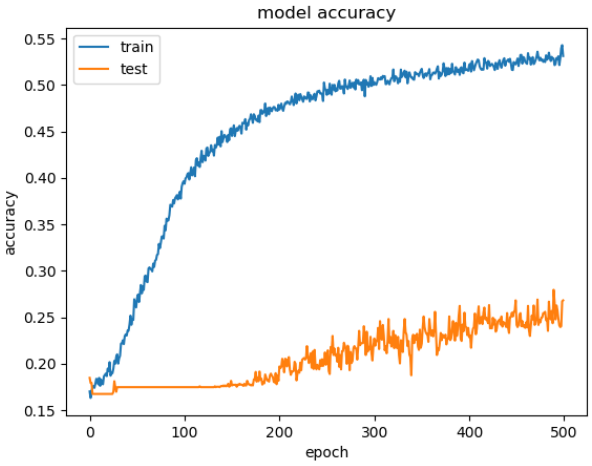
- 모델을 json파일로 변환하고, h5파일로 학습 모델을 저장한다.

3.4.2.1. Neural Network Layer 개선

데이터를 학습함에 있어서 정확도를 높이고자 심층 레이어의 수를 다양하게 해봄으로써 가장 accurate이 높았을 때의 layer 개수를 찾아보았다. Conv Layer를 최대 9단계에서 7단계까지 줄여보면서 가정을 세워보았다.

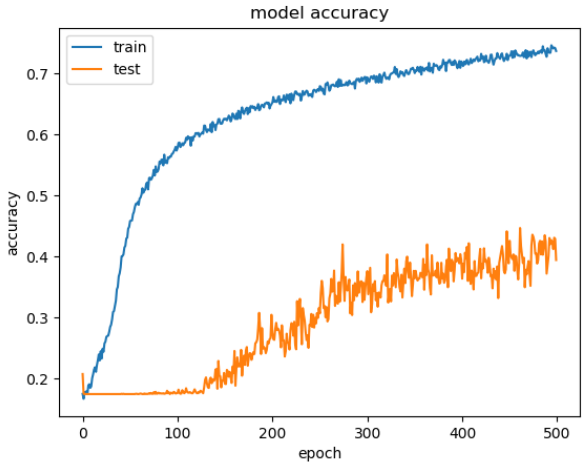
-9개의 Conv Layer를 거쳤을 때

```
x = layers.Conv1D(2, 15, padding='valid', activation='relu', strides=1)(input_tensor)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(4, 13, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(8, 11, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(16, 7, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(32, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(64, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(128, 3, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
output_tensor = layers.Dense(len(label_value), activation='softmax')(x)
```



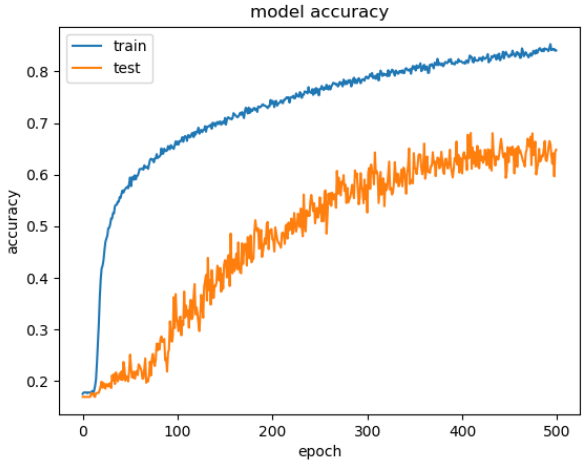
-8개의 Conv Layer를 거쳤을 때

```
x = layers.Conv1D(4, 15, padding='valid', activation='relu', strides=1)(input_tensor)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(8, 11, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(16, 7, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(32, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(64, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(128, 3, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
output_tensor = layers.Dense(len(label_value), activation='softmax')(x)
```

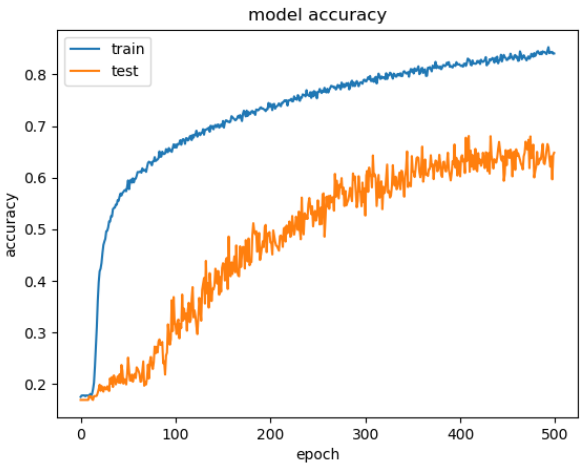


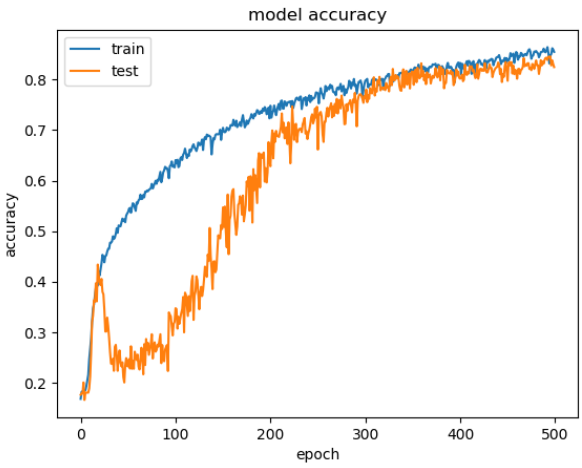
-7개의 Conv Layer를 거쳤을 때

```
x = layers.Conv1D(8, 11, padding='valid', activation='relu', strides=1)(input_tensor)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(16, 7, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(32, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(64, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Conv1D(128, 3, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
output_tensor = layers.Dense(len(label_value), activation='softmax')(x)
```



기본적으로 Layer의 개수를 늘리면 정확도가 높아지는것이 정론이지만 적절하지 못하면 문제가 발생하여 위와같은 결과가 나타나는걸 알게 되었다. 우선 망이 커지면 우리 프로젝트처럼 학습할 데이터의 양이 한정적이기에 중간에 Dropout 층을 삽입하더라도 과적합 문제가 일어나 테스트 데이터를 제대로 인식하지 못하는 상황이 발생하는걸 알 수 있다. 또 다른 문제로 Layer가 많아 지면 그만큼 연산량이 많아져서 연산능력이 뛰어난 GPU를 사용하더라도 연산량의 증가는 심각한 문제가 된다. 미미한 차이였지만 실제로 9Layer에서 7Layer로 줄이면서 학습했을 때의 학습 속도의 차이가 존재하였다. 게다가 학습이 잘못되어 fiter의 kernel이 특정한 무리에 쏠리게 된다면 기껏 망의 크기를 늘렸음에도 불구하고 최적의 결과를 얻지 못할 수도 있다.





컴퓨터 자체의 성능도 데이터들을 학습함에 있어서 많은 결과 차이를 낳았다. 팀원이 4명으로 표본이 적었지만 동일한 코드로 학습을 하였는데도 불구하고 이처럼 정확도(학습시간의 차이도 존재) 차이가 있는 것으로 따졌을 때 CPU성능 차이에 있어서도 학습에 중대한 영향을 끼치는 것을 알았다.

3.4.2.2. 딥러닝 학습 모델 해석

- 모델은 7레이어로 이루어져 있다.

```
# Conv1D Model
input_tensor = Input(shape=(input_shape))
```

1) 데이터를 인풋

input_1 (InputLayer)	(None, 880, 1)	0

기존에 설정했던 880개의 데이터가 정상적으로 입력되었음을 알 수 있다.

2)첫번째 레이어

```
x = layers.Conv1D(8, 11, padding='valid', activation='relu', strides=1)(input_tensor)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
```

conv1d (Conv1D)	(None, 870, 8)	96
max_pooling1d (MaxPooling1D)	(None, 435, 8)	0
dropout (Dropout)	(None, 435, 8)	0

첫번째 레이어에 필터8과 커널크기 11을 적용하였고 이에 따라 출력의 크기가 870인 window가 8개 생성 된 것을 확인 할 수 있다. 이후 정확도 향상을 위해 이후 Maxpooling과 Dropout을 이용하였고 크기를 반으로 줄었다.

3)두번째 레이어

```
x = layers.Conv1D(16, 7, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
```

conv1d_1 (Conv1D)	(None, 429, 16)	912
max_pooling1d_1 (MaxPooling1	(None, 214, 16)	0
dropout_1 (Dropout)	(None, 214, 16)	0

두번째 레이어에 필터16과 커널크기 7을 적용하였고 이에 따라 출력의 크기가 429인 window가 16개 생성 된 것을 확인 할 수 있다. 이후 정확도 향상을 위해 Maxpooling과 Dropout을 이용하였고 크기를 반으로 줄었다.

4)세번째 레이어

```
x = layers.Conv1D(32, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
```

conv1d_2 (Conv1D)	(None, 210, 32)	2592
max_pooling1d_2 (MaxPooling1	(None, 105, 32)	0
dropout_2 (Dropout)	(None, 105, 32)	0

세번째 레이어에 필터210과 커널크기 32을 적용하였고 이에 따라 출력의 크기가 210인 window가 32개 생성 된 것을 확인 할 수 있다. 이후 정확도 향상을 위해 Maxpooling과 Dropout을 이용하였고 크기를 반으로 줄였다.

5)네번째 레이어

```
x = layers.Conv1D(64, 5, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Dropout(drop_out_rate)(x)
```

conv1d_3 (Conv1D)	(None, 101, 64)	10304
max_pooling1d_3 (MaxPooling1	(None, 50, 64)	0
dropout_3 (Dropout)	(None, 50, 64)	0

네번째 레이어에 필터64과 커널크기 5를 적용하였고 이에 따라 출력의 크기가 101인 window가 64개 생성 된 것을 확인 할 수 있다. 이후 정확도 향상을 위해 Maxpooling과 Dropout을 이용하였고 크기를 반으로 줄였다.

6)다섯번째 레이어

```
x = layers.Conv1D(128, 3, padding='valid', activation='relu', strides=1)(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Flatten()(x)
```

conv1d_4 (Conv1D)	(None, 48, 128)	24704
max_pooling1d_4 (MaxPooling1	(None, 24, 128)	0
flatten (Flatten)	(None, 3072)	0

다섯번째 레이어에 필터128과 커널크기 3를 적용하였고 이에 따라 출력의 크기가 48인 window가 128개 생성 된 것을 확인 할 수 있다. 이후 정확도 향상을 위해 Maxpooling과 Dropout을 이용하였고 크기를 반으로 줄였다.

7)여섯번째 레이어

```
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
```

dense (Dense)	(None, 256)	786688
dropout_4 (Dropout)	(None, 256)	0

여섯번째 레이어부터는 평탄화 이후 추출된 특징들을 통합하는 과정이다. 크기가 256으로 줄어든 것을 확인 할 수 있다.

8)일곱번째 레이어

```
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(drop_out_rate)(x)
```

dense_1 (Dense)	(None, 128)	32896
dropout_5 (Dropout)	(None, 128)	0

일곱번째 레이어도 추출된 특징들을 통합하는 과정이다. 크기가 128으로 줄어든 것을 확인 할 수 있다.

9)아웃풋

```
output_tensor = layers.Dense(len(label_value), activation='softmax')(x)
```

dense_2 (Dense)	(None, 7)	903

최종적으로 7개으 특성이 성공적으로 추출 된 것을 것 수 있다.

-결과

```
Total params: 859,095
Trainable params: 859,095
Non-trainable params: 0
```

```
Train on 9756 samples, validate on 2440 samples
```

학습 결과이다.

3.4.3. 딥러닝 추론 모델

```
def pred(model,file_path):
    samples, sample_rate = librosa.load(file_path)
    samples = librosa.feature.mfcc(y=samples, sr=sample_rate)
    samples = np.array(samples)
    samples = samples.flatten()[:size]

    # (-1, size, 1)
    samples = samples.reshape(-1, 880, 1)
    pred = model.predict(samples)
    pred_cls = np.argmax(pred, axis=1)[0] + 1
    if pred_cls==1:
        print("up")
    if pred_cls == 2:
        print("down")
    if pred_cls==3:
        print("on")
    if pred_cls==4:
        print("off")
    if pred_cls==5:
        print("house")
    if pred_cls==6:
        print("stop")
    if pred_cls==7:
        print("명령어가 아닙니다.")
        showwarning("warning","명령어가 아닙니다")

    return pred_cls
```

- 알고 있는 sampling rate와 audio time series를 이용해서 'samples'라는 series를 만들고, flatten함수를 통해 1차원으로 변환시킨 후, size 880으로 변환한다. 저장된 모델을 이용하여 추론한 결과를 pred_cls에 넣고, 각각의 경우의 수를 분할한다.
  - 어떤 결과가 나왔는지 확인하기 위해서 각각의 결과를 print하도록 설정해두었다.
- 그리고, house, on, off, up, down, stop 이외의 명령어가 들어왔을 경우에는 Python 콘솔창에 "명령어가 아닙니다"라고 띄운다.

3.4.4. GUI

1) 음성인식 인터페이스

```
def up():
    global img1, chk_on, updown, onoff
    if chk_on==-1:
        return

    updown=True
    if chk_on==1 and onoff==False:
        tkim=Image.open("slide3.jpg")
        tkim=tkim.resize((700,500), Image.ANTIALIAS)
        tkim=ImageTk.PhotoImage(tkim)
        img1.config(image=tkim)
        img1.image=tkim

    elif chk_on==1 and onoff==True:
        tkim=Image.open("slide4.jpg")
        tkim=tkim.resize((700,500), Image.ANTIALIAS)
        tkim=ImageTk.PhotoImage(tkim)
        img1.config(image=tkim)
        img1.image=tkim

    else: return
```

- Image 라이브러리를 사용하여 up, down, on, off, house, stop의 각각 경우에 따른 이미지를 띄우도록 함수를 만든다.

- background 이미지를 먼저 삽입해 놓은 후에, 함수에 따라서 각각의 이미지를 띄우도록 설계했다.

2) 시뮬레이션

```
if pred_cls==1:
    up()

if pred_cls==2:
    down()
if pred_cls==3:
    on()
if pred_cls==4:
    off()

if pred_cls==5:
    poserOn()
if pred_cls==6:
    powerOff()
```

- 모델이 추론한 각각의 경우에 따라 GUI를 띄우는 함수를 매핑시킨다.

- on(), off(), down(), up(), powerOn(), powerOff()에는 각각의 경우에 띄워야 할 이미지가 저장되어 있다. 이 함수는 1) 음성인식 인터페이스 의 코드 양식과 같다.

4. 프로젝트

4.1. 프로젝트 사용방법

4.1.1. 시작화면

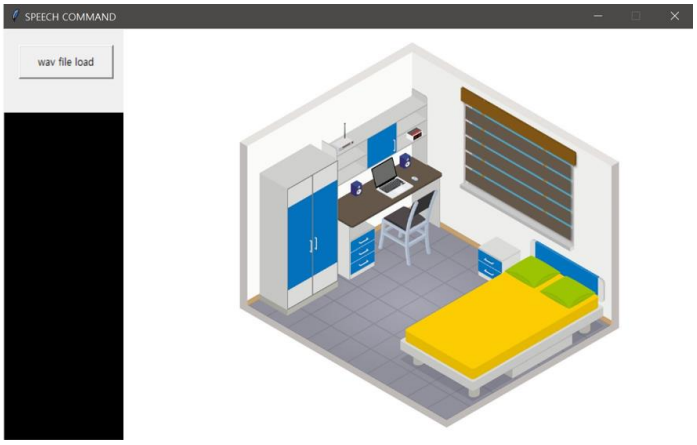
- house 명령어를 통해 power on 하기 전에는 아무 명령어도 작동하지 않은 상태이다.

- wav file load : wav file을 load하는 버튼 (1)

4.1.2. house 명령어를 통해 Power On 이후

- powerOn이 되면 맨 처음 상태는 조명 Off되고, 커튼이 Down된 상태이다.

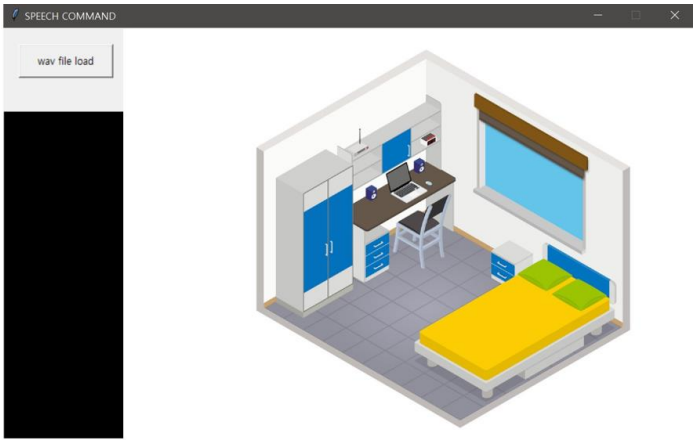
4.1.2. On/Off 명령어를 통해 조명을 On/Off



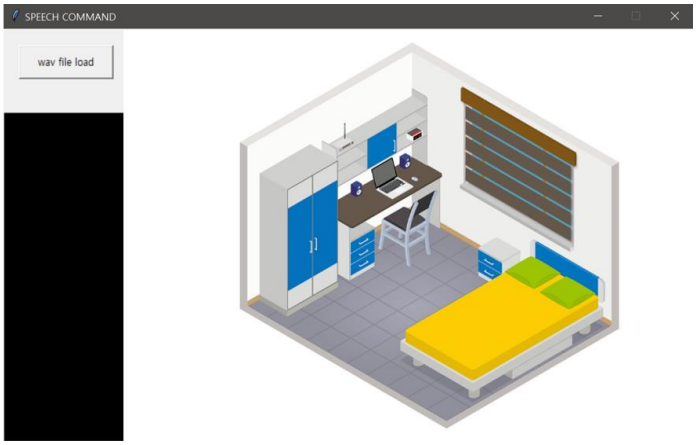
- 조명이 On된 상태는 방이 환해진 상태이다.

- 조명이 Off된 상태는 방이 어두워진 상태이다.

4.1.3. Up/Down으로 커튼을 Up/Down



- 커튼이 Up된 상태는 커튼이 올라간 상태이다.



- 커튼이 Down된 상태는 커튼이 내려간 상태이다.

4.1.4. 즉, 모든 경우의 수

- Power On
- 조명 On / 커튼 Up
- 조명 On / 커튼 Down
- 조명 Off / 커튼 Up
- 조명 Off / 커튼 Down
- Power Off

5. 결론

5.1. 다음 프로젝트 계획

애초에 이 프로젝트는 마이크를 통한 실시간 음성인식, 이후 실제 하드웨어를 통해 명령을 구현 하는 것을 계획하였으나 여러 제약으로 인해 제작한 음성인식 모델이 각 명령어를 구별 할 수 있고, 명령어를 입력했을 때 GUI를 통해 명령을 실행하는 것을 확인하는 것으로 프로젝트 범위를 축소 한 것이다. 때문에 다음 프로젝트에서는 마이크를 통해 실시간으로 음성 명령을 입력하고 그 명령을 실제 하드웨어를 통해 구현하는 것을 목표로 한다. 음성 입력은 마이크로 진행하며 명령은 아두이노 혹은 라즈베리를 이용해 구현하는 것으로 계획한다.

5.2. 장기적인 프로젝트 진행 방향

현재 이 프로젝트는 6개의 학습 단어를 패턴인식 시키고 이 명령어를 구별하는 것을 목적으로 하였고, 성공적으로 구별되는 것을 확인하였다. 때문에 이 이후의 프로젝트 진행 방향은 크게 2가지 방향으로 나뉘어 진다.

1) 학습 단어의 수 증가.
   현재는 6개의 단어만 인식되기에 집안의 모든 기기들을 음성을 통해 통제 할 수 없는 상태이다. 때문에 각각의 기기들에 필요한 명령어에 맞게 명령어를 추가로 선별하고 이 명령어들을 학습시킴으로써 최종적으로 집 안의 모든 기기들을 음성인식 스피커를 통해 사용자가 원하는 명령을 실행시키도록 계획한다.
2) 고립단어 인식에서 연속단어 인식으로 발전
   이번 프로젝트는 음성인식 프로세스에서 앞의 2단계만 구현 해 놓은 상태이다. 음성 데이터를 전처리 하여 학습 시킬 수 있는 데이터로 변환시켰으며 패턴인식을 통해 사용자가 말하는 것이 무엇을 의미하는 지 학습모듈을 통해 구별 시켰다. 이후의 프로젝트에서는 패턴인식을 넘어

언어모델을 통해 언어처리까지 하는 모델을 구현하는 것을 목적으로 한다. 현재의 모델로는 각각의 명령어가 인식이 되어 각각의 명령어들이 기기에 종속되지만 언어모델을 통해 언어처리를 할 수 있게 된다면 사용자가 원하는 행위를 학습모델에서 판단하고 이에 맞는 명령어를 각각의 기기에 전달함으로써 사용자의 편익이 늘어 날 것으로 예상한다.

5.3 이번 프로젝트의 한계점

이 번 프로젝트를 더 성공적으로 마무리 하고 싶었지만 조금은 아쉬운 상태로 마무리 된 몇 가지 요소들이 있었다.

첫 번째 요소로는 우리가 수집 할 수 있는 음성 데이터의 숫자가 너무 부족하다는 점이다. 그나마 구글에서 무료로 음성데이터 셋을 배포 하는 것이 있어서 그 데이터를 통해 학습 시킬 수 있었다.

두 번째 요소로는 개인 노트북으로 대용량 음성 데이터를 학습시키기에는 컴퓨팅 자원이 너무 많이 필요했다는 점이다. 팀원 중 한 명의 노트북은 학습 도중 CPU가 녹아버리는 현상이 발생 할 정도였다.

세 번째 한계점은 제한된 시간으로 인해 여러 알고리즘들을 비교 후 시작하지 못했다는 점이다. 충분히 많은 자료를 통해 최적의 알고리즘을 시행 하였지만 실제로 돌려 본 결과는 달랐을 지도 몰라서 약간의 아쉬움이 있다.

네 번째 한계점으로는 졸업 프로젝트 가이드 라인의 부재이다. 전 팀원들이 가이드 라인 없이 자율적으로 A-Z까지 프로젝트를 처음으로 진행 하였는데, 그 과정에서 미숙한 점이 많이 발견되었다. 그 과정에서 배운 점은 많았지만 가이드라인이 있었더라면 훨씬 더 좋은 결과 도출이 가능했을 것이다.

마지막 한계점이자 가장 큰 한계점이었던 것은 바로 부족한 시간이다. 약 3개월간 프로젝트가 진행되었는데 중간에 프로젝트 주제가 바뀜으로 인해 실제 프로젝트 진행 기간은 2개월밖에 되지 않았다. 2개월간 새로운 언어와 기술을 공부 해 만족스러운 Out Put을 산출하는 과정이 쉽지 않았다. 그래도 성공적으로 음성을 인식하는 프로그램을 개발 해 만족스럽다.

5.3 이번 프로젝트의 시사점

이 번 프로젝트는 우리에게 몇 가지 시사점을 선사하였다. 이론적으로 검증되지는 않았지만 이번 프로젝트에서 발견 한 내용도 있고 추가적으로 알아보아야 할 내용들도 있다. 마지막으로 이번 프로젝트가 선사하는 의미까지 알아보겠다.

첫 번째 시사점으로는, 음성인식의 경우 소음을 추가해 줌으로써 학습의 정확도를 올릴 수 있는 경우가 많았는데 이번 프로젝트에서는 소음을 삽입해도 유의미한 인식률 향상이 일어나지 않았다는 점이다. 이번 프로젝트가 짧은 음성, 한 단어만 학습시켰기에 소음 삽입 시에도 유의미한 인식률 향상이 없었던 것으로 예상한다. 실제로 짧은 음성, 한 단어에서는 소음이 삽입되어도 유의미한 인식률 향상이 없는지 추가 조사가 필요하다.

두 번째 내용으로 음성의 경우 시간에 흐름에 따른 데이터 값으로 데이터의 순서가 시간의 흐름과 일치한다는 특성을 가지고 있다. 때문에 1Direction Data이고 이로 인해 convolution 2D가 아닌 convolution 1D를 사용해도 충분한 학습 성능을 보여준다.

세 번째 요소로는 Hardware의 성능에 따라 학습 성능이 다르게 나왔다는 점이다. 같은 알고리즘과 코드를 사용했지만 하드웨어가 달랐을 때 단순 시간의 차이뿐만이 아니라 학습 정확도 또한 다르게 나왔다. 이 또한 추가 조사가 필요하다.

네 번째 시사점으로는 음성인식 학습을 시키는 과정에서는 전처리가 굉장히 중요하다는 점이다. 다른 문서 데이터나 사진데이터와는 달리 음성은 기본적으로 analog를 digital로 변환하는 과정 또한 인공지능 학습의 전처리에서 고려해야 하는 사항이다. 때문에 음성파일의 포맷이나 음성의 저장 형식이 그리고 가공 방법에 따라 학습에 영향을 상당히 많이 받는다는 것을 확인 하였다.

마지막 시사점은 이번 프로젝트가 음성인식과 관련된 모든 응용기술에서 사용 할 수 있다는 것을 알게 되었다는 점이다. 이번에는 6개의 단어로 패턴인식을 시켰지만 인식시키고 싶은 단어의 파일들만 수집 할 수

있다면 패턴인식을 통해 인식이 가능하다. 이는 영어 뿐만이 아니라 모든 언어에서 사용 가능하며 데이터만 충분하다면 어떤 언어라도 패턴 학습을 통해 인식 가능하다는 점에서 이번 프로젝트가 상당히 유용한 의미를 가지고 있다는 점을 알 수 있다.

5. 요약 및 음성인식 AI

이 논문에서는 음성인식 딥러닝의 여러가지 알고리즘 중에서 MFCC와 CNN모델을 사용하여 음성 인식 모델을 만들고, 테스트 해보았다. 딥러닝 모델을 만들면서 알게 된 것은 컴퓨터가 배우는 방식을 매우 논리적이게 설계해야 한다는 것이다. 그리고 완성된 모델에서 생각보다 accuracy값에 비해 실제 테스트 정확도는 현저히 떨어진다는 점을 알았고, 좋은 모델을 만들기 위해서 다양한 값의 epoch, batch와 다양한 알고리즘, 모델들을 테스트해야 한다는 것을 알았다. 딥러닝에 대해서 무엇인지 잘 모르던 4명의 학생이 모여서 같이 공부하고, 조사하고 코드를 분석해가면서 새로운 분야를 배워봤다는 것에 논문의 의의가 있다.

최근 스마트폰 시장 포화 상황에서 새로이 주목받고 있는 새로운 시장 중 하나는 사물인터넷(IoT)이다. 무선 인터넷을 통해 집안의 각종 시설과 가전, 전자 제품이 서로 통신을 주고 받을 수 있게 되면서, 이를 컨트롤하는 허브 역할을 두고 TV, 냉장고, 스마트폰 등이 경쟁해왔다. 그러나 2019년 현재 스마트홈을 통제할 핵심 기기로 가장 주목받는 것은 음성인식 및 AI 기능이 결합된 제품이다. Google, Apple, Microsoft 등 글로벌 IT 기업들이 제각기 자사 소프트웨어를 탑재한 스피커를 출시하며 기존 Echo가 독식하던 시장에 진입했으며, 국내 시장도 경쟁이 가속화되고 있다.

인공 지능은 4차 산업혁명의 핵심 기술이라고 여겨지고 있으며, 국가 및 기업들 간 활발한 지원이 이루어지고 있다. 현재 독일과 미국이 인공 지능 및 4차 산업혁명 기술 시장을 장악하고 있으나 중국, 일본 등이 이를 빠르게 추격하고 있으며 한 국 및 기타 국가들 역시 4차 산업혁명의 흐름을 따라잡기 위해 정부 측면에서 노력하고 있다. 최근까지 성장세를 보여왔던 스마트폰 시장의 경우 스마트폰 기술 혁신이 더디어짐에 따라 성장 속도가 줄어들고 있으며 대부분의 수요를 교체 수요에 의존해야 하는 상황이라 예측되고 있다. 이러한 상황에서 기업은 새로운 활로로서 사물인터넷(IoT)에 주목하고 있으며 많은 기업들이 AI 음성 인식 비서를 탑재한 스피커를 출시함으로써 경쟁을 벌이고 있다. 오늘날 스마트폰 뿐 아니라, 가정용 음성 중심 기기까지 포함하는 경우, 이러한 형태의 음성 인식 기기는 약 3천 3백만대에 이를 것으로 예상되고 있다.

한국의 경우 각 기업들과의 협력을 통해 음성 인식 금융 서비스, 종합 홈 오토메이션 AI 시서비스 제공 등 새로이 열린 AI 시장 선점을 목표로 활발하게 움직이고 있다. 또한 애플, 마이크로소프트, 삼성 등 고객 충성도를 확보하고 있으며 자사 하드웨어 제품을 내놓고 있는 기업의 경우 AI 비서 서비스를 자사 제품에 적용함으로써 고객 충성도를 높이고, 시장 점유율 역시 극대화하고자 하는 움직임을 보이고 있는데 이는 시장 점유뿐 아니라 향후 음성 인터페이스의 표준을 선점하려는 노력으로 볼 수 있을 것이다. 특정 음성 인식 AI 소프트웨어나 스피커를 이용하여 더 많은 기기를 제어할수록 고객의 편의성을 극대화하고 시장을 점유할 기회가 높아질 것이기 때문이다.

반면 음성 인식 서비스 시장이 확대됨에 따라 부작용 이야기도 심심치 않게 흘러 나오고 있는 것이 사실이다. 먼저, 이용자의 프라이버시 문제가 대두되고 있다. 매우 쉽게 음성이 수집되어 악용될 수 있기 때문이다. AI 스피커를 통해 무작위로 수집된 데이터가 기업들에게 저장되어 기업들의 파놉티콘 문제도 꾸준히 제기되고 있으며, 데이터 도난 및 해킹에 대한 우려도 존재한다.

음성 인식오류 등 AI 기술 자체의 내재적인 움직임 역시 문제가 되고 있다. 빠른 개선이 이루어지고 있는 것은 사실이나 아직 음성이나 문자 인식 오류가 빈번한 상황이기 때문이다. 이러한 문제를 해결하기 위해서는 보안 및 규제 문제 등이 선결 과제로 지적되고 있으며 기술적인 측면에서는 음성 인식률을 높이고 자연 언어 처리기술 및 빅데이터 및 인공지능 분야에서의 연구가 지속적으로 개선되어 야 할 것이라고 보인다

참 고 문 헌

[1] 김지은, 이인성, MFCC를 이용한 GMM 기반의 음성/혼합 신호 분류 (Speech/Mixed Content Signal Classification Based On GMM Using MFCC), 2013

[2] Thomas F. Quantieri, Discrete-Time Speech Signal Processing, 2011

[3] Y.Lecun, L.Bottou, Gradient-based learning applied to document recognition, 1998

[4] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012

[5] 모두를 위한 머신러닝/딥러닝 강의(웹사이트), http://hunkim.github.io.ml/

[6] 딥러닝 위키백과(웹사이트), http://ko.wikipedia.org

[7] Convolutional Neural Networks for Visual Recognition(웹사이트), http://cs231n.stanford. Edu/syllabus.html