13주차 연구노트

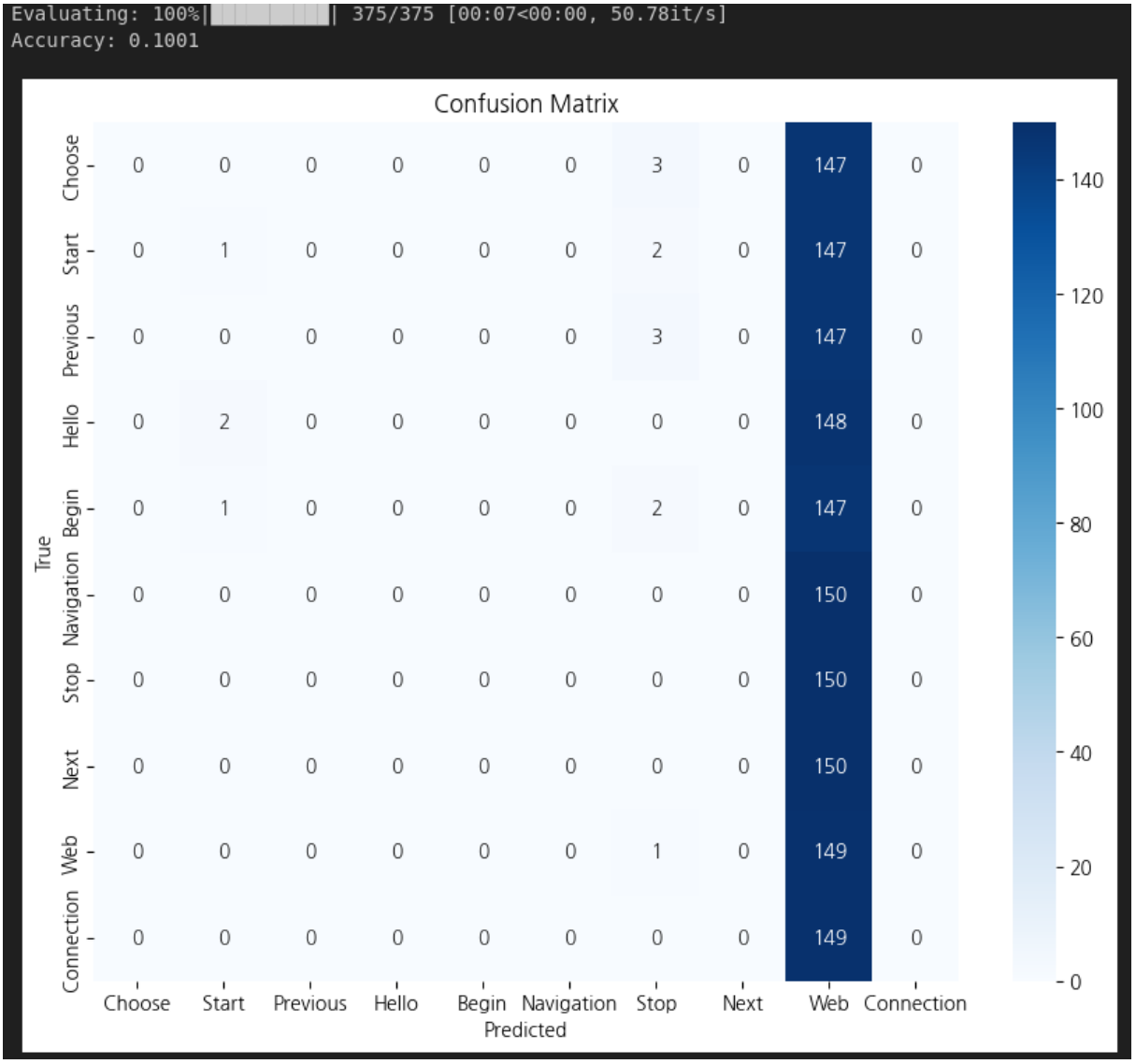
○ 13주차 계획 (12주차 피드백 반영)

- 한국어 립리딩 영상 데이터셋 전처리 → 입술 부분만 잘라서 0.1초에 한번씩 저장
- MobileNet v2 + LSTM 모델 선정 및 비교
- 구현한 트레이닝 모델 성능 확인

○ Miracle-VC 데이터셋을 통한 입술 데이터 추출 결과

Label: Hello

※ Epoch : 10, Loss : 1.84 → 0.95



⇒ 데이터셋의 화질이 좋지 못해 학습이 제대로 이루어지지 않아 해당 데이터셋 사용하지 않음

○ 동영상에서 입술 부분만 추출하는 전처리 코드

Step 1 : 라이브러리 및 모델 초기화

```python
import os
import cv2
import dlib
import json
import torch
import torchvision.transforms as transforms
from torchvision.utils import save_image
import numpy as np
from tqdm import tqdm

# dlib의 얼굴 탐지기와 랜드마크 예측기 초기화
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# Pytorch Tensor 변환 함수
transform = transforms.ToTensor()
```

Step 2 : 동영상 파일 경로 및 라벨링 데이터 경로 설정

```python
# 현재 작업 디렉토리 설정
current_dir = os.getcwd()

# 동영상 파일이 있는 디렉토리 설정
video_dir = os.path.join(current_dir, "video", "원천데이터")
label_dir = os.path.join(current_dir, "video", "라벨링데이터")

# mp4 파일 목록 가져오기
video_files = [f for f in os.listdir(video_dir) if f.endswith(".mp4")]
video_files = ["lip_K_5_M_06_C920_A_001.mp4"]
# 이미지 저장 경로 설정
output_dir = os.path.join(current_dir, "output_lips2")
os.makedirs(output_dir, exist_ok=True)

video_files
```

Step 3 : 라벨링 데이터 로드 함수

```python
def load_label_data(label_file):
    with open(label_file, 'r', encoding='utf-8') as f:
        label_data = json.load(f)
    return label_data[0]


def get_labels_for_frame(frame_number, fps, label_data):
    time = frame_number / fps
    labels = [entry['sentence_text'] for entry in label_data['Sentence_info'] if entry['start_time'] <= time <= entry['end_time']]
    return labels if labels else ["No Label"]
```

```python
        # 얼굴 탐지
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = detector(gray)

        for face in faces:
            # 랜드마크 예측
            landmarks = predictor(gray, face)
            lips_points = []
            # 입술 좌표 추출 (랜드마크 포인트 48-67)
            for n in range(48, 68):
                x = landmarks.part(n).x
                y = landmarks.part(n).y
                lips_points.append((x, y))

            # 입술 부분만 잘라내기
            x_coords = [p[0] for p in lips_points]
            y_coords = [p[1] for p in lips_points]
            x_min, x_max = min(x_coords), max(x_coords)
            y_min, y_max = min(y_coords), max(y_coords)

            lips_roi = frame[y_min:y_max, x_min:x_max]

            # 이미지 저장 경로 설정
            processed_image_path = os.path.join(sentence_text_folder+'/M1/01', f"processed_video_{os.path.splitext(video_file)[0]}_{save_count}.png")

            # 저장 경로의 디렉토리 생성
            os.makedirs(os.path.dirname(processed_image_path), exist_ok=True)

            # 이미지로 저장
            lips_tensor = transform(cv2.cvtColor(lips_roi, cv2.COLOR_BGR2RGB))
            save_image(lips_tensor, processed_image_path)

            # 라벨링 데이터 저장
            labels = get_labels_for_frame(frame_number, fps, label_data)
            label_info[processed_image_path] = labels

            save_count += 1

    cap.release()
```

Step 4 : 동영상 처리 및 입술 이미지 저장

```python
# 프레임 간격 설정 (예: 0.5초 간격)
seconds_interval = 0.1

# 라벨링 데이터를 저장할 딕셔너리 초기화
label_info = {}

# 특수 문자 제거 함수
def clean_folder_name(name):
    return "".join(c for c in name if c.isalnum() or c in (' ', '_')).rstrip()

for video_file in tqdm(video_files, desc="Processing Videos"):
    video_path = os.path.join(video_dir, video_file)
    label_file = os.path.join(label_dir, os.path.splitext(video_file)[0] + ".json")
    # 라벨링 데이터 로드
    if os.path.exists(label_file):
        label_data = load_label_data(label_file)
    else:
        print(f"Label file for {video_file} not found.")
        continue

    cap = cv2.VideoCapture(video_path)
    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_interval = int(fps * seconds_interval)  # 프레임 간격 계산

    for sentence in label_data['Sentence_info']:
        start_frame = int(sentence['start_time'] * fps)
        end_frame = int(sentence['end_time'] * fps)
        save_count = 0

        # sentence_text를 폴더 이름으로 사용하여 특수 문자를 제거하고 폴더 생성
        sentence_text_folder = os.path.join(output_dir, clean_folder_name(sentence['sentence_text']))
        os.makedirs(sentence_text_folder, exist_ok=True)

        cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)
        for frame_number in range(start_frame, end_frame + 1, frame_interval):
            cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
            ret, frame = cap.read()
            if not ret:
                break
```

Step 5 : 라벨링 데이터를 Json으로 저장

```python
# 라벨링 데이터를 JSON 파일로 저장
label_output_path = os.path.join(current_dir, "output_lips", "label_info.json")
with open(label_output_path, 'w', encoding='utf-8') as f:
    json.dump(label_info, f, ensure_ascii=False, indent=4)
```

● 전처리한 시퀀스 데이터셋

⇒ 구문 별로 입술 이미지들이 저장되고 있음을 확인할 수 있다.

⇒ 하나의 구문 파일에서 확인한 시퀀 스 데이터 이미지

○ MobileNet v2 + LSTM 모델 구현 코드

```python
# LSTM 모델 정의
import torch.nn.functional as F

# MobileNet + LSTM 모델 정의
class MobileNetLSTM(nn.Module):
    def __init__(self, num_classes):
        super(MobileNetLSTM, self).__init__()
        mobilenet = models.mobilenet_v2(pretrained=True)
        mobilenet.features = nn.Sequential(
            *list(mobilenet.features),
            nn.AdaptiveAvgPool2d((1, 1))  # 추가: AdaptiveAvgPool2d로 마지막 출력 크기를 (1, 1)로 만듦
        )
        self.mobilenet = mobilenet.features
        self.lstm = nn.LSTM(1280, 256, batch_first=True)
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        batch_size, seq_length, c, h, w = x.size()
        x = x.view(batch_size * seq_length, c, h, w)
        x = self.mobilenet(x).squeeze(-1).squeeze(-1)  # (batch_size * seq_length, 1280, 1, 1) -> (batch_size * seq_length, 1280)
        x = x.view(batch_size, seq_length, -1)
        x, _ = self.lstm(x)
        x = self.fc(x[:, -1, :])
        return x
```

● 구현한 학습 모델 테스트 결과

ㄱ. 데이터 셋 : CIFAR18

※ Epoch : 10 , Loss : 1.28 → 0.28 감소

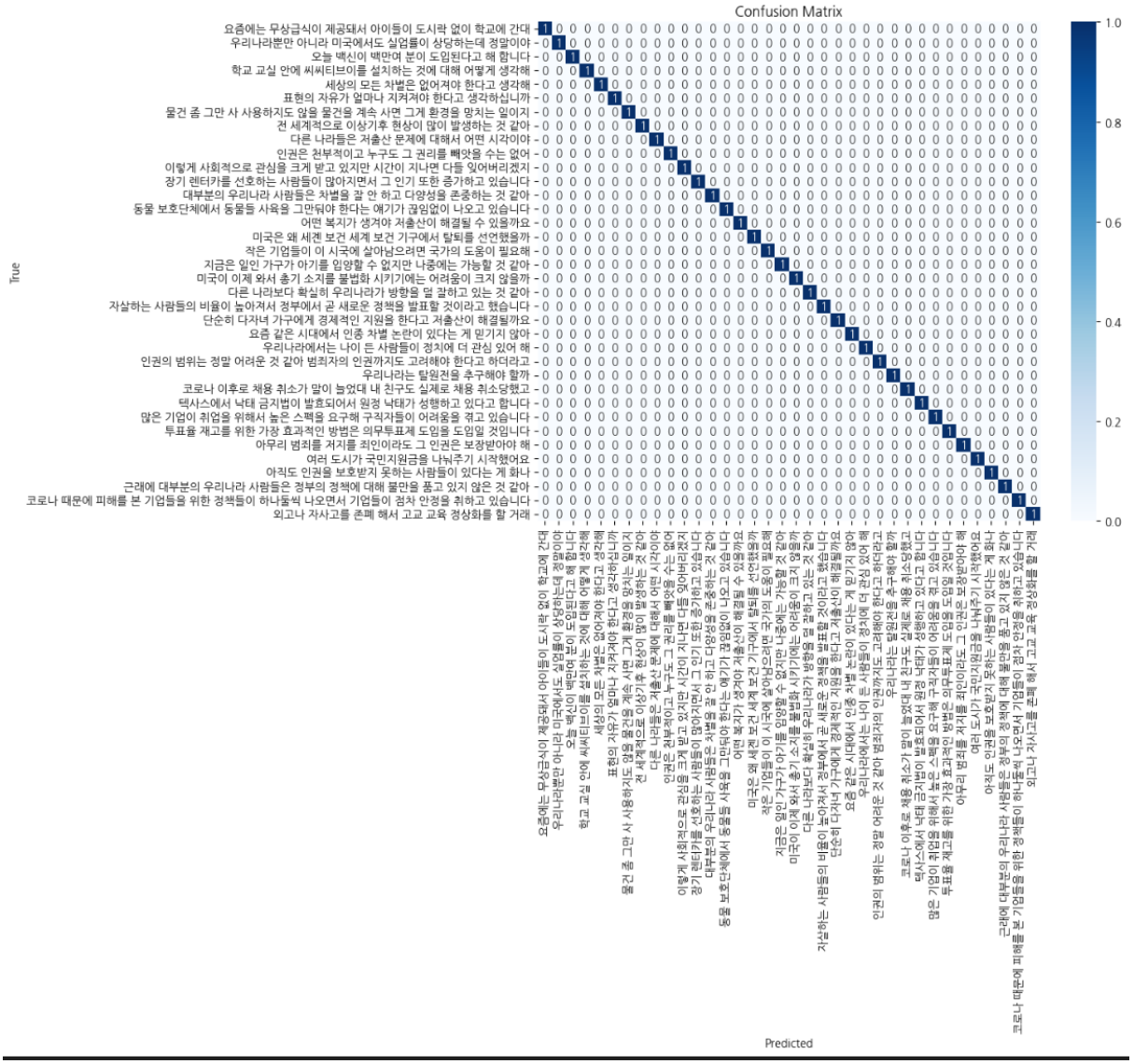
```
Epoch [1/10], Step [100/1563], Loss: 1.2836
Epoch [1/10], Step [200/1563], Loss: 0.9708
Epoch [1/10], Step [300/1563], Loss: 0.8691
Epoch [1/10], Step [400/1563], Loss: 0.7590
Epoch [1/10], Step [500/1563], Loss: 0.7544
Epoch [1/10], Step [600/1563], Loss: 0.7055
Epoch [1/10], Step [700/1563], Loss: 0.6978
Epoch [1/10], Step [800/1563], Loss: 0.6702
Epoch [1/10], Step [900/1563], Loss: 0.6709
Epoch [1/10], Step [1000/1563], Loss: 0.6282
Epoch [1/10], Step [1100/1563], Loss: 0.5897
Epoch [1/10], Step [1200/1563], Loss: 0.6251
Epoch [1/10], Step [1300/1563], Loss: 0.5961
Epoch [1/10], Step [1400/1563], Loss: 0.5891
Epoch [1/10], Step [1500/1563], Loss: 0.5725
Epoch [2/10], Step [100/1563], Loss: 0.5230
Epoch [2/10], Step [200/1563], Loss: 0.5155
Epoch [2/10], Step [300/1563], Loss: 0.5125
Epoch [2/10], Step [400/1563], Loss: 0.4872
Epoch [2/10], Step [500/1563], Loss: 0.5236
Epoch [2/10], Step [600/1563], Loss: 0.5053
Epoch [2/10], Step [700/1563], Loss: 0.4838
Epoch [2/10], Step [800/1563], Loss: 0.5112
Epoch [2/10], Step [900/1563], Loss: 0.5010
Epoch [2/10], Step [1000/1563], Loss: 0.4742
...
Epoch [10/10], Step [1300/1563], Loss: 0.2404
Epoch [10/10], Step [1400/1563], Loss: 0.2756
Epoch [10/10], Step [1500/1563], Loss: 0.2804
Finished Training
```

※ Test Accuracy : 89.12%

```
Accuracy of the model on the test images: 89.12 %
```

⇒ 이를 통해, 학습 모델이 잘 구현되었음을 확인할 수 있다.

ㄴ. 데이터 셋 : 영상에서 추출한 시퀀스 데이터(구문 1개) → 시퀀스 길이 : 100



⇒ 먼저 하나의 구문에 대해서만 학습을 진행하였고, 정확도가 100% 출력됨을 확인하였다.

○ 앞으로의 과제들

- 여러 명의 발화 데이터 셋을 학습시켜 정확도 확인
- '구문 → 단어'로 폴더 트리 구조 변경

○ 구현 코드

- 학습 모델 코드 : MobileNet + LSTM

lip_reading_train.ipynb

- 데이터 전처리 코드(이미지)

lib_reading_fromgpt.ipynb

- 데이터 전처리 코드(영상)

video.ipynb