Limited Vocabulary KWS & Classifier for STT

- 2020 졸업작품 최종 보고서 –

2020. 10. 08

성 명	김선미 | 원혜진
소 속	융합전자공학부
학 년	4학년
학 번	2017027629 | 2016024084
지 도 교 수	장준혁 교수님 (인)

융합전자공학부
한양대학교

목차

요 약 본

프로젝트 정보		
기술분야	AI, 음성인식, Deep Learning	
프로젝트명	Limited Vocabulary KWS and Classifier for STT	
역할 분담	김선미	Deep Learning 모델 구축, 학습, 프로그램 및 Demo 개발, 보고서 및 발표 자료 제작
	원혜진	서버 Interface 구현, 프로그램 개발, 영상 편집
프로젝트 소개	여러가지 Deep Learning 구조로 음성 KWS과 30words Classifier 모델을 구축하고 학습할 수 있다. 학습된 모델을 이용하여 10초 동안 녹음한 음원에서 Keyword (e.g. “STOP”) 유무를 판단하고, 존재한다면 Classifier가 작동하여 Speech to Text 결과를 출력한다.	
개발배경 및 필요성	오늘날 AI 스피커(비서)가 등장함에 따라 음성을 Interface로 사용하는 사례가 늘어나고 있다. 이는, AI가 적용되는 다양한 분야 중 자연어 처리라는 한가지 예시일 뿐이다. 그 외에도 BIO 혈압 추정, 음성 합성 등 응용 분야는 무궁무진하다. 이러한 IT 동향에 발맞추어 음성인식의 기본과제인 KWS과 Classifier 모델을 다양한 DL구조로 구현하는 것을 1차 목표로 잡게 되었다. 이를 통해, DL의 입력단부터 출력단까지의 전체적인 흐름을 이해할 수 있고, 더 나아가 학부 수업 [DSP], [인공지능개론] 등에서 배운 내용을 심도 있게 받아들이는 것을 본 작품의 최종 목표로 한다.	
프로젝트 주요기능	Keyword Spotting 30words Classifier	
작품의 특징과 기대효과	학부 수업에서 배운 내용을 직접 구현해본다는 취지에 맞추어 몇 가지 제한 속에서 음성인식기를 만들어보았다. 10초 내의 음원, 30개의 단어 사전이라는 제한을 가지고 나만의 음성인식기를 만들어 봄으로써, 음성 데이터를 Machine Learning에 적용하는 일련의 과정을 공부할 수 있다. KWS 모델의 경우 낮은 정확도를 가지고 있어, 보조 모델로 Classifier가 한 번 더 사용되었다. 또한, 2가지 기능을 구현하기위해 2배 이상의 노력을 들였다. 각 기능에 대해 다음과 같은 Task를 정의할 수 있는데, KWS : Multi-Class Task, 30 Words Classifier : Multi-Label Task 각 Task에 필요한 Model, Hyper-parameter, Loss함수 간에 적절한 조합을 찾기 위해 여러 실험 과정이 담겨 있다. 이러한 실험 과정은, 추후 시중에 나와있는 음성인식기 만큼의 성능을 구현할 수 있는 수준에 도달할 큰 발돋움이 될 것으로 기대한다.	

1. 서론

1.1. 작품 소개

[작품명] Limited Vocabulary KWS and Classifier for STT

여러가지 Deep Learning 구조로 음성 Keyword Spotting과 30 Words Classifier 모델을 구축하고 학습할 수 있다. 학습된 모델을 이용하여 10초 동안 녹음한 음원에서 Keyword (e.g. “STOP”) 유무를 판단하고, 존재한다면 Classifier가 작동하여 Speech to Text 결과를 출력한다.

1.2. 개발 배경 및 목적

오늘날 AI 스피커(비서)가 등장함에 따라 음성을 Interface로 사용하는 사례가 늘어나고 있다. 이는, AI가 적용되는 다양한 분야 중 자연어 처리라는 한가지 예시일 뿐이다. 그 외에도 BIO 혈압 추정, 음성 합성 등 응용 분야는 무궁무진하다.

그림 1 (2018. 09 기준) 국내 AI 스피커 출시 현황 [1]

이러한 IT 동향에 발맞추어 음성인식의 기본과제인 KWS과 Classifier 모델을 다양한 DL구조로 구현하는 것을 1차 목표로 잡게 되었다. 이를 통해, DL의 입력단부터 출력단까지의 전체적인 흐름을 이해할 수 있고, 더 나아가 학부 수업 [DSP], [인공지능개론] 등에서 배운 내용을 심도 있게 받아들이는 것을 본 작품의 최종 목표로 한다.

1.3. 축약어 정리

KWS (Keyword Spotting), Classifier (30words Classifier), STT (Speech to Text), DL (Deep Learning), ML (Machine Learning)

1.4. 작품 구성도

1.4.1. 전체 흐름도

① 10초 동안 음원을 녹음한다. 녹음된 음원을 KWS 모델을 이용하여, 키워드 “STOP” 이 존재할 위치를 찾아낸다. 위 모델은 입력 데이터가 모두 “STOP”이 존재한다는 가정하에 학습되었기 때문에 보조 모델로 Classifier가 사용된다.

② KWS 모델이 예측한 확률 최대값을 기준으로 “STOP”이라고 추정되는 1초짜리 단어를 30words Classifier에 넣어준다.

③ 그 결과가 Threshold (0.6) 이상일 때만, Keyword가 존재한다고 판단하여, STT가 활성화된다.

④ 10초 녹음파일이 Peak를 기준으로 1초짜리 N개의 단어로 추출되어 30 Words Classifier에 들어가고 그 결과를 문장으로 출력한다.

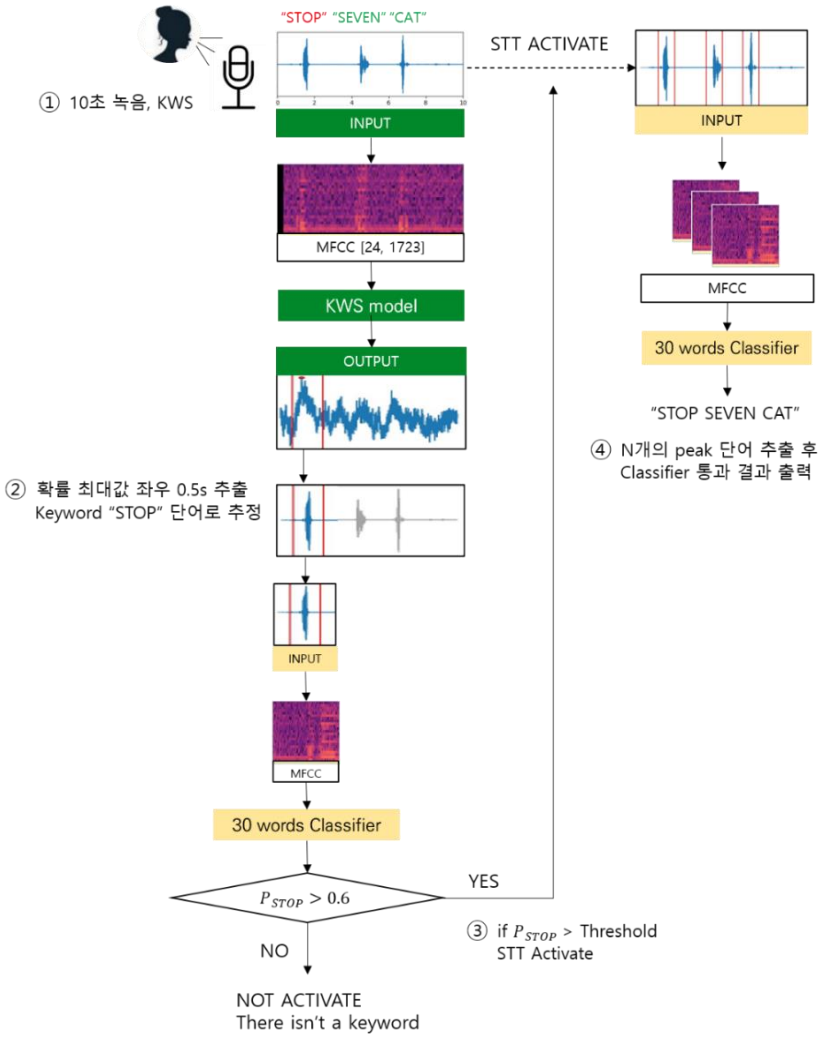
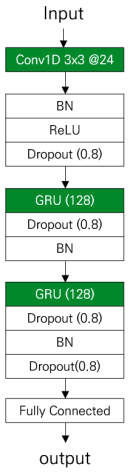
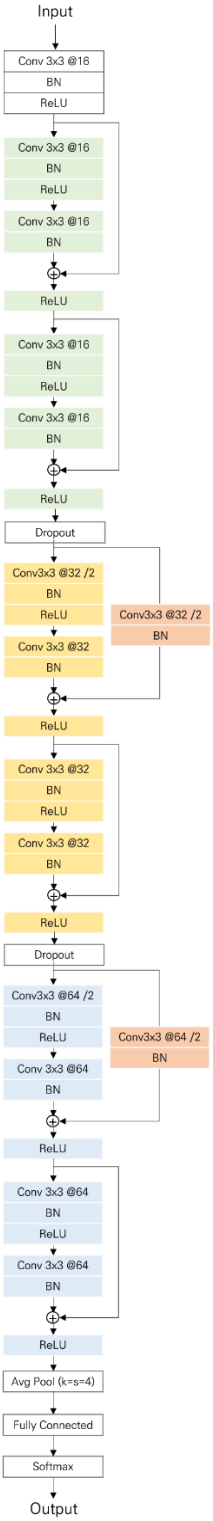


그림 2 작품 전체 흐름도

1.4.2. 구현한 Deep Learning 모델 구조

KWS Task	30words Classifier Task
GRU 2-layer	3-Blocks ResNet
	

1.5. 작품의 특징과 기대효과

현존하는 AI 음성인식기 (ex. 삼성 Bixby, 애플 Siri, 그리고 Google Assistant)의 기본적인 기능 2가지를 구현하였다.

① 음성인식기 활성화 – Keyword Spotting

② Speech to Text

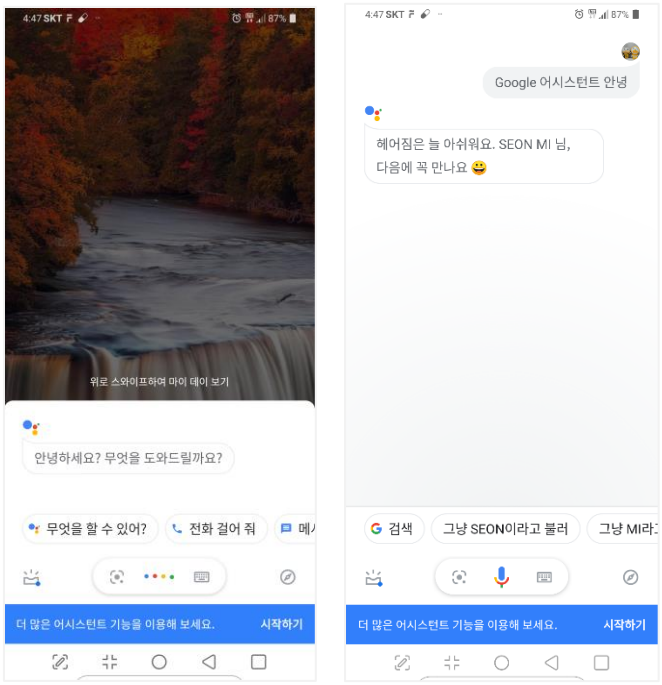


그림 3 기존의 음성 인식기 기능 (좌) KWS, (우) STT

우리의 작품은 학부 수업에서 배운 내용을 직접 구현해본다는 취지에 맞추어 몇 가지 제한 속에서 음성인식기를 만들어보았다. 10초 내의 음원, 30개의 단어 사전이라는 제한을 가지고 나만의 음성인식기를 만들어 봄으로써, 음성 데이터를 Machine Learning에 적용하는 일련의 과정을 공부해볼 수 있었다. KWS 모델의 경우 낮은 정확도를 가지고 있어, 보조 모델로 Classifier가 한 번 더 사용되었다.

마지막으로, 2가지 기능을 구현하기위해 2배 이상의 노력을 들였다는 점을 강조하고 싶다. 각 기능에 대해 다음과 같은 Task를 정의할 수 있는데, KWS : Multi-Class Task, 30 Words Classifier : Multi-Label Task 각 Task에 필요한 Model, Hyper-parameter, Loss함수 간에 적절한 조합을 찾기 위해 여러 실험 과정이 담겨 있다. 이러한 실험 과정은, 추후 심층 학습 및 연구를 통해 시중에 나와있는 음성인식기 만큼의 성능을 직접 구현할 수 있는 수준에 도달할 수 있는 큰 발돋움이 될 것으로 기대한다.

2. 본론

2.1. Machine Learning 기본 원리

Deep Learning의 상위 개념인 Machine Learning은 데이터를 군집화, 분류하는 기술이다. 이를 구현하는 여러 알고리즘 중에 하나가 Deep Learning이다. DL 모델마다 세부적인 구조는 다르지만, 기본적인 학습과정은 다음과 같다. Model에 Input 데이터가 들어가, Forward 계산되어 나온 Output과 Target간의 Loss를 Backward시켜 Model의 Parameter를 학습하는 과정이다.

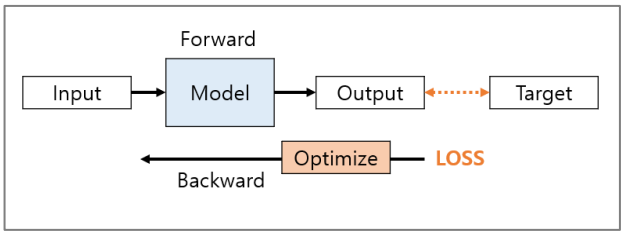


그림 4 Machine Learning 기본 흐름도

기존의 프로그램들은 사람이 직접 Model을 설계하였다면, ML은 Input-Target간의 Pair 데이터에 적합한 Model을 찾는 과정이라 볼 수 있다.

2.2. 학습 시작 전 구조 및 Hyper-parameter 설계

ML은 적정 Model Parameter를 찾는 과정으로서, 그 학습은 Model이 알아서 진행하기 때문에 얼핏 보기에 간단해 보일 수 있다. 하지만, 그 학습을 시작하기 위해 개발자가 지정해줘야 할 것이 많기에 공부할 것은 무궁무진하다. Feature Extraction, Model Architecture(# of neurons), Loss, Optimizer, Regularization, Metric 등 Task마다 적절한 조합이 존재한다. 특히 Learning rate, # of Epochs, Momentum, Regularization constant 등의 숫자들을 Hyper-parameter라고 칭한다.

2.3. Training Data 만들기

2.3.1. Voice Dataset

1) Google Speech Commands Dataset v0.01 [2]

Google에서는 초보자 개발용으로 KWS에 적합한 무료 공개 데이터셋을 제공하였다. 30개의 짤막한 단어에 대해 65000가지의 1초짜리 발성이 포함되어 있는데, 이는 AIY 웹사이트를 통해 수천 명의 일반 사용자가 참여한 결과로 구축한 자료이다. 이 데이터셋은 'YES', 'NO'와 같이 흔히 사용되는 단어와 숫자, 그리고 지시를 포함하여 기본적이지만 유용한 음성 인터페이스를 빌드할 수 있도록 설계되었다.

2) Dataset Spec

언어	길이	성별	sr	단어 종류	총 개수
영어	1초	혼성	16k	30	65000개

3) 단어 종류

bed, bird, cat, dog, down, eight, five, four, go, happy, house, left, marvin, nine, no, off, on, one, right, seven, sheila, six, stop, three, tree, two, up, wow, yes, zero

2.3.2. Feature Extraction : MFCC

1) 흐름도

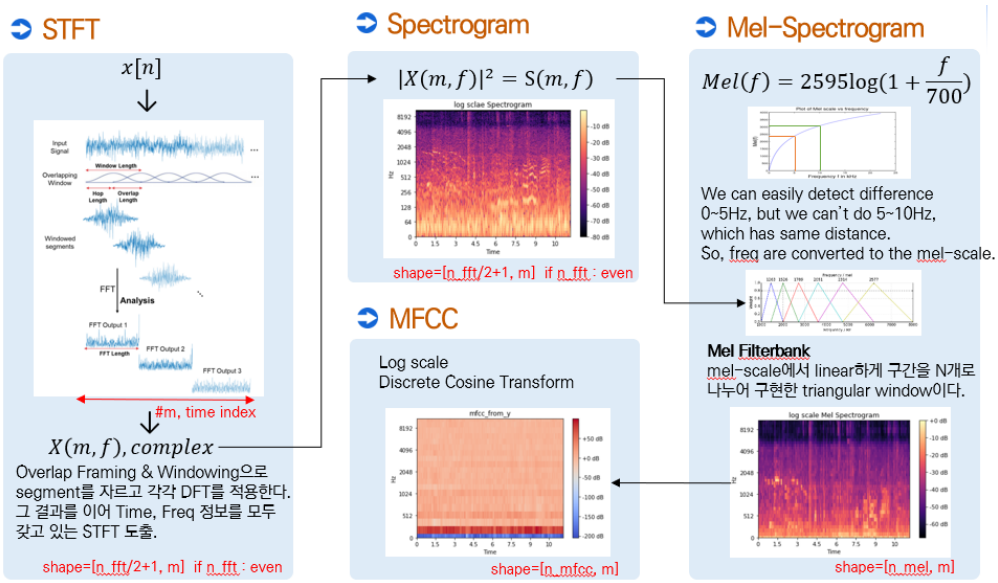


그림 5 MFCC

2) 세부 설명

(1) Wav signal

마이크에서 나온 전기신호가 ADC를 통과하면서 Signal을 Wav 형태로 기록한다. X 축이 Time, Y축이 신호의 Amplitude이다.

(2) STFT

일반적인 FFT는 시간성분이 사라지는 반면에, Time 축에서 Frame 별로 FFT를 진행하여 얻은 STFT는 시간과 주파수 2개의 축으로 이루어져 시간 정보가 담겨있다는 장점이 있다. 그 과정은 다음과 같다.

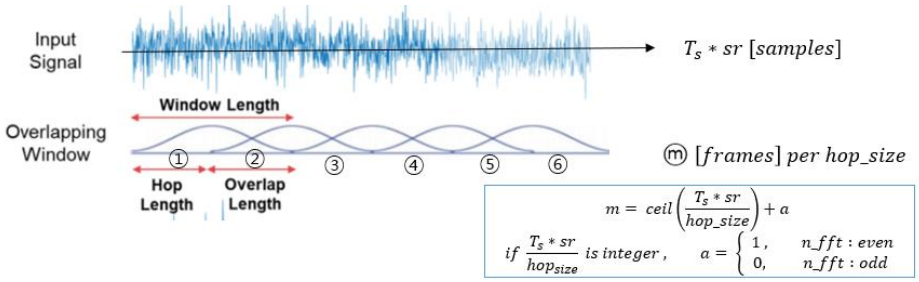


그림 6 STFT Framing and Windowing

Window Length 만큼 Frame을 자르고, 각 Frame에 Windowing을 적용하여 Spectral Leakage를 줄이고 FFT를 적용하여 주파수 도메인으로 변환한다. 이때, FFT을 위한 샘플 개수는 보통 Window Length와 같거나, Zero-Padding을 통해 그보다 큰 2^n 꼴로 설정할 수 있다. 다음 Frame은 Hop Length 만큼 Slide하여 위 과정을 반복한다.

(3) Spectrogram

STFT의 결과는 복소수 형태여서 크기의 제곱형태인 Spectrogram으로 주로 사용된다. 소리나 파동을 시각화하여 파악할 수 있는 특징값으로 파형과 스펙트럼의 특징이 조합되어 있다. 시간축과 주파수 축의 변화에 따라 진폭의 차이를 색상의 차이로 나타낸다. Y축은 Nyquist 이론에 따라 원본 Sampling rate의 1/2만큼이 최대 주파수가 된다.

(4) Mel-Spectrogram

우선, Mel-scale이란 사람의 음을 인지하는 기준을 반영한 Scale 변환함수이다. 이를 구간 n_{mel}개로 나누어 구현한 Triangular Window가 Mel-Filterbank이고, Mel-Spectrogram y축은 곧 Mel-Filterbank의 개수와 같다. 즉, n_{mel}개의 각 Fiterbank들이 커버하는 영역의 주파수를 Triangular Filter로 검출하여 요약한 정보이다.

(5) MFCC

Mel-Specgroram에 Log, DCT 결과를 수행하여 그 중 n_{mfcc}개의 계수만을 추출한다.

2.3.3. Training data : KWS Task [3]

1) 직접 생성하는 과정

10s Background, 1s Activate word, 1s Negative (Not Activate) word를 각 Class에서 랜덤하게 뽑는다. 그리고 pydub.overlay 함수를 이용하여 Background 음원에 선택한 단어를 덮어 씌운다.

Class	Description
Background	10s, with noise
Activate word	0~1s, ex "Stop"
Negative word	0~1s, ex "Two"
Mixed result	Activate insert time = (5107, 6022) Negative insert time = (2176, 2830)

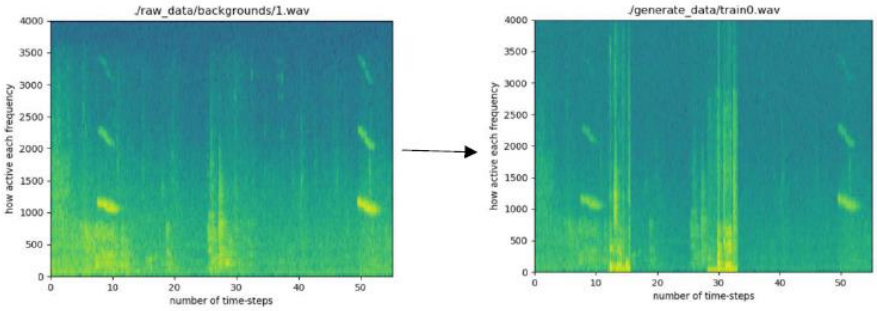


그림 7 Words 삽입 전/후 Spectrogram

2) Input feature, MFCC : (24, 1723)

Mic Bit Rate = 44100 → (down sampling) sampling rate = 22050

hop_legnth 5ms (= 128 samples), n_fft 4.6ms (= 101 samples)

no overlap, n_mfcc = 24

3) Label 정의

Activate word가 삽입된 Time 축에 "1"을 그 외에는 "0"을 Labeling 한다.

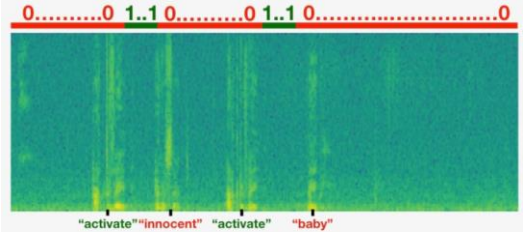


그림 8 생성된 Label 형태

이때, sr = 22050인 10s짜리 음원이 STFT 이후 1723 time index로 차원이 축소된 것을 고려하여 173 samples (=1s) 길이를 Positive Labeling한다.

2.3.4. Training data : 30Words Classifier Task

1) Dataset

Google Speech Commands Dataset v0.01

30개의 단어 * 각 1000개씩 사용

2) Input feature, MFCC : (24, 173)

Raw sr = 16k → (up sampling) sr = 22050

hop_legnth 5ms (= 128 samples), n_fft 4.6ms (= 101 samples)

no overlap, n_mfcc = 24

2.4. 구현한 Deep Learning 모델 구조

2.4.1. KWS Task : GRU 2-Layer

1) RNN 모델

Deep Learning 모델의 종류에는 크게 기본 구조인 (인간의 신경망 구조를 본떠 시작된) DNN과 이미지 패턴에 강한 CNN, 그리고 Sequential Data에 적합한 RNN 모델이 존재한다. 특히 RNN 모델은 아래 그림과 같이 Time Step마다 같은 Hidden State를 반복적으로 사용한다는 특징이 있다.

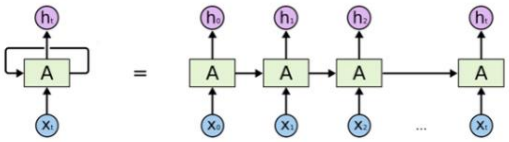


그림 9 RNN 기본 구조

2) GRU (Gated Recurrent Unit) 모델 [4]

RNN 모델은 Time Step이 길어질수록 과거의 정보를 까먹는 장기 의존성 문제를 지니고 있다. 이에, LSTM 모델과 그 간략화 버전인 GRU가 등장하였다.

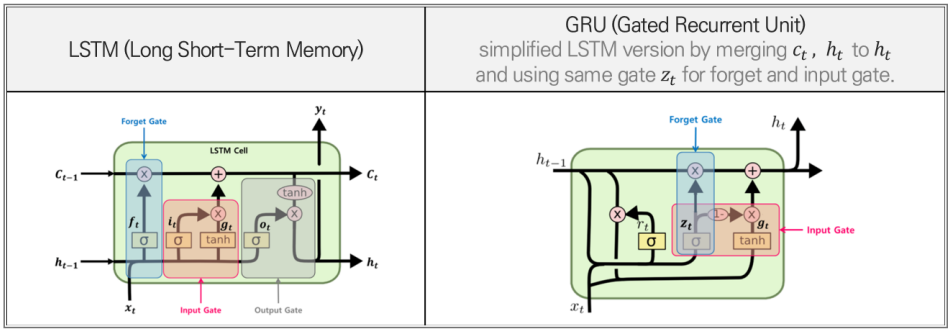


그림 10 LSTM, GRU 구조 [5]

LSTM부터 살펴보자. 크게 3가지 기능이 있다. Gate값을 이용해 기억할 부분, 삭제할 부분, 읽어드릴 부분을 학습하는 것이다.

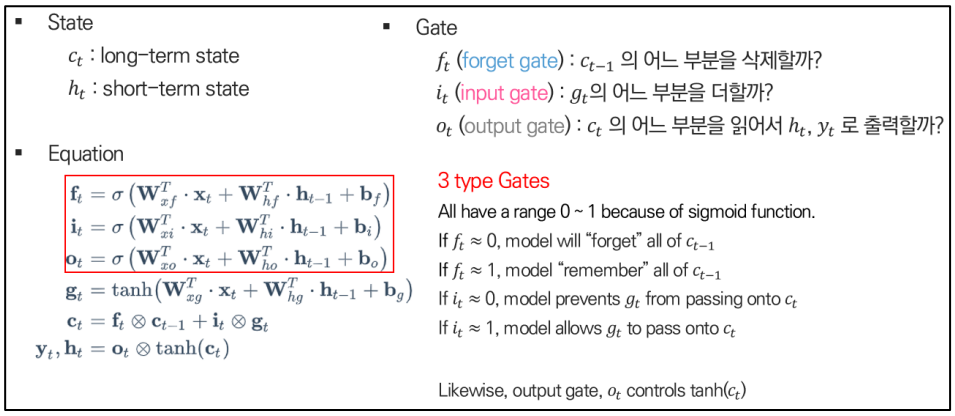


그림 11 LSTM 필요 수식 정리

그리고 매번 은닉상태 업데이트 계산을 줄여 보다 간단한 구조를 갖는 GRU 모델이 등장하였다. LSTM에서는 출력, 입력, 삭제 게이트라는 3개의 게이트가 존재했다. 반면, GRU에서는 업데이트 게이트와 리셋 게이트 2가지 게이트만 존재한다. 따라서 LSTM보다 학습속도가 빠르다.

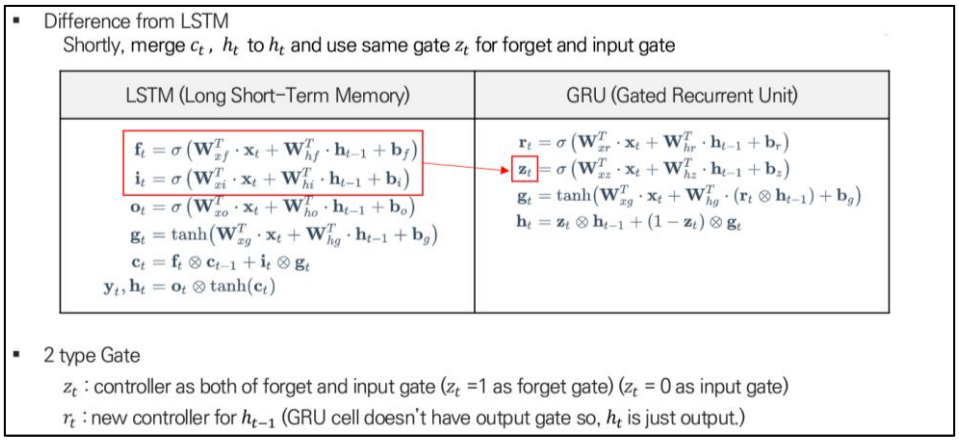


그림 12 GRU 필요 수식 정리

2.4.2. 30 Words Classifier Task : 3-Blocks ResNet

1) ResNet 구조 [6]

Deep Learning은 기본적으로 Layer가 깊어지면 성능이 더 좋아진다는 생각을 가질 수 있다. 하지만, Gradient Vanishing/Exploding 현상으로 성능이 더 나빠질 수 있으며 이러한 이유 때문에 Layer 개수 또한 Hyper-parameter로서 성능에 좌지우지되는 요소이다. 이에, ResNet 구조가 제안되었다.

Plain Network에서는 바로 직전 Layer만이 입력값으로 다음 Layer에 전달이 된다. 하지만, ResNet은 과거 Layer 출력값이 입력값에 더해진다.

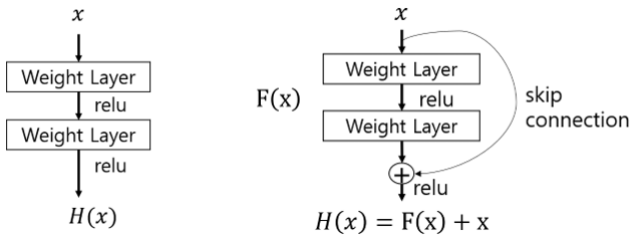


그림 13 (좌) Plain Network, (우) ResNet

ResNet은 $F(x) = H(x) - x$ 가 0이 되는 방향으로 학습을 진행하고 이는, 나머지(Residual)을 학습한다고 볼 수 있어 ResNet이라고 불린다. 만약 x와 H(x)의 차원이 다르면 차원을 맞추기 위해 Identity, x에 추가 연산이 진행될 수 있다.

Cornell 대학에서 발표한 논문[i] 에 따르면 ResNet에 대해 다음과 같이 분석하기도 했다. Plain Network와 달리, 각 Layer의 입력은 다른 구조의 Network의 출력이 된다. 즉, 각 Layer의 Path 길이가 다르며, 각 Layer가 독립적이라 볼 수 있다.

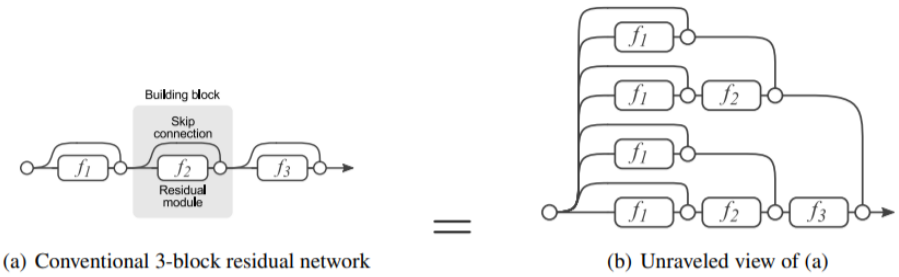


그림 14 3-Block Residual Network [7]

[i] (NIPS 2016) Residual Networks Behave Like Ensembles of Relatively Shallow Networks – Andreas Veit

2.5. Task별 Loss 설정

2.5.1. Multi-class & Multi-Label Task

기본적인 ML 문제들은 Regression이나 Classification으로 나뉜다. 그 중 Classification은 Multi-Class, Multi-Label Task로 분리된다. 다음 그림에서 볼 수 있듯이 Multi-Class Task는 여러 개 중 한 개의 Class만 Positive Label을 가질 수 있고, Multi-Label Task는 여러 개의 Class 객체를 가질 수 있다.

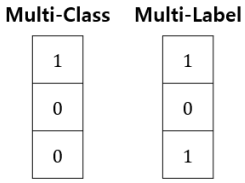


그림 15 Target 예시

2.5.2. Loss for Multi-class Task

1) Categorical Cross-Entropy Loss

$$\text{when } y_i = target,\ \tilde{y}_i = prediction$$

$$Loss = -\sum_{i=1}^{C} y_i \times \log(p_i), \qquad p_i = \frac{\exp(\tilde{y}_i)}{\sum_{j=1}^{C} \exp(\tilde{y}_i)}$$

이때, Negative Log Likelihood의 특성을 이용하여, 잘못 예측했을 경우 무한대의 손실, 일치할 경우 0이 나온다. Softmax Activation 뒤에 Cross-Entropy Loss를 주로 사용해서 Softmax Loss라고도 불린다. Class가 2개일 경우 Binary Cross-Entropy라고 볼 수도 있다.

2.5.3. Loss for Multi-label Task

1) Sigmoid Binary Cross-Entropy

$$\text{when } y_i = target,\ \tilde{y}_i = prediction = \frac{1}{1+e^{-x}} = \delta(x)$$

$$Loss = -\sum_{i=1}^{C} [y_i \times \log(\tilde{y}_i) + (1 - y_i) \times \log(1 - \tilde{y}_i)]$$

N개의 항목을 갖는 분류 문제에 대해 신경망의 마지막 계층에 모두 Sigmoid 함수를 적용하고 각 항목마다 Binary-Cross-Entropy Loss를 구한다.

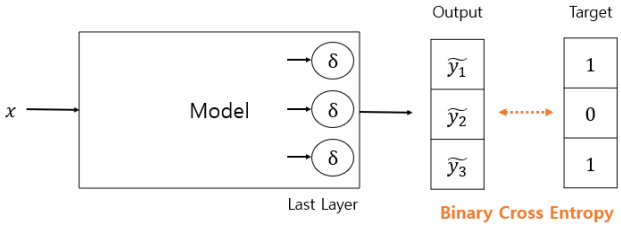


그림 16 Loss for Multi-Label Task

2.6. Optimizer

2.6.1. Optimizer 종류

We can train parameters (W, b) to reduce cost with various optimizer.

Standard Gradient Descent	$W := W - \alpha * dW$ gradient 반대방향(cost가 줄어드는) 으로 learning rate, α 곱해 Parameter Upgrade.
Stochastic /Mini-Batch G.D	Batch Training needs to long per iteration. So, mini-batch concept occurs. - Stochastic G.D : each 1-example mini batch, upgrade parameter Lose speedup because without vectorization. - Mini-batch G.D : each N-example mini batch, upgrade parameter. It should trend downwards with noise
Momentum	Compute with Exponentially Weighted Moving Average. $V_{dW} = \beta_1 V_{dW} + (1-\beta_1)(\frac{dJ}{dW}) \rightarrow W := W - \alpha V_{dW}$ It reduces oscillations of GD because of store previous gradients (direction).
RMSprop	$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)\left(\frac{dJ}{dW}\right)^2 \rightarrow W := W - \alpha * \frac{dW}{\sqrt{S_{dW}}}$
Adam (RMSprop + Momentum)	$V_{dW} = \beta_1 V_{dW} + (1-\beta_1)(\frac{dJ}{dW})$, $V_{dW}^{corrected} = \frac{V_{dW}}{1-\beta_1^t}$, $S_{dW} = \beta_2 S_{dW} + (1-\beta_2)\left(\frac{dJ}{dW}\right)^2$ $W := W - \alpha * \frac{dV_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}} + \epsilon}$

2.6.2. Adam Optimizer [8]

주요 장점은 Step Size가 Gradient의 Rescaling에 영향을 받지 않는다. 즉, Gradient가 커져도 Step Size가 Bound 되어있어서 안정적으로 Optimizing 하강이 가능하다. 또한, Step Size를 과거의 Gradient의 크기를 참고하여 적용시킬 수 있다.

2.7. Regularization

Training Data에 Overfit 되는 현상을 해결하자.

2.7.1. Batch Normalization [9]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // mini-batch mean

$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$ // mini-batch variance

$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ // normalize

$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$ // scale and shift

그림 17 Batch-Norm 수식

학습을 진행함에 있어 Network 각 층이나 Activation마다 Input의 Distribution이 달라지는 Internal Covariance Shift현상이 발생한다. 이에, 각 Feature마다 z-norm을 통해 평균 0, 표준편차 1인 Input으로 바꿔주는 작업을 진행한다. 하지만 이 과정은 Activation Function의 Non-linearity를 없앨 수 있기 때문에 Scale factor(γ)와 shift factor(β)를 적용해준다. 이 값들은 Trainable 하다. 이에, Gradient Vanishing/Exploding 방지하여 안정적인 학습 가능하다.

2.7.2. Weight Decay [10]

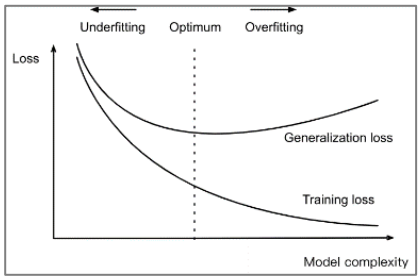


그림 18 Model Complexity and Loss

Overfitting은 Model의 Complexity가 높아지는 것으로 해석할 수 있다. 이는, Model의 Parameter Weight (차수)가 증가한다는 의미로, 이를 작게 유지하기 위해 손실함수에 이 값을 패널티로 더하는 것이다. 수식은 다음과 같다.

$$L(w,b) := L(w,b) + \frac{\lambda}{2}||w||^2$$

(λ 값은, Regularization 정도를 조절하는 Hyper-Parameter이다.)

2.7.3. Drop out [11]

Weight Decay는 학습 과정에서 네트워크 연산 경로에 노이즈를 집어넣는 것이다. 이때, 지나친 편향을 추가하지 않으면서 노이즈를 추가하는 것이 관건이다.

$$\epsilon \sim N(0,\delta^2),\ x' = x + \varepsilon \rightarrow E[x'] = E[x] \cdots ①$$

하지만, 중간층에서는 노이즈의 스케일이 적절치 않기 때문에 Weight Decay 적용 후 위 조건이 충족되지 않을 수 있다. 그 대안으로 Dropout 아이디어가 등장하였다. 은닉 뉴런을 p 확률로 누락(Drop out) 시킨다고 했을 때

$$x' = \begin{cases} 0, & \text{p에 속해 버려질 Node/Edge} \\ \dfrac{x}{1-p}, & \text{그 외 살아있는 Node/Edge} \end{cases}$$

$$E[x'] = p * 0 + (1-p) * \frac{E[x]}{1-p} = E[x]$$

수식① 을 만족하면서 Regularization이 가능하다.

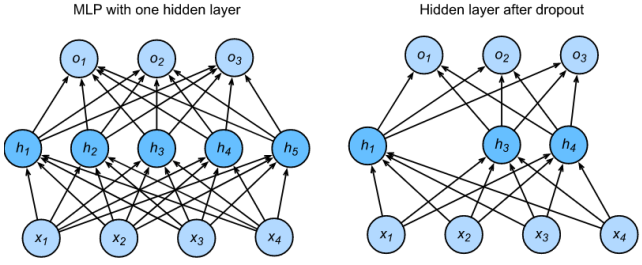


그림 19 Dropout 적용 전/후

2.8. Metric [12]

2.8.1. Confusion Matrix

1) Classifier의 성능을 평가하기 위해 사용되는 지표이다.

Confusion Matrix (Count)		Predicted Class	
		Positive	Negative
Label Class	Positive	TP	FN
	Negative	FP	TN

- ✓ TP : 예측에 성공하고, 추정값 Positive (Label=1)
- ✓ TN : 예측에 성공하고, 추정값 Negative (Label=0)
- ✓ FP : 예측에 실패하고, 추정값 Positive
- ✓ FN : 예측에 실패하고, 추정값 Negative

2) Multi-label Task

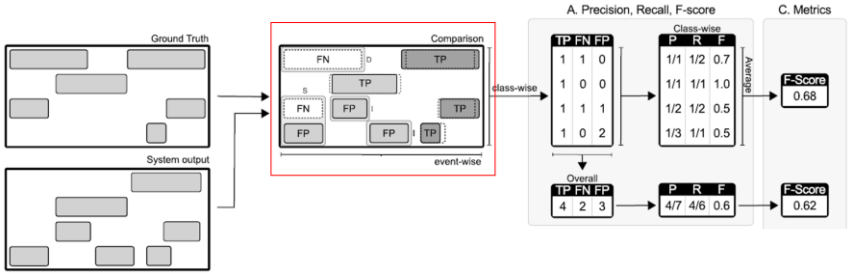


그림 20 Confusion Matrix and Metrics

2.8.2. Accuracy, Precision, Recall, F1-score

1) Accuracy (정확도)

1을 1로 0을 0으로 정확하게 분류해내는 것을 의미한다. 모델이 얼마나 정확한지를 평가하는 척도

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

2) Precision (정밀도, 양성 예측도)

모델을 통해 1이라고 분류한 그룹 A가 있을 때, 신뢰도 계산

$$Precision = \frac{TP}{TP + FP}$$

3) Recall (재현도)

정밀도와 비교되는 척도로써, 관심 영역만을 얼마나 추출했는지

$$Recall = \frac{TP}{TP + FN}$$

4) F1 score : Precision과 Recall을 함께 평가하는 수치로써, 둘의 조화평균이다.

$$F1\ score = 2 * \frac{Precision \times Recall}{Precision + Recall}$$

2.9. Training 결과

2.9.1. KWS Task

1) Train, Valid, Test Dataset 비율

Total	Train	Valid	Test
5000	4000 (80%)	800 (16%)	200 (4%)

2) Model Baseline

(1) Input.shape = (batch_size, Tx=1723, in_channels=n_mfcc=24)

(2) Layer Spec

① Conv1D

(kernel_size=3x3, stride=1, padding=1, out_channels=24)

② GRU

number of GRU layers = 2,

hidden_cell_size = 128

(2번째 Layer는 첫번째 Layer에서 학습된 hidden_cell을 사용한다)

③ Others

Conv1D, GRU layer 이후에 BatchNorm, ReLU 혹은

Dropout(drop_prop=0.2) layer 를 적용한다.

(3) Output.shape = (batch_size, Tx=1723)

Fully_connected로 GRU hidden_cell size로 확장되었던 width를 줄인다.

(4) Optimizer

Adam Optimizer (lr=1e-3, weight_decay(=λ) 1e-3)

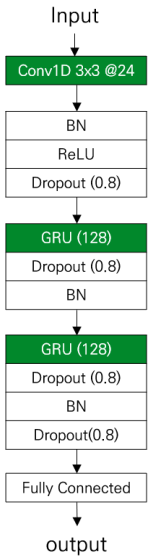


그림 21 KWS Task Baseline

3) Metric

(1) F1-score

(2) Loss : BCEloglogitsLoss

4) 실험 결과 (위 Model spec 적용)

(1) Hyper-parameter Tuning에 따른 결과 정리

F1-score	drop_prop = 0.2 λ = 0.001	drop_prop = 0.2 λ = 0.0001	drop_prop = 0.3 λ = 0.001
Train	0.971	0.925	0.683
Valid	0.699	0.663	0.235
Test	0.691	0.659	0.238

(2) Best Result

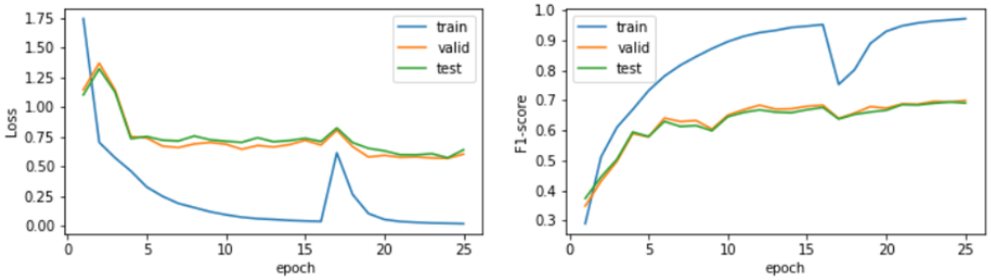


그림 22 Loss and F1-Score for KWS Task

2.9.2. 30 Words Classifier Task [13]

1) Train, Valid, Test Dataset 비율

Total	Train	Valid	Test
30000= 30*1000	27000 (90%)	2100 (7%)	900 (3%)

2) Model Baseline

(1) Input.shape = (batch_size, chnnel=1, Tx=173, n_mfcc=24)

(2) Layer Spec

① Conv2D 3x3 filters

입력과 출력의 Feature Map 크기는 유지하되 Channel 수를 2배씩 (16 → 32 → 64) 늘린다. 채널 수를 늘릴 때는, Feature Map의 크기는 절반씩 줄어든다. 즉, Feature Map 크기를 유지하기 위해서는 stride =1, 절반씩 줄이기 위해서는 stride=2를 적용한다. 만약 stride=2를 적용하였다면, Shortcut Connection Tensor에도 stride=2를 적용하여 같은 Conv2D 연산을 통해 Down-sample 해주어야한다.

② Basic Block

1개의 Block 내부에는 2번의 Shortcut Connection을 진행한고 입력과 출력의 Feature Map 크기를 유지한다. Skip connection 이후에 ReLU와 Dropout(drop_prop=0.2) Layer를 적용한다는 특징이 있다.

③ Output.shape = (batch_size, 30)

Fully_connected Layer로 최종 결과에서 Class 개수만큼으로 줄이고 Softmax 연산 결과를 출력한다.

(3) Optimizer

Adam Optimizer (lr=1e-3, weight_decay(=λ) 1e-3)

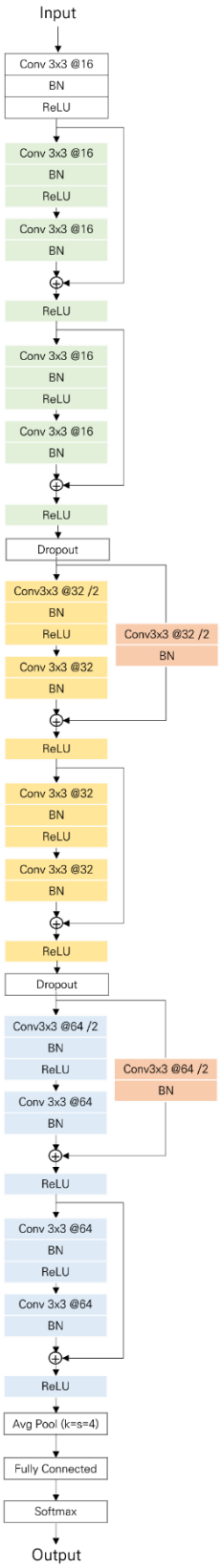


그림 23 30 Words Classifier Task Baseline

3) Metric

(1) Accuracy

(2) Loss : CrossEntropyLoss = LogSoftmax + Negative Log Likelihood Loss

4) 실험 결과 (위 Model spec 적용)

(3) Hyper-parameter Tuning에 따른 결과 정리

Accuracy	drop_prop2 = 0.1	drop_prop2 = 0.2
Train	96.985	92.348
Valid	91.381	90.762
Test	89.667	89.778

(4) Best Result

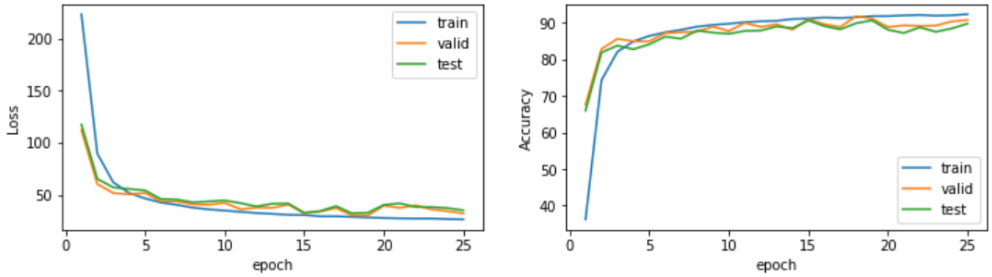


그림 24 Loss and Accuracy for KWS Task

5) Confusion Matrix (Row : Target, Column : Predict)

Test Dataset으로 확인한 결과이다. 데이터 분포는 고르게 퍼져 있고 Classifier가 높은 정확도의 성능을 보임을 가시적으로 확인할 수 있었다.

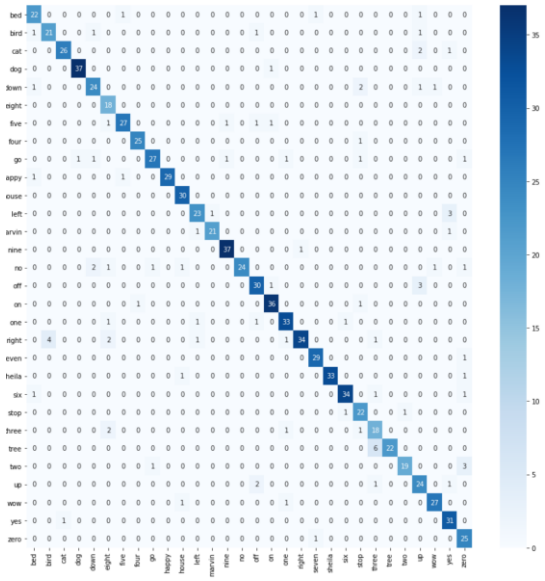


그림 25 Confusion Matrix for Test Dataset

2.10. Server Interface

2.10.1. Django

장고(Django)는 Python으로 작성된 오픈 소스 웹 프레임워크로, 모델-뷰-컨트롤러(MVC) 패턴을 따르고 있다.

고도의 데이터베이스 기반 웹사이트를 작성하는 데 있어서 수고를 더는 것이 장고의 주된 목표이다. 장고는 콤포넌트의 재사용성(reusability)과 플러그인화 가능성(pluggability), 빠른 개발 등을 강조하고 있다. 또한, "DRY(Don't repeat yourself: 중복배제)" 원리를 따랐다. 설정 파일부터 데이터 모델에까지 파이썬 언어가 구석구석에 쓰였다.

인스타그램, NASA, 빗버킷, Disqus , 모질라에서 장고를 사용하는 것으로 알려져있다.

2.10.2. 구현 결과

1) Blog 형태

기록을 확인하기 위해서는 블로그 형식을 취하는 것이 익숙하고 눈으로 확인하기 편하기 때문에 아래와 같이 인터페이스 형태를 구축하였다.

각 HTML마다 동일한 부분은 'base_generic.html'을 만들어 수정할 부분이 생기면 모든 HTML 파일을 수정할 필요 없이 하나의 파일만 수정할 수 있도록 하였다.

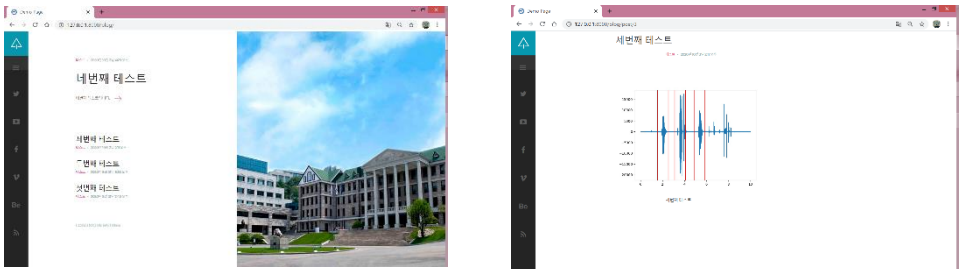


그림 26 Home & Content

2) 기록

기록하고 싶은 그래프와 Extraction 파일을 서버에 업로드하여 향후 지속적으로 결과를 확인할 수 있다.

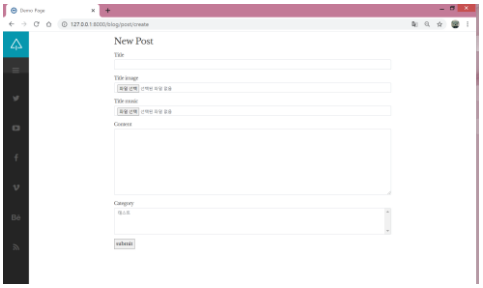


그림 27 Post

3. 최종 결과물

3.1. 구현 방식, 기능 설명

3.1.1. 최종 개발 코드 – GitHub 저장소

https://github.com/ddubbu/Univ-Grad-Portfolio

3.1.2. 실습 환경 세팅

1) Anaconda 설치 필요성 [14]

Anaconda는 프로젝트간 가상환경을 관리해주는 장치이다. 특히 ML Task에서는 다양한 라이브러리를 사용하게 되고, 이들은 다양한 의존성을 가지고 있어 버전 관리가 매우 중요하다. 이에, 특정 버전을 유지하며 개발을 진행하기 위해 Anaconda와 같은 프로그램이 필요한 것이다.

2) Local 가상 환경 만들기

(1) Package 버전

Windows 10 / (IDE) Pycharm / Python 3.5 / PyTorch 1.5.1 / 기타 아래와 같음.

audioread	2.1.5	PyAudio	0.2.11
certifi	2018.8.24	pydub	0.24.1
cycler	0.10.0	pyparsing	2.4.7
decorator	4.4.2	python-dateutil	2.8.1
Django	2.2	pytz	2020.1
joblib	0.14.1	resampy	0.2.2
kiwisolver	1.0.1	scikit-learn	0.20.0
librosa	0.6.3	scipy	1.1.0
llvmlite	0.31.0+0.gfe7d985.dirty	setuptools	40.2.0
matplotlib	3.0.0	six	1.15.0
numba	0.47.0+0.g4eb9cf8.dirty	sqlparse	0.3.1
numpy	1.14.2	torch	1.5.1
olefile	0.46	torchvision	0.6.1
pandas	0.25.3	tornado	5.1.1
Pillow	5.2.0	wheel	0.35.1
pip	10.0.1	wincertstore	0.2

그림 28 설치한 Package 버전

(2) Anaconda로 최종 Stable 환경 구축 과정

```
system path> conda create -n “project name”

system path> conda activate “project name”

(project name)> conda install python=3.5

(project name)> conda install -c numba numba

(project name)> conda install -c conda-forge librosa=0.6.3

(project name)> conda install pytorch==1.5.1 torchvision cpuonly -c pytorch

(project name)> pip install pyaudio pydub matplotlib pandas

(project name)> pip install django==2.2
```

* 추가 설치 패키지 : main_GUI_version.py 용

playsound, PyQt5

(3) Stable 환경 구축의 중요성 : Package간 호환성 문제

Package 하나를 설치하면 Required Package가 자동 설치된다. 여기서 바로, 호환성 문제가 발생함을 알게되었다. 특히 ML을 위한 PyTorch나 Tensorflow와 같은 모듈은 librosa보다 먼저 설치하는 경우 librosa가 제대로 설치되지 않는다. 그래서, package간에 stable 버전 및 선행 설치 패키지 조사가 필요했다. 추가적인 팁으로 최신버전은 깔지 않는 것이 좋다.

3) 모델 학습 시 GPU 사용을 위한 Google Colaboratory 환경 설정

(1) 순서 [15]

① Google Drive 접속 → 작업공간 생성 → Colaboratory 연결

② Colaboratory 접속

③ [런타임] > [런타임 유형 변경] > 하드웨어 가속기 : GPU 선택

④ *.ipynb 파일에서 오른쪽 상단 [연결] 클릭 후 기다리기

⑤ 소스코드 작성 후 학습할 Tensor GPU에 올리기

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

model = CNN_Net(p=drop_prob).to(device)
x, target = Variable(x).to(device), Variable(target).to(device)

output = model(x)
loss = F.nll_loss(output, target).to(device)
```

3.1.3. 구현 과정 및 기능 설명

① 10초 동안 음원을 녹음한다. 녹음된 음원을 KWS 모델을 이용하여, 키워드 “STOP” 이 존재할 위치를 찾아낸다. 위 모델은 입력 데이터가 모두 “STOP”이 존재한다는 가정하에 학습되었기 때문에 보조 모델로 Classifier가 사용된다.

② KWS 모델이 예측한 확률 최대값을 기준으로 “STOP”이라고 추정되는 1초짜리 단어를 30words Classifier에 넣어준다.

③ 그 결과가 Threshold (0.6) 이상일 때만, Keyword가 존재한다고 판단하여, STT가 활성화된다.

④ 10초 녹음파일이 Peak를 기준으로 1초짜리 N개의 단어로 추출되어 30 Words Classifier에 들어가고 그 결과를 문장으로 출력한다.

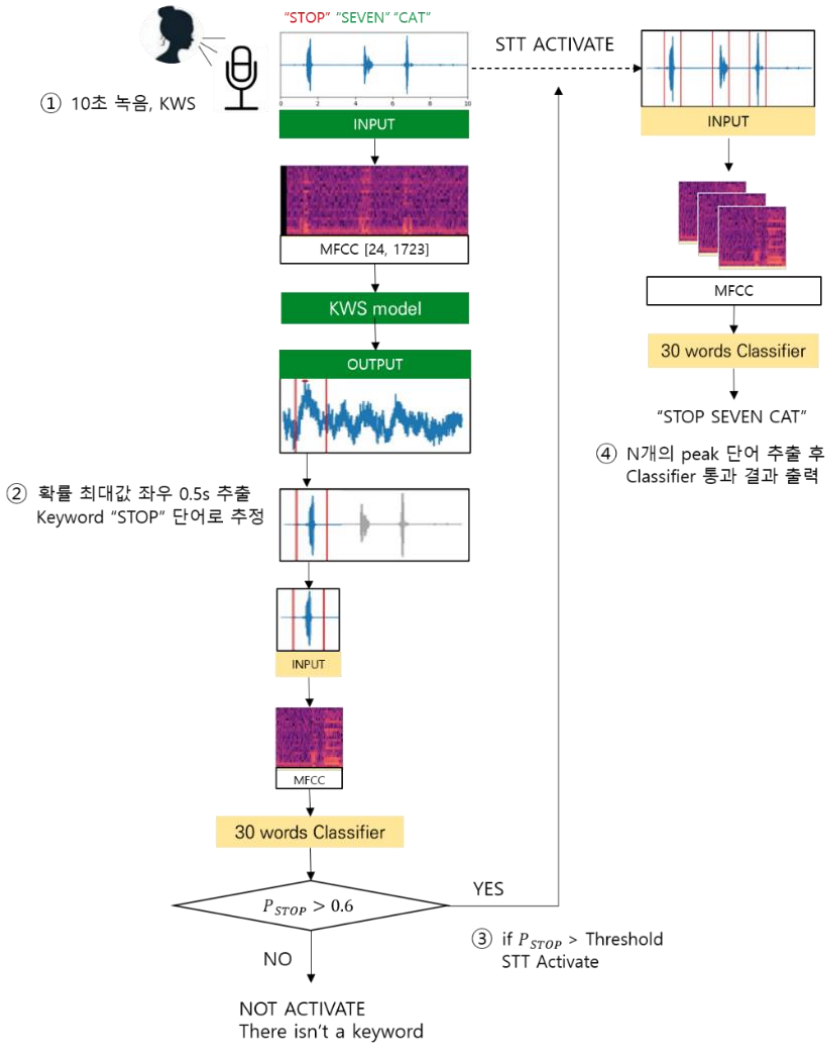


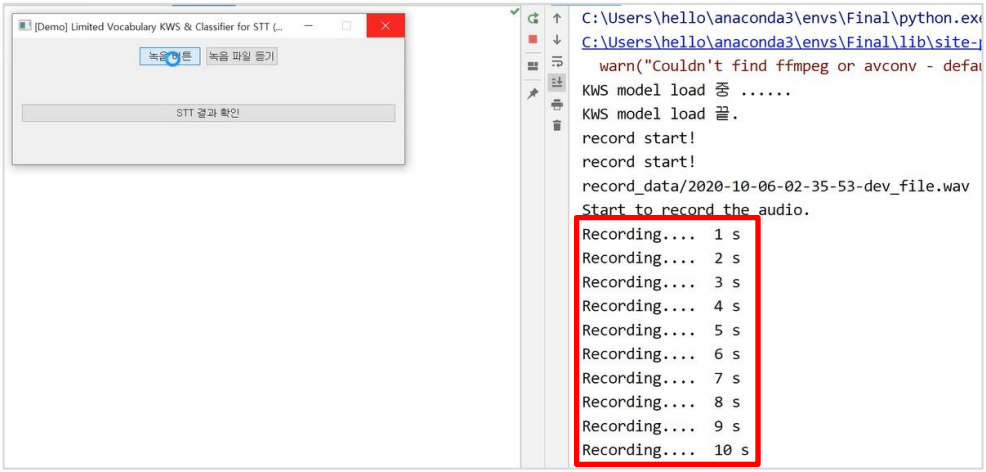
그림 29 전체 흐름도

3.2. 결과 산출물 – 프로그램 리스트에서 main_GUI_version.py 실행.

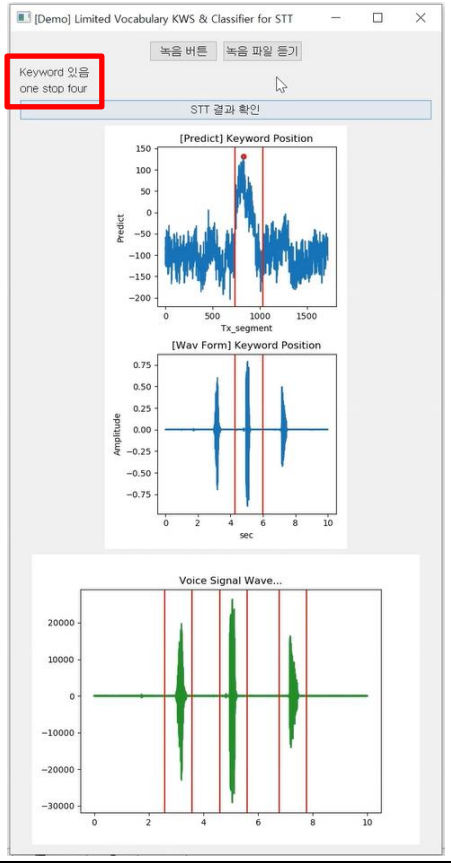
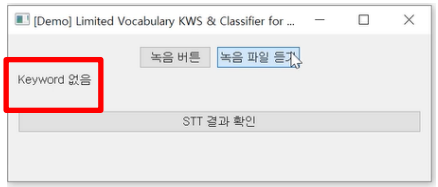
3.2.1 발표 및 시연 영상 링크 : https://youtu.be/mAukIRp3yaU

3.2.2 시연 영상 스크린샷

1) 10초 녹음



2) 결과

Keyword 있는 경우	Keyword 없는 경우
	

4. 결론

이 작품은 현존하는 AI 음성 인식기의 기능 2가지 KWS, STT를 구현하기 위해 2가지 Task를 구상하게 되었다. 학습 데이터 생성부터 Baseline 구축, Hyper-parameter Tuning 및 학습, 결과 저장 등 최적의 조합을 찾기 위해 오랜 시간 공부를 진행했다. 그 결과는 다음과 같다.

4.1. KWS : Multi-Classification Task

GRU 2 layer를 Baseline으로 삼았다.

이때, 첫번째 layer에서 학습된 hidden_cell을 2번째 layer로 넘겨줌으로써 학습의 성능을 높일 수 있었다. F1-score를 성능지표로 삼았으며, Train 0.971, Test 0.691로 꽤나 차이가 큰 Overfitting 결과를 보여, drop_prop을 더 키워보았지만, Test 0.238까지 확 떨어지는 것을 보고, 또 다른 Regularization 방법을 공부할 필요성을 느꼈다.

낮은 정확도 때문인지 몰라도, 이 KWS 모델을 가지고 “Stop” Keyword 인식을 진행하면, 일부는 “Up” 단어로 인식되는 경우도 있었다. 이러한 문제의 발생 원인으로 2가지를 생각해보았다. 첫번째, “Stop”은 2음절, “Up”은 1음절 단어로 전자의 경우 느리게 말하면서 [스타-압]과 같이 음절이 분리되기도 한다. 이로 인해 “Up”으로 인식되는 상황이 발생한 것 같다. 뿐만 아니라 Input Feature로 MFCC를 선정하였는데, STFT를 위한 Framing 진행 시 Overlap 없는 Feature를 추출하여 위 음절이 분리되는 현상을 잡지 못하지 않았나 하는 문제점 또한 생각해볼 수 있었다.

4.2. 30 Words Classifier : Multi-Label Task

실은, 음성 데이터를 적용하기 전에, MNIST 이미지 데이터를 이용해 Classifier를 구현해보았다. 이때는 단순한 DNN, CNN 모델로도 높은 정확도를 보이기에 이 모델을 가지고 음성 데이터에 적용해보았다. 하지만, 어떤 Layer를 쌓아도 정확도 70% 이상의 결과를 갖기가 어려웠다.

그러던 중 Layer가 깊어지며 성능을 높일 수 있는 ResNet 모델을 공부하게 되었고 이를 baseline으로 선정하게 되었다. 그 덕분에 바로 8-90% 이상의 정확도를 확인할 수 있었다. 하지만, 모델의 Parameter 수가 증가한 만큼 모델 저장 시 차지하는 용량이 거대해지는 단점 또한 발견할 수 있었다.

4.3. 기타 알게된 점

2 Task 모두 Valid, Test Dataset에 대한 결과가 비슷하게 나와, 분포가 비슷한 Dataset을 갖고 있음을 확인할 수 있었다. 또한, 실제 구현할 최종 프로그램은 10초짜리의 1가지 음원이기 때문에, batch_size=10으로 최소화하였다.

5. 졸업작품 후기

5.1. 김선미

우선, 나만의 음성 인식기를 만들었다는 점이 매우 뿌듯했다. 1월부터 9월까지 장장 9개월 간의 공부 과정이 이 작품에 녹아 들어있다. Python 기초부터, 인공지능개론, 신호처리 ~ DSP 복습, Coursera Deeplearning.ai, Tensorflow, PyTorch 실습까지 ML에 대한 노베이스에서 현재까지 도달했다. 사실, 이번 졸업작품의 결과는 나의 자신감을 되찾고 대학원 진학에 대한 스스로의 척도가 될 것이라고 다짐했다. 헌데, 이 정도의 구현 역량이면 나도 진학해볼 수 있겠다는 자신감을 되찾게 해주었다. 이 경험을 시작으로 앞으로 진행할 공부와 연구에서 실패를 경험해도 다시 일어설 수 있는 칠전팔기 정신을 잊어버리지 않았으면 좋겠다.

7월부터 장준혁 교수님 지도 하에 ASML 인턴 생활을 하고 있다. 그 덕분에 ResNet 모델을 공부해볼 수 있는 기회를 가졌고, 이 졸업작품에서 구현하여 좋은 성능을 찍을 수 있었다. 또한, 공부에 속도를 잃지 않고 꾸준히 새로운 것을 배울 수 있었다. 한가지 예시로, 처음 졸업작품을 시작했을 당시 wav 음원으로부터 MFCC Feature를 뽑기 위해 librosa 모듈을 사용하였는데, 각각의 함수 인자의 의미도 알지 못한 채 사용하였다. 하지만, 지금은 그 의미가 무엇인지 알고 응용할 수 있게 되었다. 이처럼 처음에는 모르고 사용한 지식들을 인지해두면, 나중에 공부할 기회가 있을 때 깊게 공부하는 여러 경험들이 너무 재밌고 신기했다. 대학원에 진학해서도 이러한 과정을 경험할 생각에 설레는 감정이 들기도 했다.

또한 다른 관점으로 보면 “대학원을 진학한다는 것”은 내가 모르는 것을 인지하는 고통의 과정일 수도 있겠다. 졸업작품을 마무리할 즈음에 마이크 녹음을 위한 bitrate의 의미를 알게 되었다. bitrate는 sampling rate였고, 나는 그것도 모른 채 모든 데이터의 sampling rate를 기본값 22050으로 연산하였다. 그러다 보니 학습 데이터와 구현(dev) 데이터 간에 spec이 부적절했음을 깨달았다. 하지만, 그에 대한 수정을 하기에는 늦은 상태였다.

이렇게 공부의 기쁨과 쓴 맛을 모두 맛봄으로써 졸업작품을 통해 대학원 진학에 대한 고민을 다방면으로 해볼 수 있었던 것 같다. 다만, 내가 진학해서 어떤 분야를 깊게 공부할지는 아직 모르겠다. 졸업작품을 위해 크게는 ML, 작게는 음성데이터를 가지고 ResNet 모델까지 구현해보는 수준에는 도달했지만, 그럼 다음에는 무엇을 공부하면 되지라는 큰 그림이 보이지 않았다. 현존하는 모든 Model을 구현해보면, 수학적 정의를 이해하면, 새로운 Loss 함수를 공부하면 되려나? 지금의 생각으로는, ML을 모든 분야에 어떻게 적용해서 Task를 정의하면 좋을지 공부하고 싶다. 진학하면 BIO 팀에서 그 데이터에 적합한 Task를 공부하겠지만, 그 외에도 음악 취향 분석, 자동 추천 시스템, 음성인식,

전기세를 줄이기 위한 건물 내 에너지 효율 계산 등. 다양한 상황에서 적용할 수 있는 방법을 공부하고 싶다. 이에, 요즘은 Auto ML이 뜨고 있다는데, 이게 내가 나아가야할 방향인가 싶다.

5.2. 원혜진

졸업작품을 내가 완성할 수 있을까 의문이었는데 어떻게 완성한 것을 보니 얼떨떨하다고 아쉬운 점도 많다. 초기의 목표였던 웹 서버와의 연동은 공부를 해보니 관련 자료가 부족하여 결국 인터페이스의 큰 뼈대만이 만들어졌을 뿐이다. Python과 연동이 좋다는 Django를 이용하자는 것은 좋은 아이디어라고 생각했는데, 이와 관련하여 HTML, CSS, Bootstrap 등 경험하지 못했던 분야를 추가적으로 계속 공부해야 하고 예상치 못한 오류가 발생하는 등 문제점이 발생하였다. 그러나 덕분에 Deep Learning 뿐만 아니라 Web에 대해서 궁금한 점이 무척 많았는데 이에 대한 기초를 쌓고 실습을 해 볼 수 있었다는 점에서 무척이나 즐거웠고 뿌듯했다.

6. 프로그램 리스트

Univ-Grad-Portfolio

- dev_output
  - STT extractions
- model
  - Classifier
    - Classifier_ResNet_ckp.tar
    - ResNet.py
  - KWS
    - GRU_2layer.py
    - KWS_GRU_2layer_ckp.tar
- record_data
- train
  - Classifier
    - 30words_num=30000.npy
    - main_Classifier_PyTorch_cv.ipynb
  - KWS
    - XY_test
    - XY_train
    - main_KWS_PyTorch_cv.ipynb
  - generate_data
    - Classifier_gen_train_data.py
    - KWS_gen_train_data.py
- mysite
  - blog
  - mysite
- inference.py
- keyword.wav
- main.py
- main_GUI_version.py
- wav_utils.py

* 참고사항

- Model train의 경우 Google Colaboratory에서 진행하였습니다.
- 실행은 main_GUI_version.py 에서 가능.

7. 참고 문헌

[1] http://news.mk.co.kr/v2/economy/view.php?year=2018&no=573362

[2] https://developers-kr.googleblog.com/2017/09/launching-speech-commands-dataset.html

[3] https://www.coursera.org/learn/nlp-sequence-models

[4] https://arxiv.org/abs/1406.1078

[5] https://excelsior-cjh.tistory.com/185

[6] https://m.blog.naver.com/laonple/221259295035

[7] https://arxiv.org/pdf/1605.06431.pdf

[8] https://dalpo0814.tistory.com/29

[9] https://shuuki4.wordpress.com/2016/01/13/batch-normalization-설명-및-구현

[10] https://ko.d21.ai/chapter_deep-learning-basics/weight-decay.html

[11] https://ko.d2l.ai/chapter_deep-learning-basics/dropout.html

[12] https://yamalab.tistory.com/50

[13] https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py

[14] https://www.anaconda.com/products/individual#download-section

[15] https://theorydb.github.io/dev/2019/08/23/dev-ml-colab/