

Natural Language Processing Model development
using Convolutional Neural Network in
Thai speech command for controlling robot

Arunwat Moonbung

March 02, 2022

Contents

Chapter 1: Introduction	1
Chapter 2: Thai Speech Command Dataset Preparation	2
2.1 กระบวนการอัดเสียงเพื่อทำชุดข้อมูลสำหรับการพัฒนาโมเดล	3
2.1.1 การอัดเสียงคำพูด (Voice Recording)	3
2.1.2 การทำเครื่องหมายกำกับเสียงเพื่อแยกคำ (Labelling)	4
2.1.3 การ Export ไฟล์เสียง	5
2.2 รูปแบบการจัดไฟล์เดอร์สำหรับชุดข้อมูล	6
Chapter 3: Thai Speech Command Training Process	7
3.1 Audio Pre-Processing	7
3.1.1 การทำ Audio Augmentation	9
3.1.2 รูปแบบของชุดข้อมูลที่พร้อมใช้งาน	9
3.2 Model Definition	10
3.3 Model Fitting	11
3.3.1 การแบ่งสัดส่วนข้อมูล (Train – Validation Split)	11
3.3.2 การกำหนด Hyperparameter	11
3.3.3 การกำหนด Callback Function	13
3.4 Model Inference and Data Post-processing	14
Chapter 4: Thai Speech Command Real-time Inference	16
4.1 วิธีใช้งานไฟล์ SpeechControlTH_RT.py เพื่อเริ่ม Real-time Inference	17
Chapter 5: Discussions & Suggestions	19
5.1 การเก็บข้อมูลเพื่อเพิ่มประสิทธิภาพโมเดล	19
5.2 ปัญหาที่อาจเกิดขึ้นในการใช้งาน Real-time Inference	20
5.3 แนะนำแนวทางในการนำไปใช้งาน	22
References	23

Chapter 1

Introduction

เมื่อปี 2017, Google ได้ปล่อยชุดข้อมูลที่ชื่อว่า **Speech Command Dataset** ข้อมูลชุดนี้ประกอบไปด้วยคำสั่งๆ 30 คำ เป็นไฟล์เสียงความยาว 1 วินาที ซึ่งมีมากกว่า 65,000 ไฟล์ แยกคำเป็นโฟลเดอร์ และแบ่งเป็นทั้งชุดข้อมูลสำหรับการพัฒนาโมเดล และชุดข้อมูลสำหรับการทดสอบประสิทธิภาพ อีกทั้งยังได้มอบโค้ดตัวอย่างในการพัฒนาโมเดล และได้เปิดการแข่งขันให้พัฒนาโมเดลโดยใช้ชุดข้อมูลดังกล่าวด้วย เรียกได้ว่าเป็นจุดเริ่มต้นที่ทำให้ Convolutional Neural Network ได้ถูกนำมาใช้ในงานทางการ Speech-Audio recognition ภายหลังชุดข้อมูลดังกล่าวได้ถูกนำมาใช้อย่างแพร่หลาย โดยเฉพาะการพัฒนาโมเดลสำหรับการควบคุมหุ่นยนต์ บางคำในชุดข้อมูลนี้ เช่น Go, Stop, Left, Right, Up, and Down ได้ถูกนำมาใช้เพื่อการพัฒนาโมเดลสำหรับควบคุมหุ่นยนต์ ผู้จัดทำจึงต้องการพัฒนาโมเดลคำสั่งควบคุมหุ่นยนต์ชุดคำสั่งภาษาไทย โดยใช้ Convolutional Neural Network ในการพัฒนาโมเดลตามแนวทางที่ได้กล่าวมาข้างต้น โดยผู้จัดทำได้ทดลอง Train โมเดลโดยใช้ชุดข้อมูลภาษาอังกฤษก่อนโดยเลือกใช้เฉพาะบางคำเท่านั้น จากนั้นได้พัฒนาขั้นตอนการทำ Real-time Inference สำหรับควบคุมหุ่นยนต์ เมื่อพัฒนาได้แล้วจึงเริ่มลงมือพัฒนาโดยใช้ชุดข้อมูลคำสั่งภาษาไทย

Keyword:

Train - ขั้นตอนการสอนให้โมเดลรู้จักและพัฒนากฎของตัวเองขึ้นจากชุดข้อมูลที่ได้รับ

Inference - ขั้นตอนการนำโมเดลที่พัฒนาแล้วไปใช้เพื่อทำนายให้ได้ผลลัพธ์ตามกฎที่โมเดลได้เรียนรู้

Chapter 2

Thai Speech Command Dataset Preparation

ในการเตรียมชุดข้อมูลเพื่อนำไป Train โมเดลนั้นเนื่องจากเราไม่สามารถหาชุดข้อมูลที่มีการเผยแพร่บนอินเทอร์เน็ตได้ จึงจำเป็นต้องสร้างชุดข้อมูลขึ้นมาใหม่ โดยชุดข้อมูลภาษาไทยเบื้องต้นที่ได้จัดทำขึ้นมาจะประกอบไปด้วยคำว่า **เดินหน้า, ถอยหลัง, เลี้ยวซ้าย, เลี้ยวขวา, จับ, ปลอ่ย, ค้นหา, และ หยุด** รวมทั้งสิ้น 8 คำโดยมีผู้ให้เสียงทั้งสิ้น 10 คน แบ่งเป็นผู้ชาย 5 คน และผู้หญิง 5 คน ให้เสียงเฉลี่ย 15-20 เสียงต่อคำต่อคน โดยใช้โทนและน้ำเสียงที่แตกต่างกันเล็กน้อยในแต่ละคำ โดยการจัดรูปแบบให้เหมือนกับ Speech Command Dataset ของ Google


















	_background_noise_	15/2/2022 11:13	File folder
	backward	1/3/2022 17:37	File folder
	forward	1/3/2022 17:24	File folder
	grab	1/3/2022 17:36	File folder
	release	1/3/2022 17:39	File folder
	search	1/3/2022 17:32	File folder
	stop	1/3/2022 17:41	File folder
	turnleft	1/3/2022 17:30	File folder
	turnright	1/3/2022 17:26	File folder
	b1.wav	1	b1
	b2.wav	2	b2
	b3.wav	3	b3
	b4.wav	4	b4
	b5.wav	5	b5
	b6.wav	6	b6
	b7.wav	7	b7
	b8.wav	8	b8

Figure 2.1 ตัวอย่างการจัดเรียงไฟล์เดอร์และไฟล์เสียงสกุล .wav

2.1 กระบวนการอัดเสียงเพื่อทำชุดข้อมูลสำหรับการพัฒนาโมเดล

ทางผู้จัดทำได้ใช้ซอฟต์แวร์ **Audacity** ซึ่งเป็นซอฟต์แวร์ฟรีสำหรับการตัดต่อเสียงที่ใช้ได้บนระบบปฏิบัติการ Windows, Linux และ Mac โดยขั้นตอนการอัดเสียงเพื่อทำชุดข้อมูลสามารถแบ่งได้คร่าวๆทั้งหมด 3 ขั้นตอน

2.1.1 การอัดเสียงคำพูด (Voice Recording)

ก่อนจะอัดเสียงทุกครั้งตรวจสอบให้แน่ใจว่าเราได้ตั้งค่ารูปแบบเสียง อุปกรณ์เข้า-ออกของเสียงถูกต้องในการอัดครั้งนี้ โดยเฉพาะการตั้งค่าช่องเสียงให้เปลี่ยนจาก **Stereo** เป็น **Mono** และตั้งค่า **Project Rate** ให้เป็น 16000 ตามรูปแบบของชุดข้อมูล Speech Command Dataset และเพื่อความสะดวกให้เปลี่ยน Selection Mode เป็น **Start and Length of Selection** หน่วยเป็น **samples** เพื่อความสะดวกในการทำ Label เสียงในแต่ละคำพูด จากนั้นกดปุ่มอัดเสียงหรือตัว R เพื่อเริ่มอัดเสียงได้ หากต้องการหยุดให้กดปุ่มหยุดหรือ Spacebar

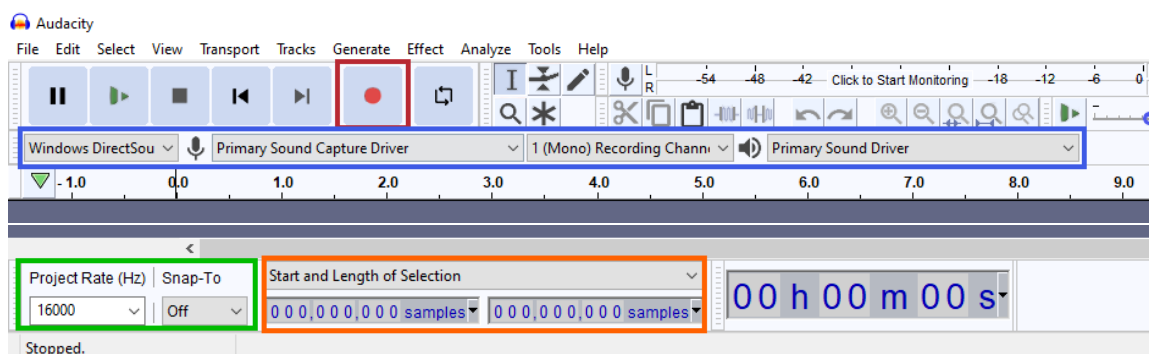


Figure 2.2 ตัวอย่างการตั้งค่าซอฟต์แวร์ Audacity

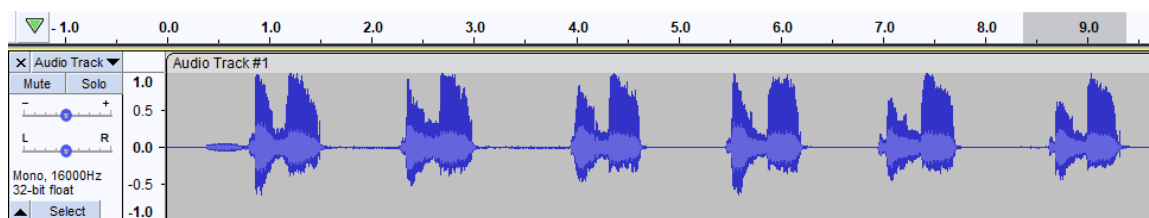


Figure 2.3 ตัวอย่างเสียงแทร็คเสียงที่ถูกอัดในโปรแกรม Audacity

2.1.2 การทำเครื่องหมายกำกับเสียงเพื่อแยกคำ (Labelling)

โดยทั่วไปแล้วเราจะทำการอัดเสียงเป็นไฟล์เสียงยาวแล้วนำมาตัดเฉพาะคำที่ต้องการภายหลัง และนำมาตัดเป็นไฟล์เสียงความยาว 1 วินาที หรือจำนวน 16,000 Samples ดังนั้นการทำ Label แต่ละคำพูดแล้วนำมา Export เป็นไฟล์เสียงสกุล .wav จึงเป็นวิธีการที่เหมาะสมและประหยัดเวลา

โดยเริ่มจากการเลือกจุดเริ่มต้นของในแทร็คเสียงจากนั้นในแถบ Selection ด้านล่าง ช่องด้านขวาที่เป็น Length of Selection ให้กำหนดจำนวน Sample เท่ากับ 16,000 จากนั้นกดไปที่ **Edit -> Labels -> Add Label at Selection** หรือคีย์ลัด **Ctrl+B** เพื่อสร้าง Label ขึ้นมา จากนั้นนำการเลือนกรอบ Label ของเรา ให้ตรงตามที่เรากำลังต้องการ ทำการตั้งชื่อ โดยชื่อที่เราตั้งจะกลายเป็นชื่อไฟล์เสียงตอน เราทำการ Export

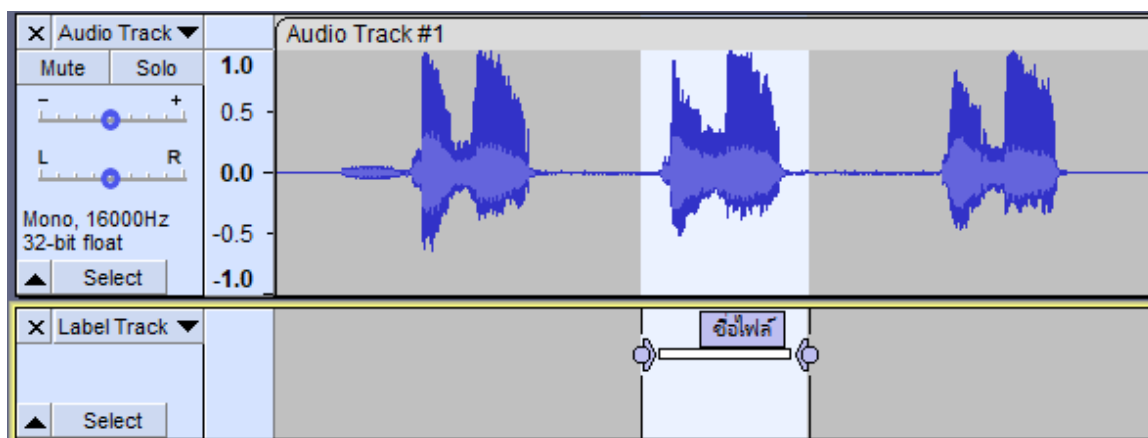


Figure 2.4 ตัวอย่างการทำ Label เสียงบน Audacity

2.1.3 การ Export ไฟล์เสียง

เมื่อทำขั้นตอนตามข้อ 2.1.2 จนได้เสียงที่เราต้องการทั้งหมดแล้ว แล้วให้เราไปที่ File -> Export -> Export Multiple.. หรือคีย์ลัด Ctrl + Shift + L จากนั้นเลือกโพล์เตอร์ที่ต้องการ และกำหนด Format ให้เป็น .wav เราจะได้ไฟล์เสียงทั้งหมดตามที่เราได้ Label เอาไว้เป็นอันเสร็จสิ้น

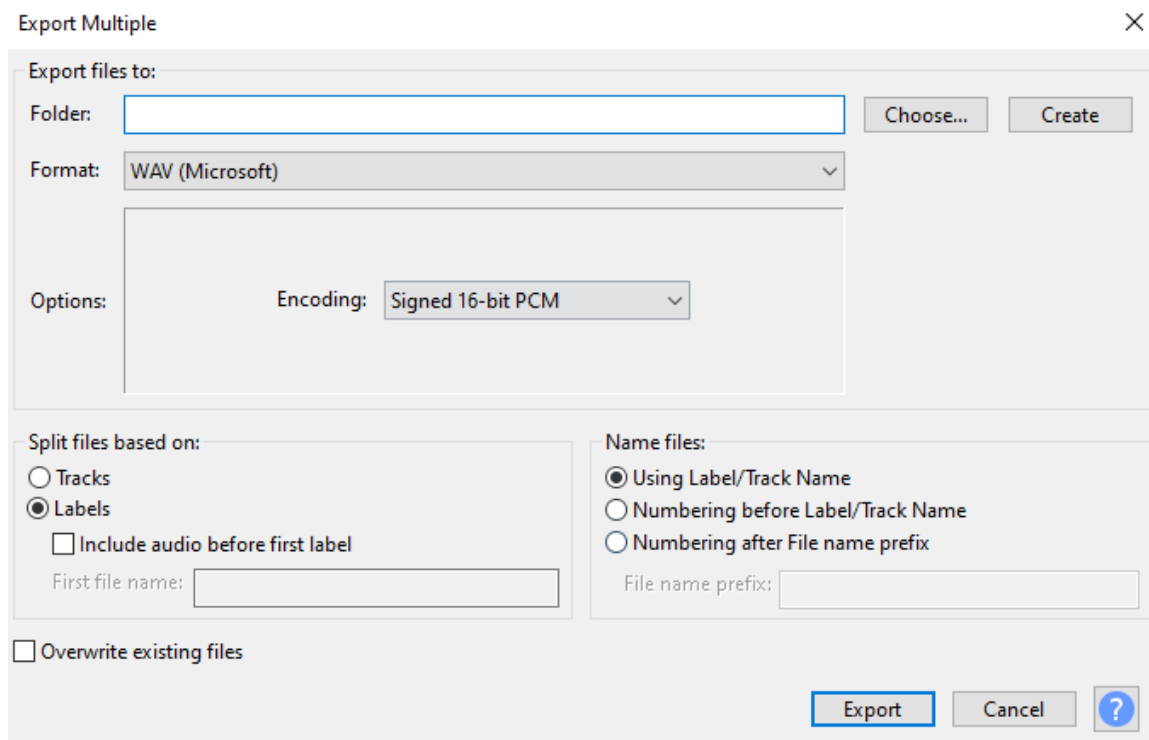


Figure 2.5 ตัวอย่างหน้า Export Multiple บน Audacity

2.2 รูปแบบการจัดโฟลเดอร์สำหรับชุดข้อมูล

ผู้จัดทำได้ทำการจัดชุดข้อมูลให้คล้ายคลึงกับตัวอย่างของชุดข้อมูล Speech Command Dataset ของ Google โดยจะแบ่งโฟลเดอร์ออกเป็น 2 โฟลเดอร์คือ **train** และ **test** ในโฟลเดอร์ train จะประกอบไปด้วยโฟลเดอร์ย่อยของแต่ละคำที่เราต้องการและ **_background_noise_** โดยจะประกอบไปด้วยเสียงรบกวนที่เราใช้เพื่อเพิ่มความแข็งแกร่ง (Robustness) ให้โมเดล

test	22/2/2022 9:46	File folder
train	1/3/2022 18:00	File folder
_background_noise_	15/2/2022 11:13	File folder
backward	1/3/2022 17:37	File folder
forward	1/3/2022 17:24	File folder

Figure 2.6 ตัวอย่างการจัดโฟลเดอร์ชุดข้อมูลเสียง

โดยในส่วนของโฟลเดอร์ test เป็นการตรวจสอบความถูกต้องของโมเดลโดยการสังเกตของเราเอง เพราะในขั้นตอนการ Train โมเดลเราจะทำการแบ่งชุดข้อมูลที่อยู่ในโฟลเดอร์ train เอาไว้ในอัตราส่วน เช่น 70:30, 80:20 หรือ 90:10 โดยหมายถึง **train_set : validation_set** ตามลำดับ

Chapter 3

Thai Speech Command Training Process

โดยปกติแล้วเรามักจะพบว่าสถาปัตยกรรมแบบ Convolutional Neural Network นั้นมักถูกใช้กับงานทางด้าน Image Classification และ Object Detection มากกว่า ดังนั้นเราจำเป็นต้องศึกษาตั้งแต่วิธีการทำ Audio Pre-processing เพื่อให้เข้าใจในตัวข้อมูลมากขึ้น ไปจนถึงขั้นตอนการออกแบบและ Train โมเดล โดยขั้นตอนทั้งหมดจะถูกเขียนโดยภาษา Python โดยมี Librosa, Tensorflow เป็น Framework หลักในการพัฒนาโมเดลของเรา โดยภาพรวมของกระบวนการ Train โมเดลแบ่งเป็นดังนี้

1. Audio Pre-processing
2. Model Definition
3. Model fitting (Training)
4. Model Inference and Data Post-processing

3.1 Audio Pre-Processing

ในการพัฒนาโมเดลเราต้องการให้โมเดลของเราเรียนรู้จากชุดข้อมูลที่เรามอบให้ แต่ในกรณีของชุดข้อมูลเสียงนั้นมีข้อมูลที่แตกต่างกันเยอะมาก และยากที่จะหารูปแบบของเสียงได้ เพื่อหลีกเลี่ยงการจัดการกับข้อมูลคลื่นเสียงดิบที่ยากต่อการหารูปแบบ เราจำเป็นต้องใช้เทคนิคบางอย่างเพื่อค้นหาลักษณะเฉพาะของเสียง เช่น เปลี่ยนการแสดงคลื่นเสียงดิบให้แสดงในรูปของสเปกตรัม เพื่อนำมาคำนวณทางดิจิทัลได้ ตัวอย่างเช่น การแปลงฟูเรียร์แบบเร็ว Fast Fourier Transform (FFT)

ข้อมูลจะถูกแบ่งออกเป็นส่วนเล็กๆก่อน ซึ่งทับซ้อนกันนำมาผ่านกระบวนการแปลงฟูเรียร์แบบเร็วจากนั้นจึงถูกนำมาต่อกันซึ่งเราเรียกผลลัพธ์นี้ว่า **สเปกโตรแกรม (Spectrogram)** และในการทำ Speech recognition ส่วนใหญ่จะนำข้อมูลสเปกตรัมที่ได้มาคำนวณหา **Mel-Frequency Cepstral Coefficients (MFCCs)**

อีกที เนื่องจากย่านความถี่คลื่นเสียงของมนุษย์นั้นค่อนข้างมีระยะที่ใกล้ชิดกัน จึงเป็นเรื่องยากที่โมเดลของเราจะเรียนรู้ลักษณะจำเพาะของเสียงแต่ละไฟล์โดยใช้ข้อมูลคลื่นที่ใกล้เคียงกันมากได้ เราจึงต้องใช้ **MFCCs** เพื่อทำให้เห็นความแตกต่างของย่านความถี่ของเสียงมนุษย์ได้ดียิ่งขึ้น สามารถอ่านไฟล์ Notebook เพื่อดูขั้นตอนการทำ Audio-Preprocessing โดยใช้ภาษา Python - แพคเกจ **librosa** ได้เพิ่มเติม

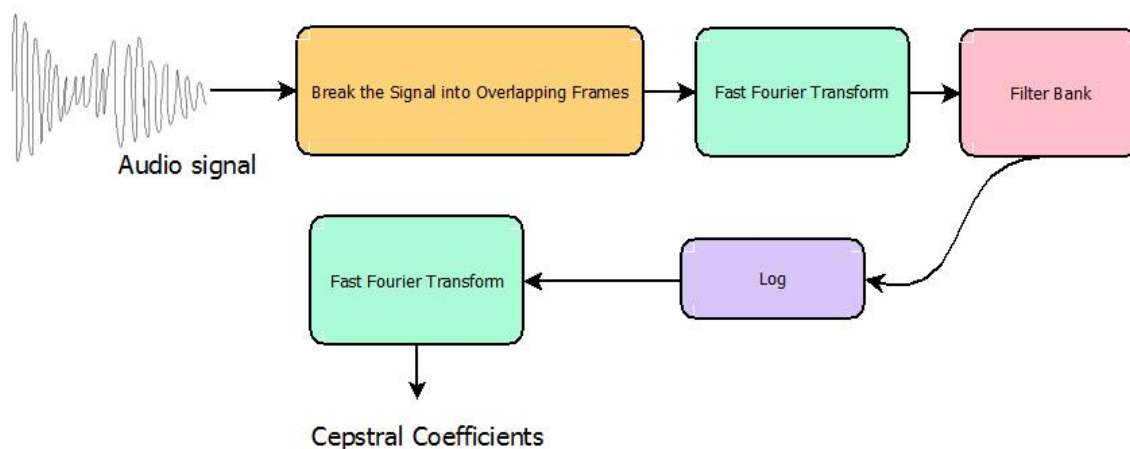


Figure 3.1 ภาพรวมการแปลงคลื่นเสียงดิบ -> MFCCs

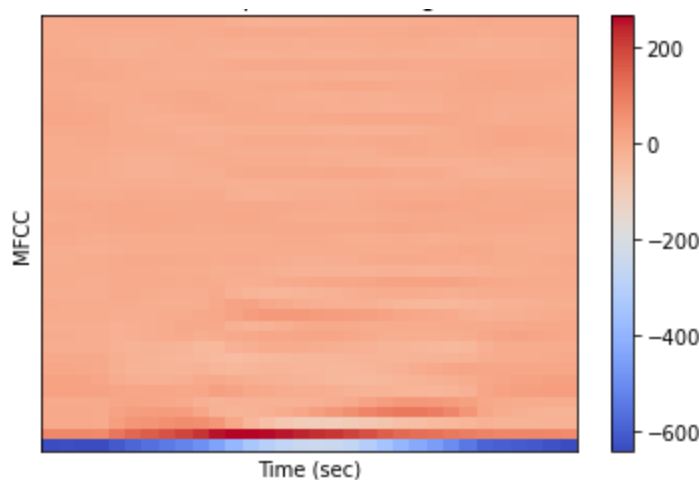


Figure 3.2 ตัวอย่าง MFCCs โดยใช้แพคเกจ librosa ใน Python

3.1.1 การทำ Audio Augmentation

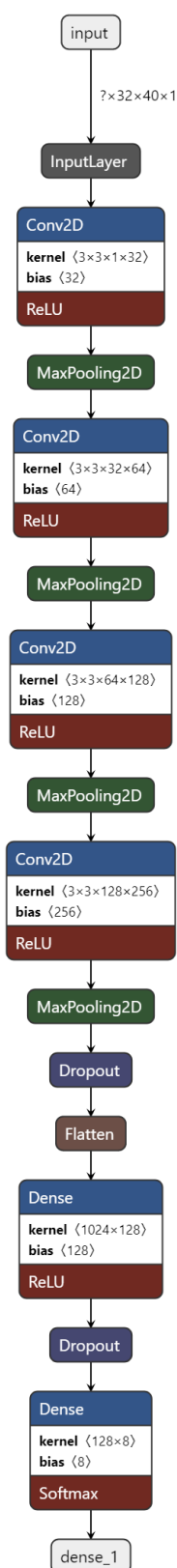
เนื่องจากมนุษย์มีโทนเสียงที่จำกัดอีกทั้งในการพูด 1 คำในหนึ่งวินาที อาจเกิดขึ้นในช่วงใดช่วงหนึ่งในวินาทีนั้นๆ ได้จึงจำเป็นต้องทำ Audio Augmentation เพื่อให้เกิดความหลากหลายของชุดข้อมูลเสียงให้มากขึ้น เช่น การเพิ่มเสียง Noise ให้กับไฟล์เสียง, การเคลื่อนตำแหน่งเสียงพูดที่เกิดขึ้นใน 1 วินาที, การเปลี่ยนระดับเสียง (คลื่นความถี่) ให้มีระดับเสียงสูงต่ำสลับกันไป, และ การเพิ่ม - ลด ความเร็วของเสียง โดยกระบวนการดังกล่าวจะเกิดขึ้นขณะการโหลดชุดข้อมูลโดยจะมีการกำหนดอัตราส่วนของการสุ่มทำ Augment ไว้ สามารถดูโค้ดเพิ่มเติมได้ในไฟล์ Notebook

3.1.2 รูปแบบของชุดข้อมูลที่พร้อมใช้งาน

เมื่อทำการโหลดชุดข้อมูล และผ่านกระบวนการ Audio Augmentation แล้ว ไฟล์ข้อมูลจะถูกจัดให้เป็นขนาด (BATCH_SIZE x ROW x COLUMN x CHANNEL) โดย CHANNEL นั้นเปรียบเหมือนกับช่องสีของรูปภาพ ในกรณีนี้เรากำหนดให้ CHANNEL = 1 เราเปรียบให้ Magnitude ของ MFCCs คล้ายคลึงกับค่าสี 0-255 ในสเกลรูปภาพ และ 1 CHANNEL นั้นคล้ายคลึงกับรูปภาพแบบ Grayscale จากนั้น ROW x COLUMNS นั้นขึ้นอยู่กับข้อกำหนดจำนวน Sample Rate ของไฟล์เสียง, ขนาดของช่วงที่ถูกนำมาแปลงฟูเรียร์แบบเร็ว, และ ระยะห่างระหว่างช่วงนั้นๆ โดยในตัวอย่างงานจะใช้ ขนาดของช่วงเป็น 4096 Samples และกำหนดระยะห่างที่ละ 512 Samples ซึ่งจะทำให้ได้ไฟล์ขนาด 40 x 32 หรือ 32 x 40 ภายหลังการทำทรานโพสเมทริกซ์ของ MFCCs

โดยสรุปเมื่อเสร็จสิ้นกระบวนการเตรียมชุดข้อมูลเราใน 1 ไฟล์เสียงเราจะได้ เมทริกซ์ หรือ Numpy Array ขนาด (1 x 32 x 40 x 1)

3.2 Model Definition



รูปแบบสถาปัตยกรรม Convolutional Neural Network ที่ได้นำมาใช้ในการพัฒนาโมเดลในครั้งนี้มีต้นแบบมาจากบล็อก **Simple Audio Classification with Keras** ที่ถูกตีพิมพ์โดย Daniel Falbel เมื่อ 6 มิถุนายน 2018 ในเว็บไซต์ AI Blog ดูเพิ่มเติมได้ที่ [หัวข้ออ้างอิง](#)

โดยผู้จัดทำได้นำรูปแบบสถาปัตยกรรมมาใช้และมีการดัดแปลงเพิ่มเติมโดยมีเลเยอร์

Conv2D หมายถึง Convolutional Neural Network 2 มิติ ซึ่งเป็นการใช้ Activation Function เป็น **Rectified Linear Unit (ReLU)** และยังมีการเพิ่ม `kernel_regularizer l2` พารามิเตอร์ เพื่อลดปัญหา Overfitting ของโมเดลที่อาจเกิดขึ้นได้

MaxPooling2D หมายถึง Max Pooling layer 2 มิติซึ่งเป็นการย่อขนาดข้อมูลให้เล็ก สกัดเอาส่วนที่สำคัญของข้อมูลนั้นๆ เพื่อเพิ่มประสิทธิภาพการประมวลผลให้เร็วขึ้น

Dropout ซึ่งเป็นวิธีการลดการเกิด Overfitting ของโมเดล

Dense หมายถึง Fully Connected Layer ทุกๆ Node ใน Neuron เชื่อมข้อมูลจากทุก Input ที่มาจาก Layer ก่อนหน้าซึ่งเลเยอร์นี้ก็มีการใช้ Activation Function เป็น ReLU

สุดท้ายเป็น Dense layer ของ Output ที่ยังคงเป็น Fully Connected Layer แต่ Activation Function นั้นเป็น Softmax ที่ใช้สำหรับการทำนาย Probability ของชุดข้อมูลที่เป็นผลลัพธ์แบบหมวดหมู่ (Categorical Result)

Figure 3.3 โครงสร้างโมเดล Thai Speech Command

3.3 Model Fitting

เมื่อทำการกำหนดสถาปัตยกรรมของโมเดลที่เราต้องการแล้ว ขั้นตอนต่อไปคือวิธีการกำหนดสัดส่วนของข้อมูล และไฮเปอร์พารามิเตอร์ที่จะต้องเป็นตัวกำหนดว่าเราจะให้โมเดล Train อย่างไร

3.3.1 การแบ่งสัดส่วนข้อมูล (Train – Validation Split)

หลังจากผ่านกระบวนการเตรียมชุดข้อมูล (Data Pre-processing) แล้วเราจำเป็นต้องแบ่งชุดข้อมูลออกเป็น 2 ส่วน ส่วนหนึ่งถูกใช้เพื่อการ Train โมเดล อีกส่วนหนึ่งจะถูกใช้ในการตรวจสอบประสิทธิภาพของโมเดลในแต่ละ Epochs ในอัตราส่วน 70:30 โดยข้อมูลจะถูกแยกข้อมูลเสียงและข้อมูล Label ของเสียงนั้นๆ เอาไว้ พร้อมกับสลับลำดับของเสียงเพื่อป้องกันการ Overfitting ที่อาจเกิดขึ้น เมื่อเสร็จสิ้นจึงนำข้อมูลที่กำลังจะถูกป้อนเข้าโมเดลรวมกันขึ้นเป็น Batch เพื่อลดระยะเวลาการ Train ของโมเดลให้สั้นลง สามารถอ่านไฟล์ Notebook เพื่อดูขั้นตอนการแบ่งสัดส่วนข้อมูลและการนำชุดข้อมูลรวมกันเป็น Batch ได้เพิ่มเติม

3.3.2 การกำหนด Hyperparameter

A) INPUT SHAPE เป็นการกำหนดขนาด Shape ของข้อมูลที่จะนำเข้าไปในโมเดลของเราซึ่งการทำ Data-Preprocessing จะเป็นตัวกำหนด ในที่นี้คือ $(32 \times 40, 1)$ หากตรวจสอบในโมเดลอาจพบว่าเป็น (None, $32 \times 40 \times 1$) ซึ่ง None ใช้แทนขนาด Batch

B) BATCH_SIZE เป็นการกำหนดว่าใน 1 ครั้งที่เรานำชุดข้อมูลเข้าสู่โมเดลเพื่อ Train เราจะใช้กี่ชุดข้อมูลเสียง ในตัวอย่างการ Train ได้กำหนด **BATCH_SIZE** ให้เท่ากับ 32 ดังนั้นจะพบว่า INPUT SHAPE จริงๆของเราคือ $(32 \times 32 \times 40 \times 1)$

C) EPOCHS เป็นการกำหนดจำนวนครั้งที่จะ Train โมเดลของเรา โดยปกติแล้ว Neural Network จะทำการคำนวณ Weight โดยใช้ Backpropagation โดยจะนับเป็น Step ไปซึ่งการทำเพียงแค่ครั้งเดียวอาจไม่เพียงพอต่อการเรียนรู้ของ

โมเดลจึงต้องมีการกำหนด Epochs ให้เกิดการเรียนรู้ซ้ำ การกำหนดจำนวน Epoch นั้นไม่ตายตัว และไม่มีตัวเลขที่ถือว่าดีที่สุด ขึ้นอยู่กับการทดลองนั้นๆ หากกำหนดจำนวน Epochs เยอะไปจะทำให้เสียเวลาในการ Train โมเดลนั้นๆและทำให้โมเดลเกิด Overfitting หากใช้ Epoch น้อยไปจะทำให้โมเดลนั้นเกิดการ Underfitting หรือสถานะที่โมเดลไม่สามารถเรียนรู้อะไร โดยตัวอย่างนั้นจะกำหนดไว้ที่ **50-100 Epochs** โดยจะมี Early Stopping ซึ่งเป็น Callback Function มาใช้ภายหลัง

D) LEARNING_RATE เป็นการกำหนดว่าในแต่ละ Step ของการทำ Gradient descent (Weight Adjustment) จะต้องปรับทีละมากเท่าไรเป็นอีกหนึ่งสิ่งที่ต้องทำการทดลองปรับให้เหมาะสม หาก Learning rate น้อยเกินไปจะทำให้เสียเวลาในการ Train โมเดลเพิ่มมากขึ้น หากปรับมากเกินไปอาจทำให้โมเดลหาจุดต่ำสุดของค่า Loss ไม่เจอทำให้โมเดลเสียประสิทธิภาพไป ในตัวอย่างจะกำหนด Learning Rate ไว้ที่ **0.001**

E) Optimizer จะเป็นตัวกำหนดในขณะทำการ Compile โมเดลว่าจะให้ดำเนินการคำนวณ Gradient descent อย่างไรในตัวอย่างได้ใช้ตัว optimizer แบบ **Adaptive Moment Estimation (ADAM)** ซึ่งสามารถปรับ Learning rate ให้เราได้อีกนำมาใช้อย่างแพร่หลายในการแก้ปัญหา decaying ของ gradients

F) Loss Function ในตัวอย่างจะใช้ **SparseCategoricalCrossentropy** ซึ่งจะทำการคำนวณค่า Loss ที่เกิดขึ้นจากการทำนายของโมเดลที่มีผลลัพธ์เป็นแบบ Categorical และมีผลลัพธ์มากกว่า 1 คลาส

```
ROWS = 32
COLUMNS = 40
BATCH_SIZE = 32
EPOCHS = 100
LEARNING_RATE = 0.001
optimizer = keras.optimizers.Adam(learning_rate = LEARNING_RATE)
loss_fn = keras.losses.SparseCategoricalCrossentropy()
acc_metric = keras.metrics.SparseCategoricalAccuracy()
```

Figure 3.4 ตัวอย่างโค้ดการกำหนด Hyperparameter

3.3.3 การกำหนด Callback Function

การกำหนด Callback Function นั้นจำเป็นต่อการ Train โมเดลในหลายๆ ด้าน ในตัวอย่างก็มีการใช้ Callback Function เช่น การสร้าง Checkpoint ของโมเดล, การลด Learning Rate ของโมเดล, รวมไปถึงการสร้างเงื่อนไข Early stop เมื่อโมเดลไม่มีการเพิ่มประสิทธิภาพในช่วง Epoch ที่กำหนดแล้ว ไม่เพียงเท่านั้น Callback Function ยังถูกใช้ในการสร้าง Log เพื่อให้เราสามารถสังเกตประสิทธิภาพของโมเดลได้อีกด้วย

- A) **Checkpoint Callback** เป็นการบันทึกโมเดล ณ Epoch นั้นๆ โดยทั่วไปแล้วหากไม่มี Checkpoint Callback เราจะสามารถบันทึกไฟล์โมเดลได้เฉพาะ Epoch สุดท้ายเท่านั้น ในตัวอย่างจะเป็นการบันทึก Checkpoint ของแต่ละ Epoch ในรูปแบบสกุลไฟล์ .h5 โดยจะทำการบันทึกเฉพาะ Epoch ที่มีค่า Accuracy มากกว่าเดิมเท่านั้น
- B) **EarlyStop Callback** เป็นการบอกให้โมเดลสังเกตประสิทธิภาพของตนเองว่าเพิ่มขึ้นหรือไม่ในแต่ละ Epoch หากโมเดลมีประสิทธิภาพเท่าเดิมเมื่อผ่านไปหลายๆ Epoch นั้นหมายความว่า การ Train ต่อไปจะเป็นการเสียเวลา การมี EarlyStop จะช่วยให้เราประหยัดเวลาในการ Train ได้มากขึ้น
- C) **Log Callback** เป็น Callback ที่ช่วยให้เราสามารถสังเกตประสิทธิภาพของโมเดลได้ ตัวอย่างเช่นการใช้ Tensorboard หรือ Wandb Callback ทำให้การวิเคราะห์ประสิทธิภาพของโมเดลนั้นง่ายขึ้น โดยอาจแสดงผลในรูปแบบกราฟ และข้อมูลอื่นๆ เป็นต้น ในตัวอย่างไฟล์ Notebook ได้ใช้ Wandb ซึ่งเป็น Log Callback ที่มีความสามารถในการส่งข้อมูลขึ้นไปยังเซิร์ฟเวอร์และแสดงผลบน wandb.ai ทำให้เราสามารถตรวจสอบประสิทธิภาพของโมเดลที่ได้ก็ได้ ซึ่งทำให้มีข้อได้เปรียบมากกว่า Tensorboard เนื่องจากจะจำกัดอยู่เฉพาะบน localhost เท่านั้น

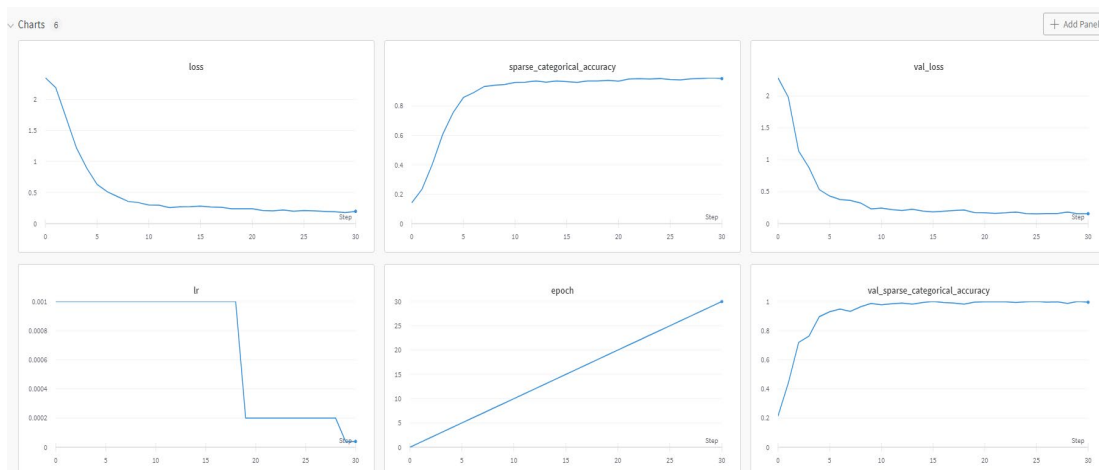


Figure 3.5 ตัวอย่างผลลัพธ์โมเดลที่ได้จากการเก็บ Log ด้วยใช้ wandb

3.4 Model Inference and Data Post-processing

การวัดประสิทธิภาพโมเดลนั้นมีวิธีการที่หลากหลายเราอาจนำไฟล์เสียงที่แยกออกจากชุดข้อมูลสำหรับการ Train มาเพื่อทดสอบว่าโมเดลของเราสามารถทำนายได้อย่างถูกต้องหรือไม่ โดยอาจเขียนคลาสเพื่อนำไฟล์เสียงนั้นมาผ่านกระบวนการ Data-Preprocessing จัดรูปแบบ ให้ตรงตาม Input Shape โมเดลของเรา และให้โมเดลของเราทำนายผลลัพธ์ โดยผลลัพธ์ที่ได้มานั้นจะเป็นความน่าจะเป็นของแต่ละคลาส หากเราต้องการทราบว่าโมเดลของเราทำนายมาเป็นคลาสไหนให้เราเลือกตำแหน่งของผลลัพธ์ที่มีค่าความน่าจะเป็นมากที่สุด แล้วนำมาเทียบกับ Label ว่า เป็นคลาสอะไร สามารถเปิดไฟล์ Notebook เพื่อดูโค้ดส่วนนี้เพิ่มเติมได้ ไฟล์ Label จะถูกเก็บในรูปแบบไฟล์ json หากใช้กระบวนการ Data-preprocessing เดียวกับตัวอย่างที่ผู้จัดทำให้ทำขึ้น

ในบทต่อไปจะนำเสนอ Workflow ของการทำ Real-time Inference และวิธีการใช้ซึ่งจะมีวิธีการที่ซับซ้อนกว่าการทำ Inference แบบไฟล์เดียว หรือเป็นชุด


```
{  
  "ถอยหลัง": 0,  
  "เดินหน้า": 1,  
  "จับ": 2,  
  "ปล่อย": 3,  
  "ค้นหา": 4,  
  "หยุด": 5,  
  "เลี้ยวซ้าย": 6,  
  "เลี้ยวขวา": 7  
}
```

Figure 3.6 ตัวอย่างไฟล์ classlabel.json แปลไทยที่นำมาใช้กับการ Inference

Chapter 4

Thai Speech Command Real-time Inference

ในบทนี้จะนำเสนอ Workflow การทำงานของ Real-time Inference และวิธีการใช้งานไฟล์ .py ที่เป็นไฟล์พร้อมใช้งานสำหรับการทำ Real-time Inference

1. รับข้อมูลเสียงพูด (อัดเสียง) โดยใช้ Pyaudio เป็นเวลา 1 วินาที หรือตามระยะเวลาที่ผู้ใช้ต้องการ Sample Rate 16000Hz / CHUCK 4096 และเสียง Mono แล้วนำมาบันทึกเป็นไฟล์เสียงชั่วคราวในรูปแบบ .wav โดยการพูดคำสั่งเสียง 1-2 พยางค์นั้นมักใช้ไม่เกิน 1 วินาที
2. นำเข้าไฟล์เสียงและทำ Data-Preprocessing จัดเรียง INPUT SHAPE ให้ตรงกับขนาดของโมเดล หากระยะเวลาของไฟล์เสียงนั้นเกิน 1 วินาทีให้แบ่งเข้าครั้งละ 1 วินาที หรือ 16000 Samples
3. ป้อนข้อมูลเข้าสู่โมเดลเพื่อให้โมเดลทำนายผลลัพธ์ โมเดลจะทำนายความน่าจะเป็น ซึ่งตัวที่มีความน่าจะเป็นสูงสุดในบรรดาคำทั้งหมดจะถูกเลือกให้เป็นผลลัพธ์ของการทำนายนั้น หากไฟล์เสียงเกิน 1 วินาที ให้นำแต่ละเสียงมาเปรียบเทียบกับ คำที่มีความน่าจะเป็นสูงสุดจะเป็นผลลัพธ์ของการทำนายในครั้งนั้น
4. ดำเนินการแสดงผลลัพธ์ หรือดำเนินการอื่นๆหากมีการกำหนดไว้ จากนั้นเริ่มกระบวนการในข้อ 1 ใหม่อีกครั้ง โดยระยะเวลาขึ้นอยู่กับที่ตั้ง Delay เอาไว้

4.1 วิธีใช้งานไฟล์ SpeechControlTH_RT.py เพื่อเริ่ม Real-time Inference

เราสามารถเรียกใช้งานไฟล์และระบุ Parameter ที่จำเป็นต่อการทำ Real-time Inference ได้ผ่าน Command Prompt หรือ Terminal ของเราได้โดย Parameter ที่จำเป็นต้องระบุสามารถดูได้ผ่าน

```
$ python SpeechControlTH_RT.py --help
```

- A) **-m, --model** เป็นการระบุตำแหน่ง PATH ของโมเดล .h5
- B) **-l, --label** เป็นการระบุตำแหน่ง PATH ของไฟล์ Label .json เพื่อทำการจับคู่ label กับผลลัพธ์ที่โมเดลทำนาย
- C) **-c, --conf** เป็นการระบุค่าผลลัพธ์ที่โมเดลทำนายออกมาจะต้องมีความน่าจะเป็นมากเท่าไรถึงจะถือว่าผลลัพธ์นั้นเป็นที่ยอมรับ ความน่าจะเป็นของผลลัพธ์นั้นน้อยกว่าค่า Confidence ที่ตั้งไว้ ผลลัพธ์จะทำการแสดงเป็น “Others” หรือ อื่นๆ แทน
- D) **-dl, --delay** คือการกำหนดวินาทีในการตีเลย์ การ Inference 1 ครั้ง
- E) **-d, --duration** คือระยะเวลาของการอัดเสียงพูดซึ่งโดยปกติแล้วเราจะใช้เวลา 1 วินาที

```
REAL-TIME KEY-WORD RECOGNITION SPEECH INFERENCE

optional arguments:
  -h, --help            show this help message and exit
  -m MODEL, --model MODEL
                        PATH TO MODEL FILE .h5
  -l LABEL, --label LABEL
                        PATH TO LABEL FILE .json
  -c CONF, --conf CONF  SET CONFIDENCE THRESHOLD OF KEYWORD (>0.9)
  -dl DELAY, --delay DELAY
                        SET DELAY BETWEEN EACH INFERENCE (sec)
  -d DURATION, --duration DURATION
                        SET DURATION OF RECORDING SOUND (1 sec)
```

Figure 4.1 รูปภาพแสดง Parameter ที่จำเป็นต้องระบุก่อนเรียกใช้งานไฟล์

ตัวอย่างโค้ดในการใช้งานจริง (Windows)

โปรตรัมภ์ระวังในการกำหนด PATH ของโมเดลและ Label การกำหนด PATH บน Windows Terminal นั้นแตกต่างจากการกำหนด PATH บน Linux Terminal

```
python SpeechControlTH_RT.py --model models\model.h5 --label
Data_Thai\classmap.json --conf 0.9 --delay 3 --duration 1
```

```
#####
#: START RECORDING FOR 1 SEC.. | PLEASE SPEAK KEYWORD NOW!!
#####
#: STOP RECORDING, SAVING CACHE FILE..
R: SUMMARIZE KEYWORDS DETECTED 'เ ดื น ห นั ้า' with Confidence 100.00%

INFERENCE USING 0.03789830207824707

#####
#: START RECORDING FOR 1 SEC.. | PLEASE SPEAK KEYWORD NOW!!
#####
#: STOP RECORDING, SAVING CACHE FILE..
R: SUMMARIZE KEYWORDS DETECTED 'ถอยห ลั ง' with Confidence 99.80%

INFERENCE USING 0.03989362716674805
```

Figure 4.2 ตัวอย่างผลลัพธ์ที่ได้จากการรันไฟล์ Inference

Chapter 5

Discussions & Suggestions

5.1 การเก็บข้อมูลเพื่อเพิ่มประสิทธิภาพโมเดล

เนื่องจากมีข้อจำกัดในการเก็บชุดข้อมูลเสียง จึงทำให้ชุดข้อมูลเสียงนั้นไม่เพียงพอที่จะสร้างความแข็งแกร่ง (Robustness) ให้กับโมเดล กล่าวคือโมเดลนั้นสามารถจดจำเสียงและคำพูดได้ค่อนข้างแม่นยำก็จริง แต่เมื่อทดสอบกับเสียงของผู้ใช้งานที่ไม่ได้อยู่ในชุดข้อมูลสำหรับ Train พบว่าประสิทธิภาพนั้นแย่งอย่างเห็นได้ชัด อีกปัญหาที่พบคือโมเดลยังคงมีความอ่อนแอต่อเสียงรบกวน เมื่อมีเสียงรบกวนจากไมค์โครโฟน อาจทำให้เกิดความผิดพลาดในการทำนายขึ้นได้ ยกตัวอย่างเช่น เมื่อเราไม่พูดแต่มีเสียงรบกวนรอบข้าง ในบางครั้งโมเดลจะทำนายเป็นคำสั่งคำหนึ่งโดยมีค่าความเป็นไปได้ต่ำ หรืออาจเป็นสูงในบางครั้ง ดังนั้นคุณภาพของไมค์โครโฟนยังคงมีผลกับการทำนายของโมเดลอยู่ โดยปัญหาดังกล่าวอาจแก้โดยการเพิ่ม Class “อื่นๆ” เข้าไปในโมเดล

เมื่อเทียบกับ Speech Command Dataset ของ Google จะพบว่าในชุดข้อมูลของ Google นั้นมีทั้งสิ้น 65,000 ไฟล์ จำนวน 30 คำ เฉลี่ยคำละ 2,100 ไฟล์ ซึ่งมากกว่าชุดข้อมูลภาษาไทยที่ผู้จัดทำได้เก็บไว้ถึง 20 เท่า โดยชุดข้อมูลภาษาไทยมีประมาณ 1,300 ไฟล์ จำนวน 8 คำ เฉลี่ยคำละ 170 ไฟล์ จึงเป็นสาเหตุหลักที่ทำให้โมเดลยังขาดความแข็งแกร่ง หากมีการต่อยอดโดยใช้ตัวอย่าง File Notebook ในการ Train โมเดลเพื่อเพิ่มประสิทธิภาพ การเพิ่มเสียงพากย์คำโดยใช้หลากหลายคนมากขึ้น ต่างเพศ ต่างวัย โดยไม่ผ่านวิธีการทำ Data Augmentation จะสามารถเพิ่มประสิทธิภาพโมเดลให้จดจำเสียงกลุ่มบุคคลที่ไม่ได้อยู่ในชุดข้อมูล Train ได้ดียิ่งขึ้น

5.2 ปัญหาที่อาจเกิดขึ้นในการใช้งาน Real-time Inference

ปัญหาหลักที่พบซึ่งทำให้การจดจำเสียงพูดไม่แม่นยำคือ ความคลาดเคลื่อนของเสียงพูดขณะอัด เนื่องจากเราจะต้องพูดคำสั่งให้ทันภายใน 1 วินาที โดยมักจะพบปัญหานี้ในคำสั่งที่มีมากกว่า 1 พยางค์ เช่น เลี้ยวซ้าย, เลี้ยวขวา, เดินหน้า, ถอยหลัง, และ ค้นหา หากมีการเริ่มอัดแล้วแต่ผู้พูดหน่วงเวลาในการพูด จะทำให้เสียงที่อัดมีความคลาดเคลื่อนได้ ทำให้คำสั่งไม่ครบ จึงอาจส่งผลให้การทำนายของโมเดลเกิดความไม่แม่นยำขึ้น

การเพิ่มระยะเวลาในการอัดเสียงเป็นทางเลือกหนึ่งที่สามารถทำได้ แต่อาจจะต้องแลกกับประสิทธิภาพโมเดลที่ลดลง เนื่องจากกระบวนการทำ Real-time Inference เมื่อมีไฟล์เสียงที่มีความยาวมากกว่า 1 วินาที จะต้องอ่านไฟล์เสียงออกเป็นครั้งละ 1 วินาทีหรือ 16000 Samples ตามที่ได้ Train มา เศษที่เหลือของวินาทีนั้นจะถูกทำการ Padding ด้วยค่า 0 การทำแบบนี้อาจช่วยลดปัญหาการพูดไม่ทันได้บางกรณี แต่การทดสอบพบว่าเมื่อมีการพูดคำสั่งคร่อมระหว่างวินาทีนั้น จะทำให้โมเดลมีประสิทธิภาพในด้านความแม่นยำการทำนายด้อยลงไปอีก และการอ่านไฟล์เสียง 1 วินาทีหรือที่ละ 16000 Sample แบบใช้ระยะการเลื่อน Sample น้อยกว่า 16000 ตัวอย่างเช่นเลื่อนที่ละ 8000 Samples นั้นอาจลดปัญหาการพูดคร่อมได้ก็จริง แต่จะทำให้ประสิทธิภาพด้านความเร็วของโมเดลนั้นลดลง เพราะต้องรับมือกับข้อมูลในถูกป้อนเข้าโมเดลมากขึ้น ในกรณีที่เลื่อนครั้งละ 8000 Samples จะทำให้ต้องป้อนข้อมูลเข้าโมเดลมากขึ้น 2 เท่า เมื่อเทียบกับการเลื่อนที่ละ 16000 Samples ในการดำเนินการ 1 ครั้ง จากนั้นโมเดลจะทำการทำนายว่าช่วงไหนของไฟล์เสียงเป็นคำพูดใดและมั่นใจมากเท่าใด จากนั้นจึงเลือกนำเสนอผลลัพธ์ที่มีความน่าจะเป็นสูงที่สุด

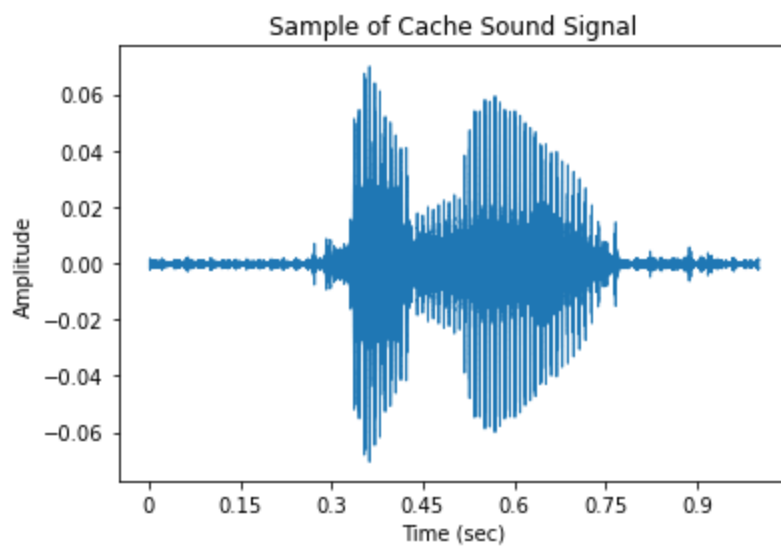


Figure 5.1 ตัวอย่างสัญญาณเสียงคำว่า “เดินหน้า” ใน 1 วินาทีเมื่อพูดปกติ

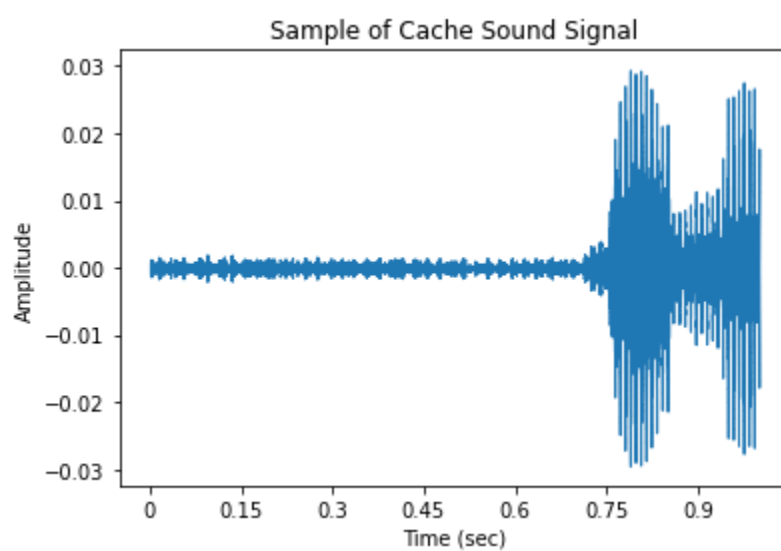


Figure 5.2 ตัวอย่างสัญญาณเสียงคำว่า “เดินหน้า” ใน 1 วินาทีเมื่อพูดช้า

5.3 แนะนำแนวทางในการนำไปใช้งาน

ในตัวไฟล์ Real-time Inference (SpeechControlTH_RT.py) นั้นจะมีฟังก์ชันที่เราเรียกใช้งาน จะมีการส่งกลับค่าผลลัพธ์ของโมเดลเป็นลิสต์ที่ถูกทำนาย และค่าความน่าจะเป็น เราสามารถนำค่าเหล่านั้นมาดำเนินการทางตรรกะ สร้างเงื่อนไขเพื่อควบคุมหุ่นยนต์ของเราได้ โดยอาจใช้งานร่วมกับ MQTT Protocol ซึ่ง Python มีแพ็คเกจรองรับเช่น Paho-MQTT เพื่อใช้งานตัว Real-time Inference บนเครื่องพีซีที่มีการ์ดจอ และส่งข้อมูลผลลัพธ์การทำนายคำสั่งของโมเดลผ่านตัว MQTT Protocol ผ่านสัญญาณ Wifi ไปยัง Microcontroller ที่สามารถเชื่อมต่อสัญญาณ Wifi และรองรับการใช้งาน MQTT Protocol package ได้เช่น Raspberry Pi และ ESP32 เพื่อสร้างเงื่อนไข ตรรกะ ในการควบคุมหุ่นยนต์อีกทีหนึ่ง วิธีการนี้จะช่วยให้ประหยัดงบในการจัดหาอุปกรณ์ที่นำมาควบคุมอุปกรณ์ เพราะอุปกรณ์ที่เป็น Edge Device ซึ่งมี I/O เอาไว้จ่ายไฟควบคุมได้นั้น เช่น NVIDIA AGX Jetson นั้นมีราคาค่อนข้างแพง เมื่อเทียบกับ Microcontroller ที่กล่าวมาข้างต้น

References

- Falbel, D. (2018, June 6). *Simple audio classification with Keras*. RStudio AI Blog. Retrieved March 2, 2022, from <https://blogs.rstudio.com/ai/posts/2018-06-06-simple-audio-classification-keras/>
- Gartzman, D. (2020, May 9). *Getting to know the Mel spectrogram*. Medium. Retrieved March 2, 2022, from <https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0>
- Launching the speech commands dataset*. Google AI Blog. (2017, August 24). Retrieved March 2, 2022, from <https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html>
- Maklin, C. (2019, December 29). *Fast fourier transform*. Medium. Retrieved March 2, 2022, from <https://towardsdatascience.com/fast-fourier-transform-937926e591cb>
- Sainath, T. N., & Parada, C. (2015). Convolutional Neural Networks for small-footprint keyword spotting. *Interspeech 2015*. <https://doi.org/10.21437/interspeech.2015-352>
- Tensorflow Guide*. TensorFlow. (n.d.). Retrieved March 3, 2022, from <https://www.tensorflow.org/guide>
- Weights & Biases - documentation*. Weights & Biases - Documentation. (n.d.). Retrieved March 2, 2022, from <https://docs.wandb.ai/>