# **Introduction**

# **ASL Recognition**

김선웅 방원준 인진교 정의환

Seoul National University

2017-16834 2018-16934 2019-10699 2018-19413

고성능의 번역 모델들이 등장하면서 여러 문화간 언어의 장벽이 낮아지는 중이다. 하지만 이러한 모델들의 수혜를 받지 못하고 여전히 고립되어 있는 언어가 있다. 바로 수화이다. 따라서 우리는 수화 번역을 위한 첫 단계로, 가장 기본적이고 널리 사용되는 수화인 ASL(American Sign Language) (Figure 1)을 문자로 변환하는 모델을 만들고자 한다. ASL은 전세계적으로 가장 널리 쓰이는 수화로, 그 범용성이 넓기에 선택하게 되었다.

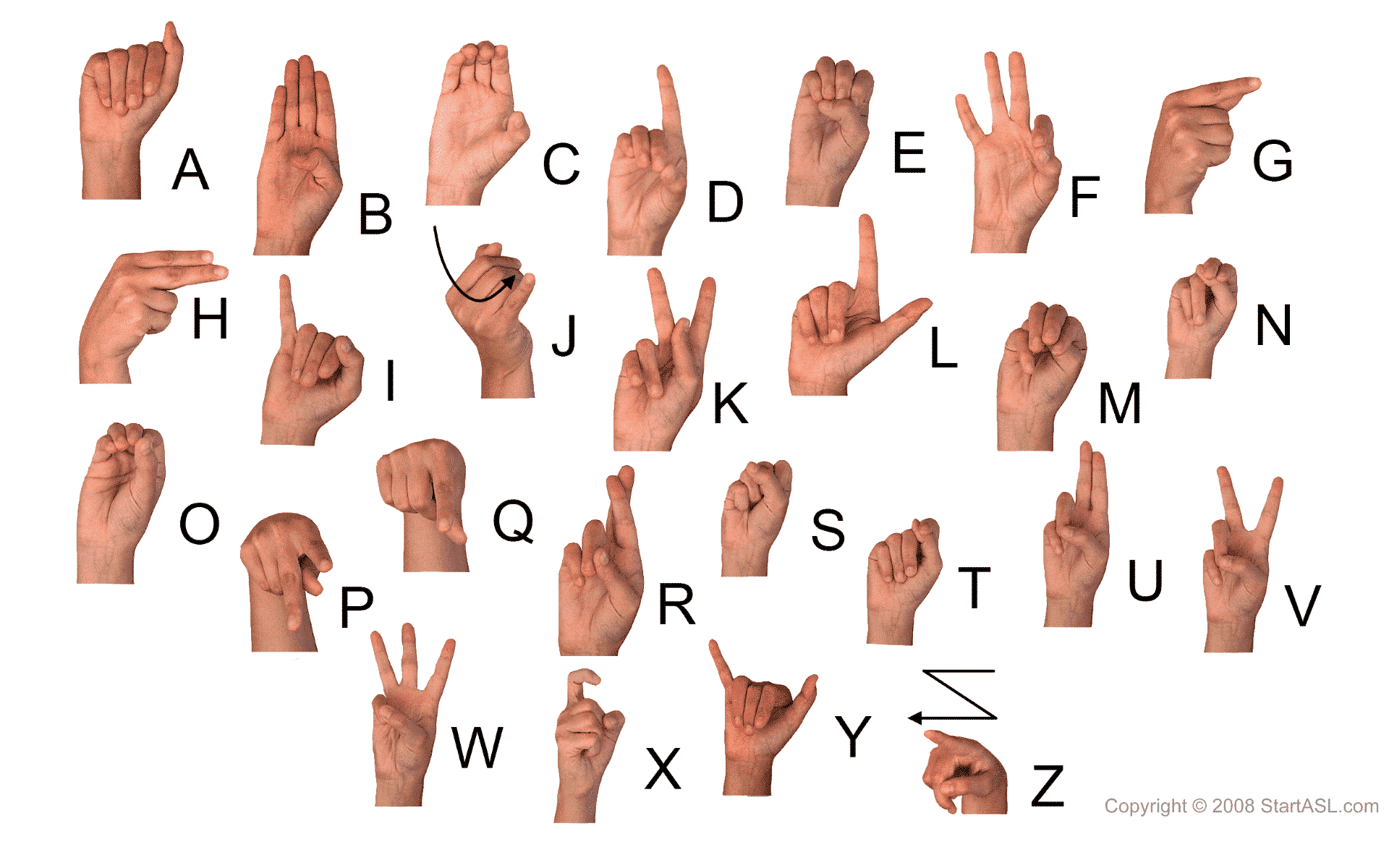


Figure 1: “Sign Language Alphabet | 6 Free Downloads to Learn it Fast”, StartASL, last modified February 15.2021, accessed December 15.2022, https://www.startasl.com/american-sign-language-alphabet/

.

# **Project Progression**

영상으로 주어진 input을 최종적으로 문자로 변환하기 위해 모델을 세 단계로 세분화하였다. 첫 단계에서는 영상을 프레임별로 split한다. 두 번째 단계에서는 split된 각 image에서 손을 감지하고 각 동작의 시작 프레임과 끝 프레임을 결정한다. 마지막으로, 결정된 각 동작의 sequence를 LSTM 모델에 통과시켜 Classification한다.

## **First Model**

우선 손의 인식은 Google에서 제공하는 AI 프레임워크인 MediaPipe(Figure 2)를 이용하였다.

MediaPipe는 이미지에서 손을 감지하고, 손을 21개 점으로 나누어 각 점의 좌표를 반환하는 모델이다.

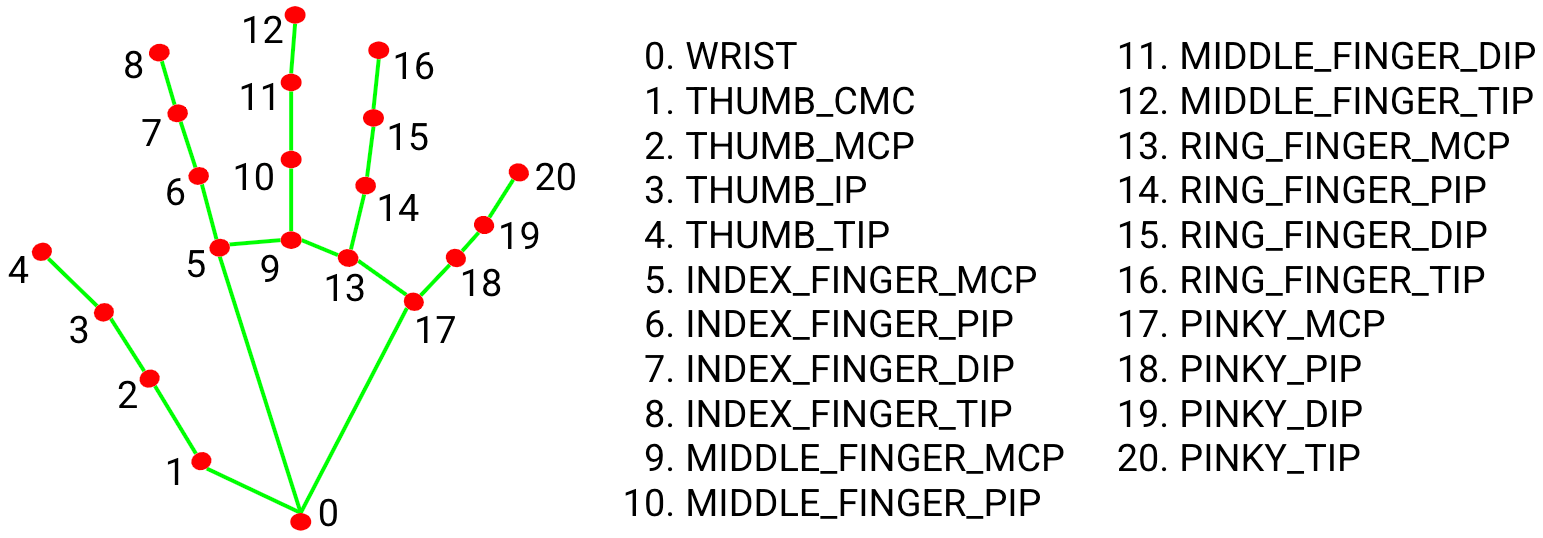
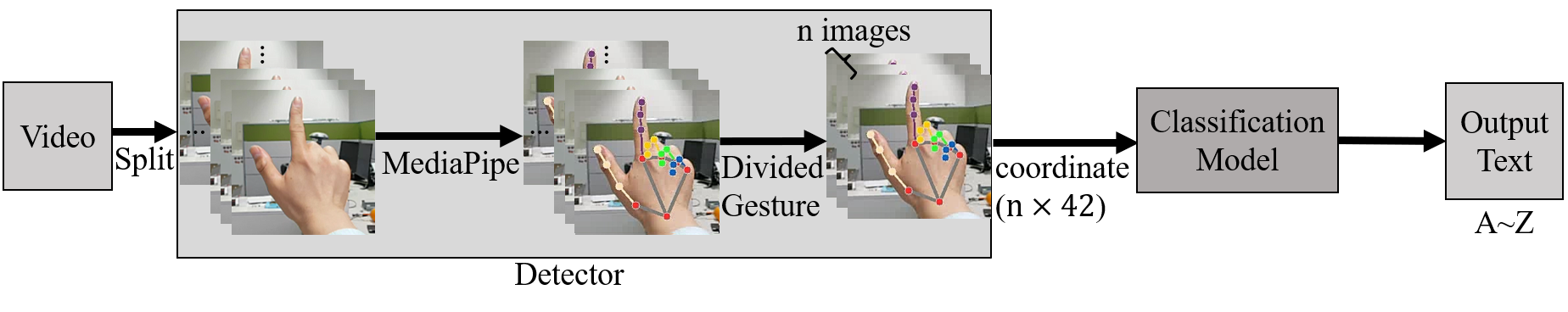
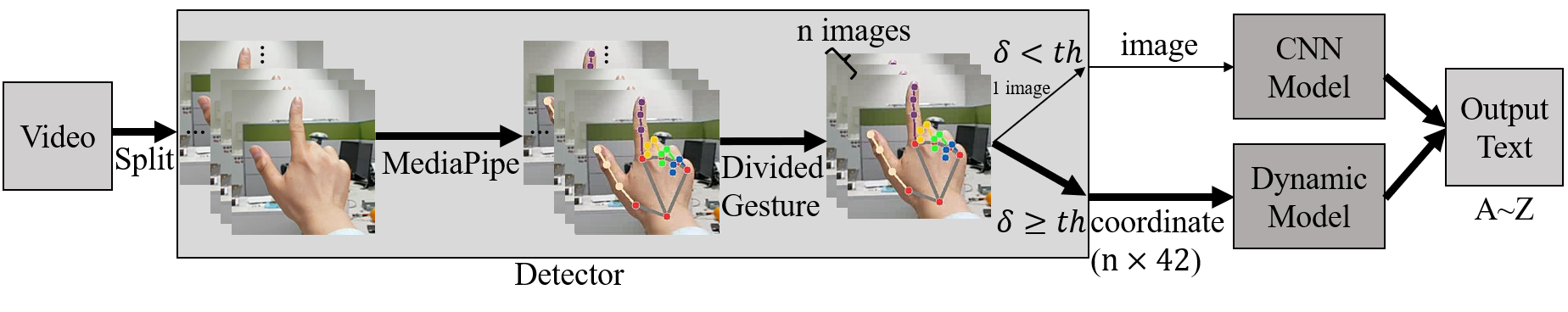


Figure 2: “MediaPipe Hands”, MediaPipe, accessed December 15.2022, https://google.github.io/mediapipe/solutions/hands

MediaPipe를 통해 얻은 21개 점의 (x, y) 좌표를 활용하였다. 각 동작의 sequence를 LSTM 모델에 입력해 Output을 얻도록 하였다. 초기 모델의 흐름도는 Figure 3에 나타나 있다.

## **Second Model**

여기에서, ASL의 특징을 더 잘 활용해 모델의 효율을 높이기 위해 새로운 방법을 고안했다. ASL은 26개의 알파벳 중 24개는 정적이며, 단 2개만이 동적이다. 이 2개의 동작만을 위해 모든 프레임 sequence를 사용해 구분하도록 하는 Classification 모델은 비효율적이므로, 모델을 사용하기 앞서 미리 동작을 Static과 Dynamic으로 구분하는 Detector를 사용하기로 하였다. 동작이 Static으로 판별된 경우, 해당 동작의 모든 sequence를 사용하는 대신 하나의 significant한 image만을 가져와 Image Classification에서 높은 성능을 보이는 CNN 모델을 사용하면 효율성을 대폭 높일 수 있다. 따라서 본 모델의 전체적인 흐름은 Figure 4와 같다.

모델의 input으로는 ASL Alphabet에 해당하는 몇 가지 동작 수행을 녹화한 동영상 파일이 주어진다. 이 input은 Detector에 입력되고, Detector에서는 전체 영상에서 유의미한 ASL 동작에 해당되는 부분의 시작 프레임과 끝 프레임을 찾아낸다. 추출된 각 동작이 정적인지 동적인지 판단하고, 정적인 경우 Significant Frame Image를 추출하여 Static Model인 CNN에 입력한다. 동적인 경우 해당하는 동작의 좌표 시퀀스를 2차원 tensor 형태로 변환하여 Dynamic Model인 LSTM에 입력한다. 각 Classifier는 알파벳 A~Z를 추측하여 그 결과값을 Output에 저장하고, 최종적으로 변환된 문자열이 출력된다.

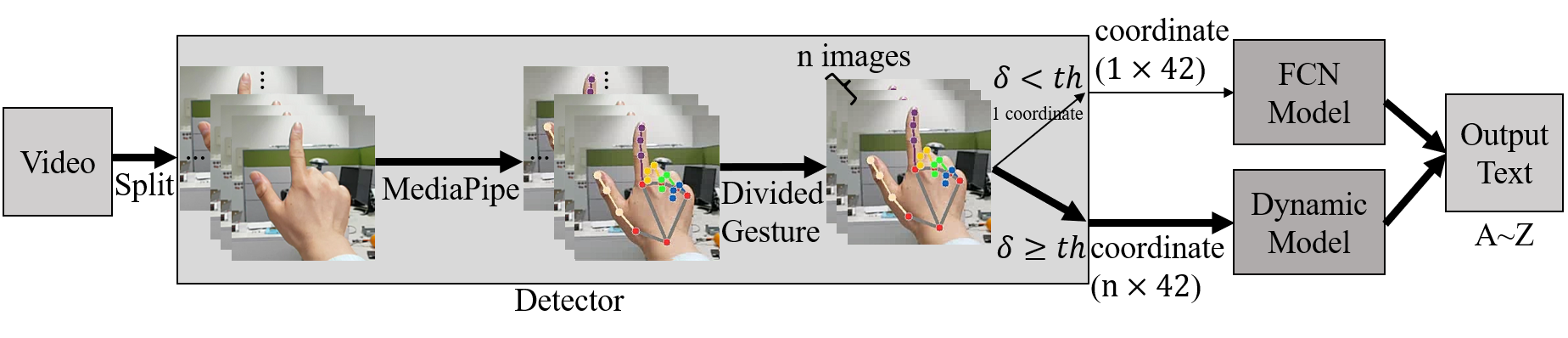


Figure 5: Block Diagram of Last Model

Figure 4: Block Diagram of Second Model

Figure 3: Block Diagram of First Model

## **Limit of Second Model and Last Model**

한 Image를 사용하는 CNN을 통해 모델의 연산량을 줄이고 속도를 올리는 아이디어는 좋았지만, CNN 모델의 정확도가 만족스러울 만큼 높지 않았다. 이에 Image를 2차원으로 분석하는 CNN의 특성이 본 목적에 잘 부합하지 않다고 여겨, 3차원 Rotation에 더 강건한 좌표를 이용해 output을 유추하도록 하였다. 마찬가지로 Significant Frame Image에서 좌표를 추출해 모델에 입력하면 Fully Connected Layer를 통해 24개의 Class를 구분하게 된다.

# **Code Description**

## **Detector**

Detector는 video input을 frame별 image로 split하고, frame들을 동작별로 구분하여 그룹화하는 역할을 맡는다. video\_splitter, delta\_coord\_gener-ator, frame\_classify, detect\_motion의 4가지 함수 및 보조 함수들로 구성되어 있다.

### **Video\_Splitter**

model의 input으로 들어온 video가 가장 먼저 통과해야 하는 함수이다. host pc에 저장되어 있는 video의 경로를 input으로 받아 프레임 단위의 image들로 split한 후 별도의 frame directory를 생성하여 저장한다. 그 후 생성된 frame directory 값을 output으로 반환한다.

### **delta\_coord\_generator**

video\_splitter가 반환한 frame directory를 받아 image에 대한 전처리를 수행하는 함수이다. 6가지 변수; delta\_coord, frame\_on, frame\_path, frame\_coord, h, w를 반환한다.

delta\_coord, frame\_coord: frame\_directory에 저장된 input image들을 시간순서대로 읽어온다. 각 image에 대해 MediaPipe hand model을 적용해 21개의 점에 대한 (x,y) 좌표, 총 42개의 좌표를 얻는다. Time step t에서 해당 좌표를 frame\_coord의 t번째 index에 리스트 형태로 저장한다.

해당 좌표값 리스트를 이용하여 현재 frame과 이전 frame에서 좌표값 차이의 L1 norm을 계산한다.

(1) 값을 라 정의한다.

그러나 주의할 점은, 모든 frame에서 hand가 detect되지는 않는다는 것이다. 따라서 MediaPipe Model이 hand를 인식하지 못한 경우를 위해 다음 규칙에 따라 값을 계산하였다.

1. timestep t와 timestep t-1에서 모두 좌표가 인식이 된다면, (1) 식을 통해 를 계산한다.

2. 둘 중 하나라도 인식이 되지 않았다면 으로 계산한다. 이렇게 계산된 모든 time step에서 값을 저장하여 이를 delta\_coord라는 list에 할당하였다.

frame\_on, frame\_path, h, w: frame\_directory에 저장된 frame들을 시간순으로 읽어오며, hand detection이 되는 frame의 경우 해당 index를 true로, detection이 되지 않는 경우 해당 index를 false로 저장하는 frame\_on 배열을 정의하고 이를 반환한다. frame\_path의 경우 각 frame에 대한 절대경로를 저장한 배열이다. h, w의 경우 각 frame image의 높이와 너비를 반환한다. 같은 영상에서 추출되었으므로 각 frame의 h, w 값은 모두 동일하다.

### **frame\_classify**

delta\_coord\_generator의 output인 delta\_coord와 frame\_on을 입력으로 받아 input video의 frame 중 유의미한 동작을 담고 있는 frame을 추출하는 역할을 한다. 화면에 손이 나타나 인식되는 순간을 start\_point, 손을 내려 인식되지 않는 순간을 end\_point로 지정하여 이 간격 사이 frame을 그룹화하고 반환한다.

### **detect motion**

frame\_classify의 output인 start\_points, end\_points와 delta\_coord\_generator의 output인 delta\_coord를 input으로 받는다. 이 함수는 (start\_points[i], end\_points[i]) 사이의 frame들, 즉 frame\_classify에서 판단한 그룹화된 frame들이 나타내는 알파벳이 dynamic ASL(j, z)인지 아니면 static(j, z를 제외한 나머지 알파벳)인지를 판단하는 역할을 수행한다. 해당 판정의 기준은 delta\_coord\_list를 이용한다. start\_points[i]와 end\_points[i] 사이의 frame들의 delta\_coord 값들을 관찰하며, 이 중 가 threshold 값 이상인 frame이 특정 개수 이상 존재하면 dynamic, 그렇지 않으면 static으로 판단한다. Threshold 값은 여러 값을 사용하며, 가장 정확하게 분류하는 값을 empirical하게 정하였다. 자세한 기준은 4절에서 서술한다.

## **Dynamic Predict**

dynamic predict는 ASL alphabet 중 동적인 j와 z를 classify한다. j와 z는 연속적인 동작으로, 동작 시간이 명확히 정해져 있고 개인차가 존재한다. 따라서 연속적이고 가변적인 길이의 데이터를 처리하는 것에 적합한 LSTM을 사용하였다.

### **dynamic\_image\_handtracker, dynamic\_img\_to\_np**

데이터를 전처리하는 과정에서 사용하였다. dynamic\_hand\_tracker는 이미지에 대해 손의 x좌표 21개, y좌표 21개를 각각 min-max normalization한 후 concatenation하여 42개의 좌표를 return하는 함수가 정의되어 있다. 이 때, 데이터셋을 만드는 과정에서 이미지가 좌우반전되어 있는 경우에 대해 opencv library를 활용하여 전체 데이터셋에 대한 일관성을 통일하도록 하였다.

dynamic\_img\_to\_np은 J나 Z의 한 동작을 numpy 배열로 저장하는 코드이다. 예를 들어 J를 표현하는 n개의 프레임에 대해서 각 프레임을 dynamic\_image\_handtracker를 통해 42개의 좌표를 얻은 후, 이를 각 프레임에 대해 stack하여 n \* 42 ndarray를 만든다. 이후 npy확장자로 저장하여 모델학습 데이터셋을 구축하였다. 이 때 특정 프레임에서 손이 흐릿하여 mediapipe가 인식하지 못하거나 2개의 손이 있다고 오인식하는 경우 해당 프레임은 포함하지 않는 등의 예외 처리를 하였다.

### **dynamic\_train\_SGD**

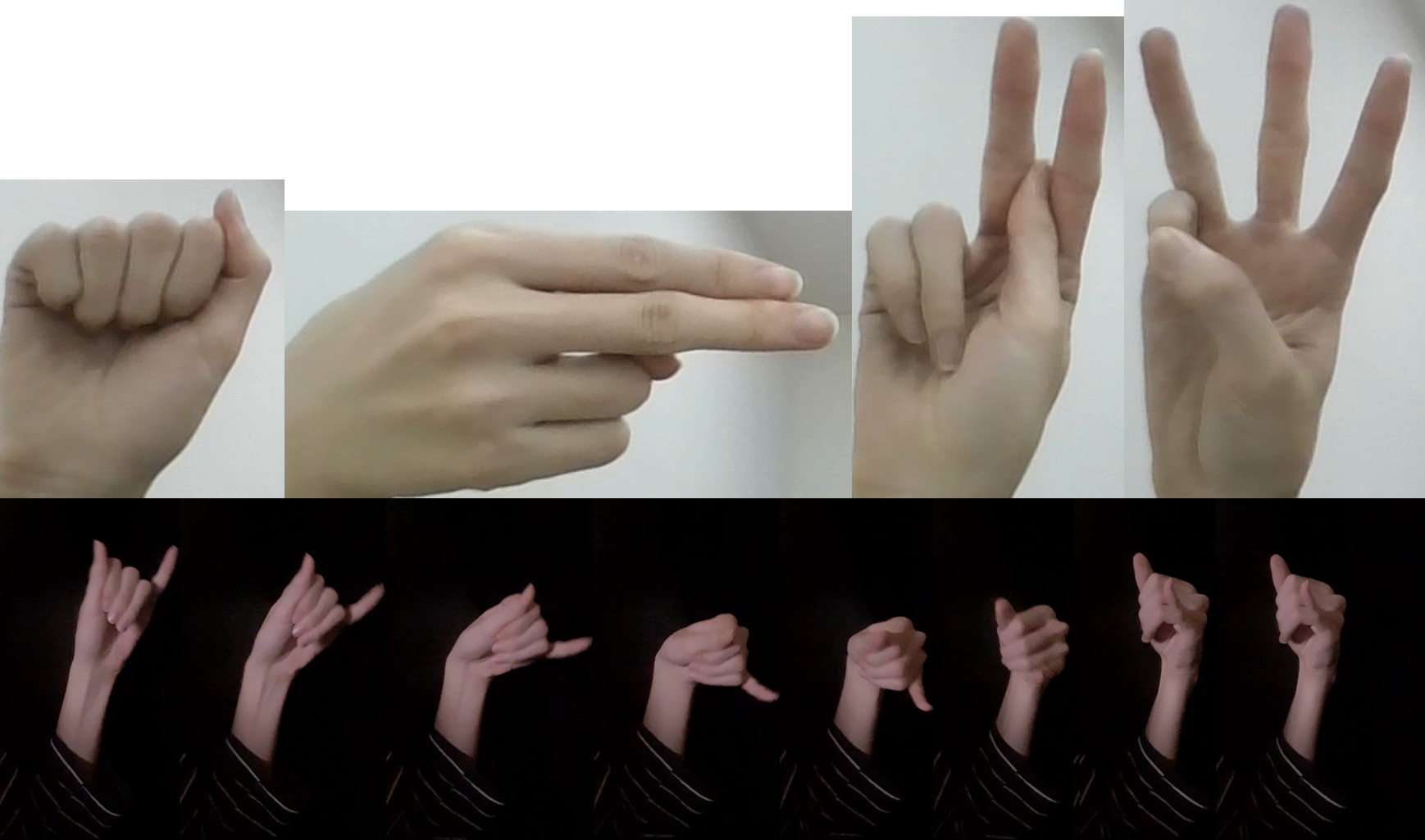
각각 데이터가 가변적인 길이를 갖기 때문에 batch로 다루기가 까다로워, SGD처럼 작동하도록 구현하였다. custom dataset class인 npydataset에서는 앞서 저장한 npy 파일을 읽어 각각을 ndarray로 저장한 후 list에 담아두는 역할을 한다. 이후 학습과정에서 하나씩 ndarray를 LSTM에 넣고, LSTM의 out을 Fully-connected layer에 통과시킨 후 sigmoid 함수를 거쳐 0과 1사이의 값을 갖도록 구현하였다. 또한 inference 시에는 input으로 들어오는 42개의 좌표에 대해 전처리를 할 때와 같은 정규화를 적용하고 모델의 output에 해당하는 alphabet을 return하도록 test\_print 함수도 구현하였다. 두 개의 class를 분류하는 문제이기 때문에 loss function으로는 pytorch의 BCELoss를 사용하였다. 분류하는 계급의 수가 적고 데이터의 구분이 쉽기 때문에 적은 epoch만으로도 학습이 잘 되었다.

Figure 6: Examples of Data Production: Static / Dynamic

Figure 6: Plot of Coordinate Delta

### **dynamic\_predict**

앞서 학습한 모델이 저장된 .pt 파일을 읽기 위해 class가 정의만 되어있는 파일이다.

## **Static Predict**

mediapipe에서 도출된 값은 마치 CNN에서 covolutional layer가 feature detection 역할을 수행하는 것과 비슷하다. 따라서 우리는 마치 전이학습(Transfer Learning)에서 기존에 학습된 layer에 fully connedted layer를 연결하는 것처럼, mediapipe로 도출된 좌표에 2개 층의 fully connected layer를 적용하여 24개의 classfication을 수행하였다.

### **static\_image\_handtracker, static\_img\_to\_np**

앞서 서술했듯 처음에는 static predict를 CNN을 통해 예측하려고 했으나 실패하였다. 따라서 생성한 dataset이 손의 크기만큼 cropped된 이미지였다. 이 이미지를 mediapipe에 인식시키면 인식률이 크게 떨어져, opencv의 padding을 활용해 일반적인 사진의 크기(약 1200x700)로 만드는 부분을 추가로 구현하였다. static\_img\_to\_np에서는 padding된 이미지에서 손이 검출된 경우에만 numpy를 저장하도록 하였다.

이를 통해 약 17000장의 dataset을 구축하였다.

### **static\_train**

A~Z를 하위 디렉토리로 갖는 각 폴더에 대해서 dataset을 생성한 후, ConcatDataset으로 모든 dataset을 합쳤다. 이후 random\_split으로 trainset과 validset을 적당한 비율로 나누어 준 후 학습을 진행하였다. 24개의 class임에도 좋은 학습 결과를 만들어낼 수 있었다.

# **Experiments**

## **Dataset Production**

수화 인식의 특성상 여러 사람의 손을 인식할 수 있어야 하므로, 하나의 손이 아닌 여러 모양의 손으로부터 수화 이미지가 필요했다. 따라서 각 조원들이 직접 수화를 촬영하여 Dataset을 제작했다. Figure 6에서와 같이, Static Alphabet들은 정지된 이미지, Dynamic Alphabet은 연속된 이미지를 이용했다.

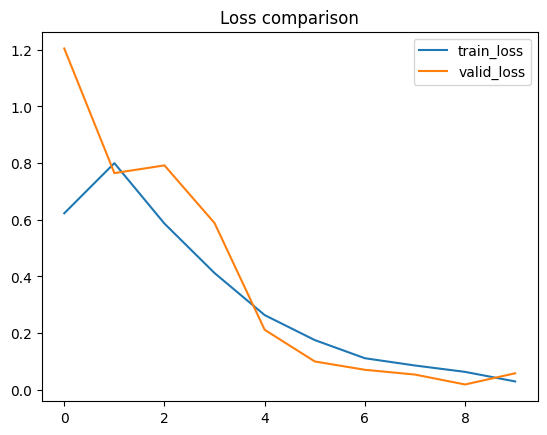
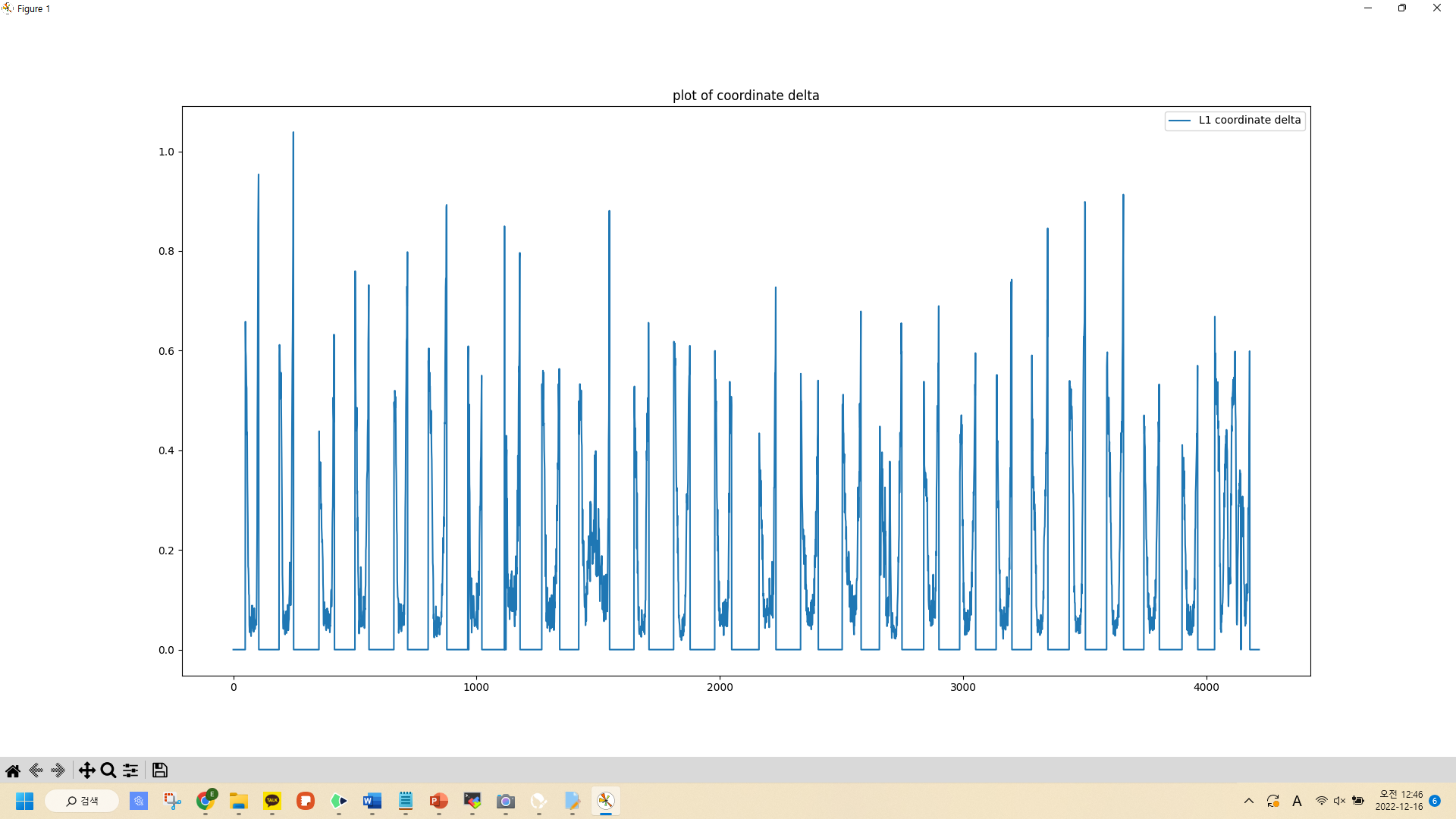
****

Figure 8: Loss Comparison for Dynamic (LSTM)

Figure 7: Plot of Coordinate Delta for A to Z

## **Static/Dynamic Threshold in Detector**

앞서 소개한 Detector에서, Static과 Dynamic을 구분하는 의 threshold 값을 empirical하게 정했다고 소개하였다. 적절한 Threshold 값을 찾기 위해, A to Z의 video를 제작해 delta\_coord 값을 출력하도록 하였다. Figure 7에서 볼 수 있듯 Static과 Dynamic Alphabet 사이 확연한 차이가 나므로, 두 경우를 명확하게 구분할 수 있는 threshold를 사용할 수 있었다.

## **Train / Validation Loss Graph of Models**

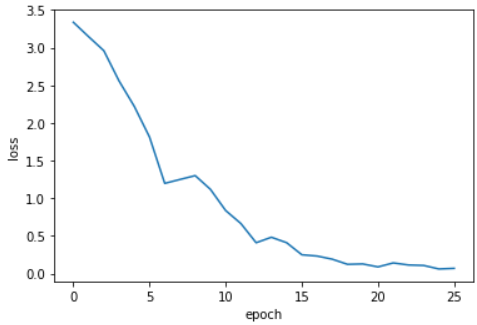
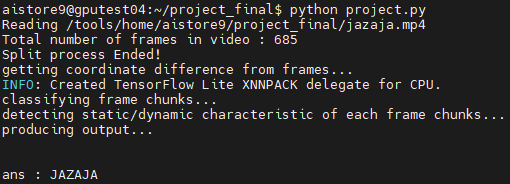
Figure 9: Loss Comparison for Static (FC)

### **Dynamic Model (LSTM) / Static Model (FC)**

두 모델의 Train / Validation Loss는 각각 Figure 8, 9와 같다. 학습이 잘 진행된 것을 확인할 수 있다.

### **Static Model (CNN)**

초창기 모델에서는 Static Alphabet을 위해 CNN을 사용하였다고 소개했다. 하지만 성능이 만족스럽지 않아 좌표를 활용한 FC 모델로 변경하게 되었다. 따라서 사용하였던 CNN 모델과, CNN이 적합하지 않았던 이유를 분석해본다.

사용했던 모델 구조는 다음과 같다. Detector가 주어진 동영상에서 J나 Z가 아닌 static alphabet이라고 인식한 부분에서 손 부분만 잘라서 이미지를 주면 CNN을 통해 이미지에서 24개의 알파벳을 인식하도록 했다. 먼저 고정된 CNN network로 학습을 진행하기 위해 데이터 전 처리 과정에서 데이터 크기를 112x112으로 고정시켜 주었다. 그 후 5개의 CNN layer을 만들어 4번의 max-pooling을 진행해 이미지 사이즈를 절반씩 줄여 나가고, channel수는 4에서 시작해 16까지 늘려서 최종 output은 16채널의 7x7크기의 이미지가 되도록 했다. CNN의 세번째 layer의 output과 5번째 layer의 output과 더해주도록 해서 gradient가 더 잘 전달되도록 했다. 마지막에 이미지를 일렬로 편 후, Fully connected network로 16x7x7데이터에서 24개의 벡터를 출력하도록 하였다.

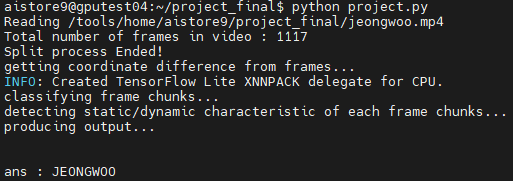
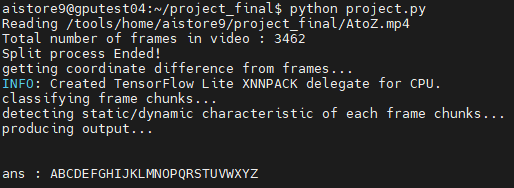


Figure 12: Extreme Case

Figure 11: Inference Examples

Figure 10: Loss for CNN

Torch.util.data의 dataset 구조를 새로 정의해서 dataloader를 이용해서 학습을 더 용이하게 진행했다. 학습은 pytorch를 이용하여 진행했으며 Cross entropy Loss를 loss function으로 이용하고, optimization method로는 Adam을 이용했다. Training data에 대해서는 학습은 잘 되어 Figure 10과 같은 loss값 그래프를 보여준다.

CNN 구조는 B, I, L, V, W, Y 와 같이 다른 손 모양과 겹치지 않는 알파벳 손 모양은 잘 인식했다.

그러나 수화 중 주먹 쥔 손모양인 M, N, S, T, A와 같은 모양을 잘 구분하지 못했다. testing set에서는 회전시킨 손 모양을 잘 인식하지만 그 과정에서 3차원 상에서는 다른 손 모양이 2차원에서는 같은 모양으로 인식이 되어 모델이 구분하는 이미지 두 손 모양 사이의 거리가 작아지는 경우가 있어 잘 구분이 되지 않았다. 회전을 배제하고 정확히 정면에서 본 데이터만을 이용하여 모델을 학습해보니 기존 모델보다는 인식률이 높아졌지만, 손 모양이 조금만 달라져도 인식이 되지 않는 문제가 있었다. 물론 데이터를 더 모으고 모델을 더 복잡하게 하면 충분히 가능할 수도 있었지만 데이터를 무한정 만들 수는 없기에 처음 계획과는 다른 방법을 찾아보게 되었다.

# **Results**

## **Inference Results**

사람의 이름 같은 General한 Case나 A to Z같은 Long Sequence에 대해서도 100%의 정확도를 보이는 것을 확인할 수 있다. 임의의 input에 대해서도 항상 100%의 정확도를 보였다.

## **Inference Results**

기존의 모델과는 다르게, Dynamic Alphabet이 Static과 High Frequency로 번갈아 등장해도 모두 성공적으로 인식하는 것을 Figure 12를 통해 확인할 수 있다.

# **Conclusion**

## **Project Flow**

처음에는 CNN -based sign language classification에 관련된 논문을 읽고, 이와 연관된 여러 오픈소스 코드들을 공부하며 해당 알고리즘에 대해 공부했다. 한편, 이러한 모델들이 dynamic sign language 에 대해서는 잘 처리하지 못한다는 것을 파악하여 RNN combined ASL classifier 구조를 설계하기로 계획하였다. 이 과정에서 detector 라는 구조를 떠올리게 되었고, 이를 mediapipe 라는 pretrained 된 모델의 output을 이용하여 구현하였다. Static Model을 구현하는 과정에서도 CNN의 특성이 수화인식과 잘 맞지 않을 수 있음을 파악하고 좌표 기반의 FC으로 변경하였다. 결과적으로, static/dynamic class 모두에서 정확한 sign language translation 이 가능한 모델을 구현하는데 성공하였다.

조사해본 대부분의 논문들은 CNN만을 사용하여 j, z 같은 alphabet을 classification하지 못하거나, RNN만을 사용하여 연산 효율이 떨어지는 등의 문제가 있었다. 하지만 본 모델은 Detector를 통해 효율적인 모델 분배를 하였으며, CNN보다도 높은 정확도의 모델을 구현하는 것에도 성공하였다.

## **Further Study**

앞서 모든 inference에서 100%의 정확도를 기록했지만, 당사자가 화면의 정면을 향하고 있지 않은 경우, 각도가 심하게 기울어진 경우에는 잘못된 output을 도출할 가능성이 존재한다. Figure 13이 그 예로, Q 동작을 하고 있지만 위로 45도 정도 기울어져 있었기 때문에 실제 output은 A를 도출하였다. 이와 같은 경우를 방지하기 위해, 시각적 정보를 좀 더 활용할 수 있도록 현재 FC에 이전에 사용했던 CNN을 weighted sum으로 결합해 더 높은 신뢰도의 결과를 얻을 수 있을 것이다.



Figure 13: Wrong Q

# **Contribution**

김선웅 - dynamic network 성공적 구현, static network FC 성공적 구현, 코드 주석 검토, validation data 10% 생성, static data 1/4 생성, dynamic data 생성,

방원준-detector 구조 구현, detector 지속적 decoding 및 수정, 발표 자료 제작, 전체적인 모델 discussion, dynamic data 생성,

안진교 - 보고서 양식 정리 및 최종 검수, 전체 모델으로 각각의 모듈 연결, static data 2/3 생성, dynamic data 생성, validation data 90%생성,

정의환 - datector 초기 구조 개발, static alphabet CNN 모델 구현, 발표 및 발표 자료 제작, static data 1/12 생성, dynamic data 생성

# **References**

1. MediaPipe, Google, https://mediapipe.dev/
2. Sign Language Recognition with RNN and MediaPipe, rabBit64, <https://github.com/rabBit64/Sign-language-recognition-with-RNN-and-Mediapipe>
3. DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals, Ahmed KASAPBASI. et al, <https://www.sciencedirect.com/science/article/pii/S2666990021000471>
4. Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks, Okan Kopuklu. et al, 2019