2020년 종합설계1

졸업작품 최종 보고서

시각장애인을 위한 상황 묘사 어플리케이션

Project Team

Team 4

Date

2020-06-18

Team Information

201113275 정준호

201411802 전민규

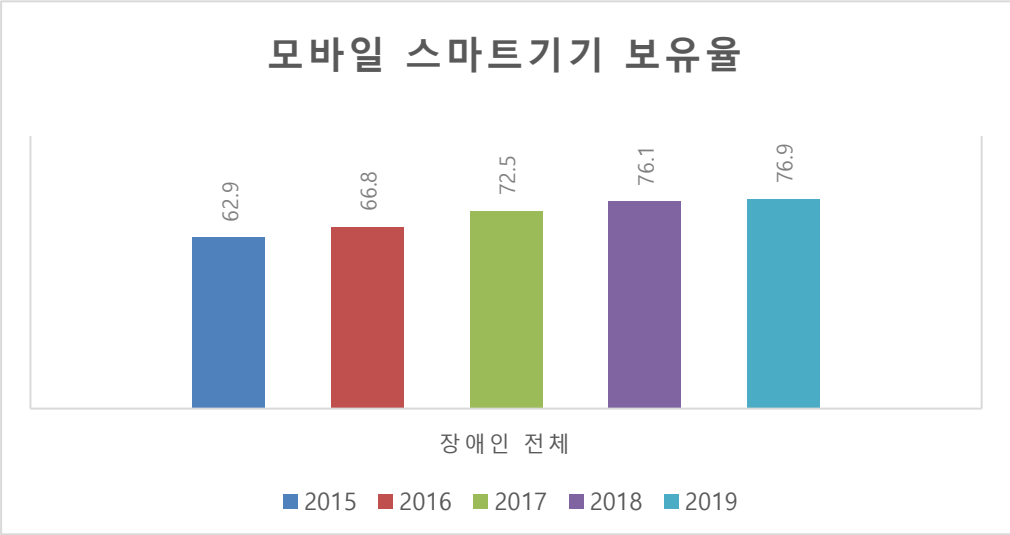
201614157 김도연

목차

1. 주제 선정 배경 및 목적

1.1 배경

장애인 스마트 보유율이 매해 증가하고 있는 추세이다. 하지만 그에 반해 장애인들이 사용할 수 있는 어플리케이션은 현저하게 적다. 장애인에게 정보를 제공하는 어플리케이션이 많다면 스마트폰을 보유하고 활용하는 장애인에게 적잖은 도움이 될 것이다. 장애인분들 중 시각장애인이 스마트폰을 사용하지 못 할 것이라는 편견이 있지만 실제 시각장애인에게 스마트폰은 많은 도움을 줄 수 있다. 첨단 기술을 적용하여 어플리케이션을 만든다면 주변 도움을 받아야 가능했던 일들을 스스로 수행할 수 있게 된다. 시각 장애인이 길을 걸을 때, '공사중 주의' 표지판을 인식하고 비켜갈 수는 없을 것이다. 하지만 전방의 상황을 알려주는 어플리케이션이 있다면 손쉽게 정보를 얻어 '공사중 주의'라는 정확한 정보는 아니더라도 표지판이 있다는 것을 알려주어 앞을 가는데 좀 더 신중할 수 있도록 도움을 줄 수 있다.



※ NIA한국정보화진흥원 – 디지털정보격차 실태조사

1.2 목적

접근성이 좋은 스마트폰의 어플리케이션을 사용하여 '시각장애인' 및 '저시력자' 등 시각의 보조가 필요한 사람들에게 카메라로 찍히는 전방의 상황에 대한 정보를 음성으로 알려줌으로써 조금이나마 생활하는데 도움이 되는 것을 목적으로 한다.

2. 개발 목표 (Success Criteria)

- 학습된 모델을 평가하였을 때, 평균적인 BLUE Score-4 기준 어절단위 0.111 의미형태소 단위 0.225 형태소 단위 0.251 이상이 나오도록 한다.
- 전방의 상황을 최대한 빨리 알려주기 위하여 이미지를 찍고 그 이미지의 상황에 부합하는 문장을 음성으로 출력하는 시간까지 10초 이내여야 한다.
- 사진은 10초마다 찍는다.

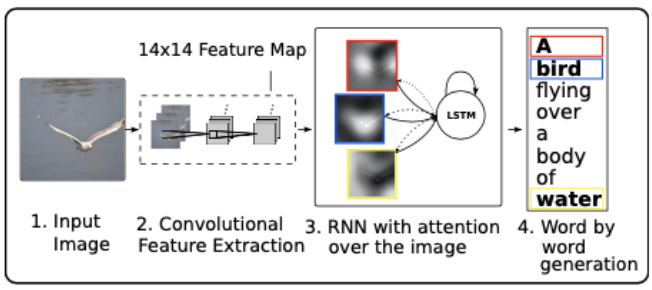
3. 요구사항 분석

3.1 기능 요구사항

1. 전면부 상황 입력
   1.1 10초 마다 주기적으로 사진을 촬영
   1.2 주어진 사진을 모델단에 입력

2. 딥러닝 학습을 통한 모델
   2.1 MS COCO 캡셔닝 데이터(이미지 약 12만장)를 사용해서 모델을 학습
      2.1.1 MS COCO 캡셔닝 데이터
         이미지 + 사진을 설명하는 약 5개의 문장들
      2.1.2 모델 설명
         input으로 들어온 사진의 특징을 잘 뽑아내는 이미지 처리 모듈(CNN, 인코더)을 거쳐 이미지에 대한 특징을 추출한다. 추출한 특징을 가지고 문장을 생성해내는 모듈(LSTM, 디코더)에 들어가 사진을 설명하는 문장을 출력한다
         (위 과정은 결국 input으로 이미지, output으로 문장이 나오는 구조이며 인코더에서 디코더로 따로 수동으로 넣어 줄 필요없이 들어갑니다.)

3. 학습된 모델을 통한 결과 도출
   모델단에 input한 사진을 학습된 모델의 weight를 통해 그에 잘 맞는 문장을 output으로 출력한다. (end to end로 input을 받아 모델을 한번 거치는 구조)



4. TTS (Text To Speech)

텍스트를 음성으로 변환하여 생성된 문장을 읽어주는 기능

3.2 비기능 요구사항

1. 응답 시간 / 성능(performance)
   : 사용자 입력으로부터 결과를 도출해내 보여주는 시간
   : 사진 한 장을 입력으로 넣었을 때 그 사진에 대한 설명이 텍스트 형태로 나오고 음성 메시지로 들려주는데까지 10초 이내여야 한다.

2. 명확한 상황 묘사 / 신뢰성(reliability)
   : 좀 더 정확하게 전방의 상황을 알리기 위한 묘사
   : MS COCO data에 있는 데이터 중 일부(12만장 中 약 5천장)를 Test data로 사용해 학습이 완료된 모델을 테스트한다.
   : BLEU-4 기준(주로 BLEU-4내외의 값을 중요하게 생각하기 때문에)
   어절단위 0.111 의미형태소 단위 0.225 형태소 단위 0.251 이상
   ※ BLEU(Bilingual Evaluation Understudy)score란 성과지표로 데이터의 X가 순서정보를 가진 단어들(문장)로 이루어져 있고, y 또한 단어들의 시리즈(문장)로 이루어진 경우에 사용되며, 번역을 하는 모델에 주로 사용된다.
   - n-gram을 통한 순서쌍들이 얼마나 겹치는지 측정(precision)
   - 문장길이에 대한 과적합 보정 (Brevity Penalty)
   - 같은 단어가 연속적으로 나올때 과적합 되는 것을 보정(Clipping)

$$BLEU = min(1, \frac{output\ length(\text{예측 문장})}{reference\ length(\text{실제 문장})})(\prod_{i=1}^{4} precision_i)^{\frac{1}{4}}$$

4. 개발 환경

- 언어
  Client – JAVA (Android)
  Server – Python 3

- Framework / Library
  Pytorch, Komoran, Okt, Kkma

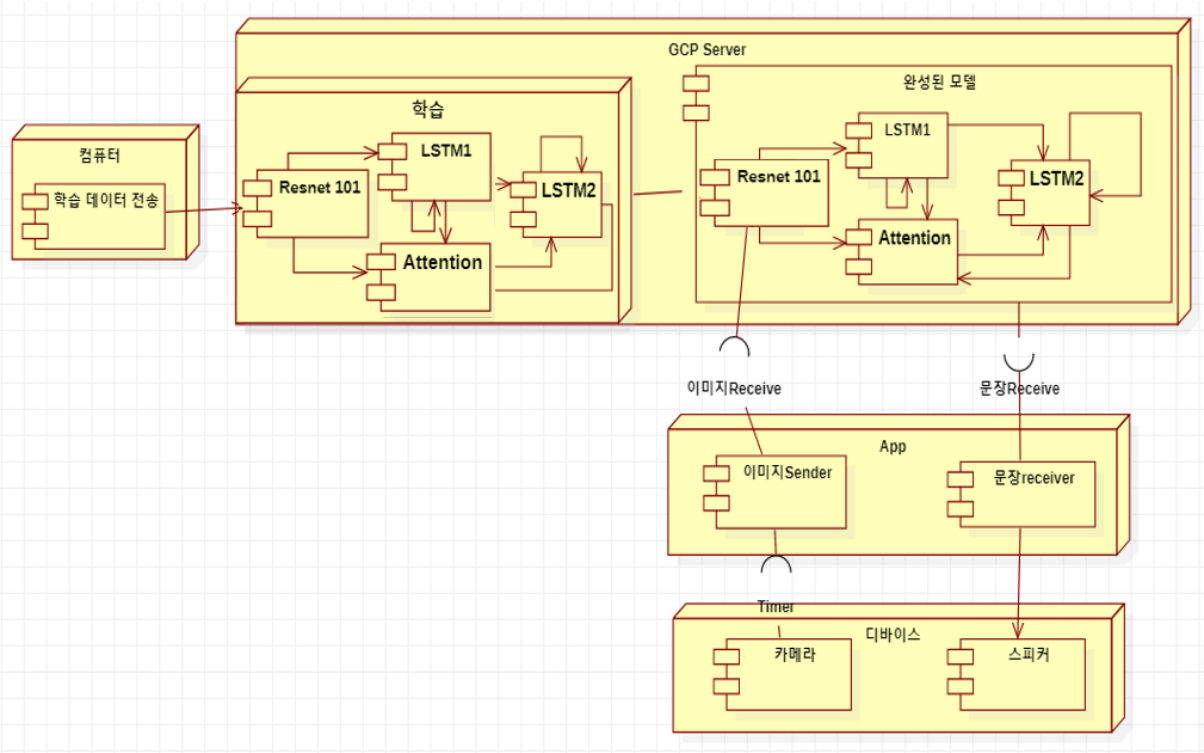
- 개발 도구
  Android 스튜디오, Pycharm, GCC, Visual Studio Code

- OS
  Windows, Ubuntu 18.04 LTS

- 테스트 디바이스
  갤럭시(스마트폰)

5. 설계 구조 및 동작 과정

5.1 설계 구조 (Component Diagram)



1. APP
   - 이미지 Sender : 이미지를 찍으라고 Device에게 요청한다.
   - 이미지 Sender : 그 이미지를 GCP Server에게 보낸다.
   - 문장 receiver : GCP Server안의 학습된 모델을 통해 나온 문장을 전달받는다.
   - 문장 receiver : TTS API를 사용하여 전달받은 문장을 음성으로 바꿔준다.
   - 문장 receiver : App은 Device의 스피커를 사용하여 음성을 출력하여 사용자에게 들려준다.
   -
2. GCP Server
   - 학습 : GCP Server 안에서 MSCOCO 이미지 데이터를 가지고 이미지에 맞는 문장과 함께 학습을 시킴 (Resnet101, LSTM1, LSTM2, Attention)
   - 완성된 모델 : 학습시켜 완성된 모델을 두어 이미지를 App에게 전달받으면 학습된 모델을 통해 이미지에 부합되는 문장을 출력한다.(Resnet101, LSTM1, LSTM2,

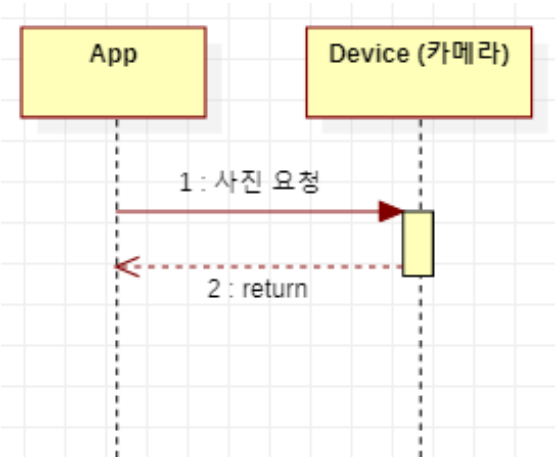
Attention)

- 완성된 모델 : 문장을 App에게 전달한다.

5.2 동작 과정 (Sequence Diagram)

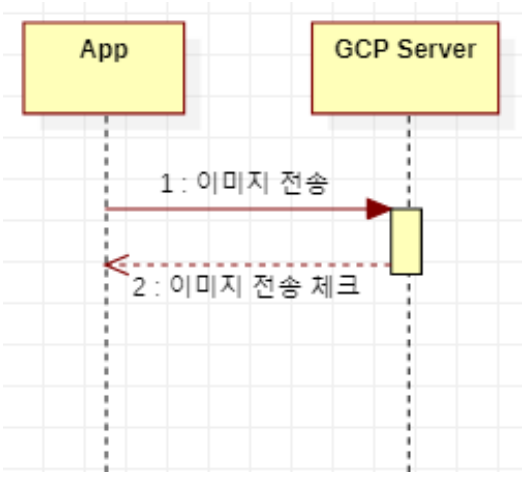
1. 전면부 상황 입력

1.1 10초마다 주기적으로 사진을 촬영



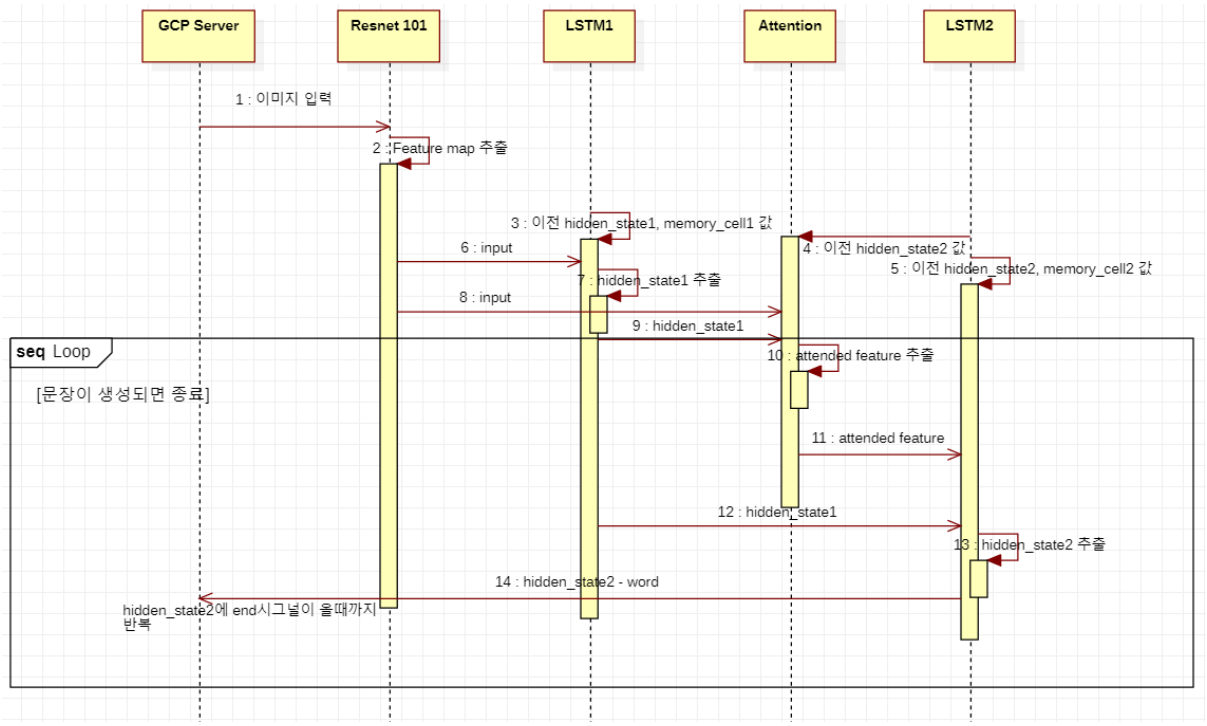
1. App이 Device에게 사진을 요청한다.
2. Device는 10초 내외로 사진을 찍어 App에게 사진을 return한다.

1.2 주어진 사진을 모델단에 입력



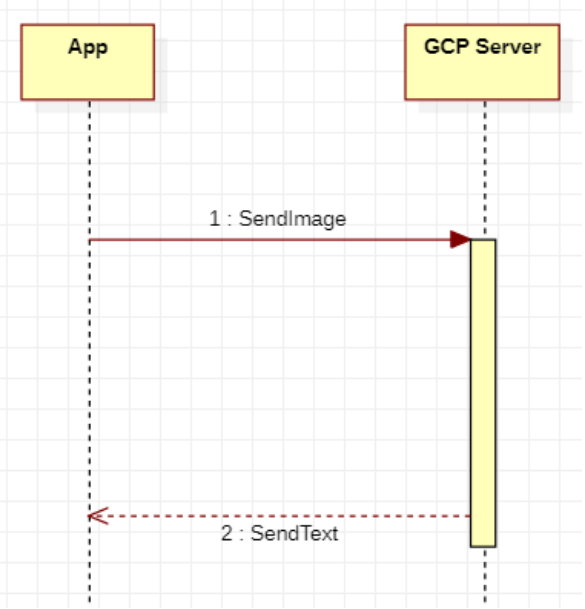
1. App이 사진을 GCP Server에 전송한다.
2. (GCP Server 내부) GCP Server에 있는 학습된 모델단에 사진을 넣어준다.
3. GCP Server가 모델단에 사진이 들어오면 App에게 안전하게 전송되었음을 알린다.

2. 딥러닝 학습을 통한 모델



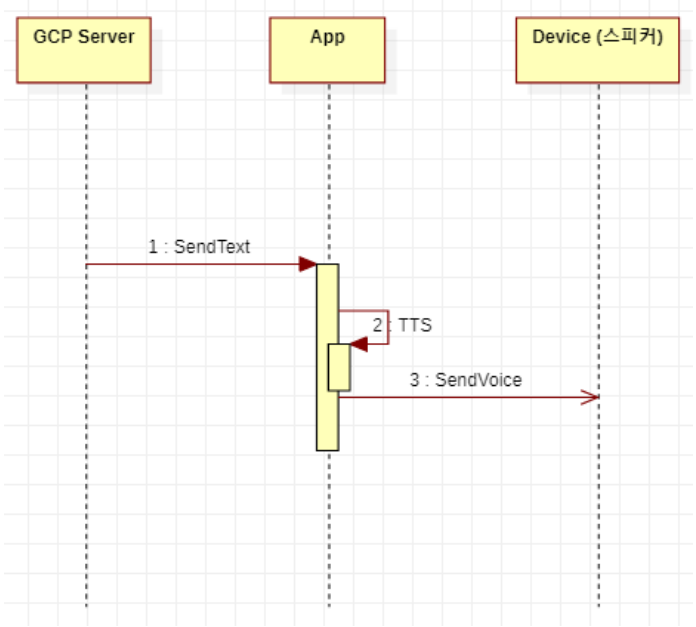
1. GCP Server가 Resnet 101에게 이미지를 입력한다.
2. Resnet 101라는 이미지 처리하는 모델을 사용하여 사진에 대한 feature map 정보를 받는다.
3. Resnet 101에서 처음에 feature A 정보와 대응하는 word embedding값의 concat한 결과와 랜덤으로 초기화한 hiddens state vector h_{t-1}1, cell state vectorc_{t-1}1 이 LSTM1의 input으로 들어간다.
4. (Attention)LSTM1의 출력값 중 ht_1과 feature A가 Attention의 인풋으로 들어간다.
5. (LSTM2)attention의 output a_t와 LSTM1의 출력값 중 h_t1을 concat하여 랜덤으로 초기화한 hidden state vector h_{t-1}2와 cell state vector h_{t-1}2 와 함께 LSTM2의 input값으로 들어간다.
6. 위 과정이 재귀적으로 이어진다.

3. 학습된 모델을 통한 결과 도출



1. Device에게 받은 사진을 App이 GCP Sever에게 전달해준다.
2. GCP Server에 있는 학습된 모델(2번 Sequence)을 거쳐 사진에 알맞은 상황묘사 문장을 만든다.
3. 만들어진 문장을 GCP Server가 App에게 전달해준다.

4. TTS (Text To Speech)



1. GCP Server가 문장을 만들어 App에게 전달한다.
2. 문장을 받은 App은 TTS API를 사용하여 음성으로 바꿔준다.

3. 음성을 Device를 통해 User에게 들려준다.

5.3 Traceability Matrix

Test Case	TC Desc	Requirment NO.	Req Desc	Design (H) (Module)	Design(모듈) NO.	Seq Diagram NO.	Design(H) Desc	Design (L) (Class)
1-1	정해진 시간에 사진이 찍히는가	1-1	10초 마다 주기적으로 사진을 촬영	App	A.1	1	사진 요청	CameraFragment
				Device	D.1	2	사진 return	카메라
1-2	찍힌 사진이 모델단에 제대로 들어가는가	1-2	주어진 사진을 모델단에 입력	App	A.2	1	사진 전송	CameraFragment
				GCP Server	G.1	2	사진 전송 체크	Resnet 101
2	신뢰도 평가	2	딥러닝 학습을 통한 모델	GCP Server	G.2	1	사진 입력	Server
				Resnet(Encoder)	R.1	2	feature map (A) 추출	Resnet 101
					R.2	6	추출한 A input	Resnet 101
					R.3	8	추출한 A input	Resnet 101
				LSTMCell1	L1.1	3	이전 hidden_state1, memory_cell1 값 input	LSTMCell
					L1.2	7	hidden_state1 추출	LSTMCell
					L1.3	9	hidden_state1 input	LSTMCell
					L1.4	12	hidden_state1 input	LSTM1
				attention	Att.1	10	attended feature 추출	Attention
					Att.2	11	attended feature input	Attention
				LSTM2	L2.1	4	이전 hidden_state2 output	LSTM2
					L2.2	5	이전 hidden_state2, memory_cell2 값 input	LSTM2
					L2.3	13	hidden_state2 추출	LSTM2
					L2.4	14	hidden_state2-word output	LSTM2
3	학습한 모델에 들어가 정해진 시간내에 결과가 도출하는가	3	학습된 모델을 통한 결과 도출	App	A.3	2	SendImage	CameraFragment
				GCP Server	G.3	3	SendText	Server
4	도출한 결과(문장)를 음성으로 바꿔서 출력하는가	4	TTS (Text To Speech)	GCP Server	G.4	1	SendText	Server
				App	A.4	2	TTS	MainActivity
				Device	A.5	3	음성출력	스피커

6. 구현 내용

6.1 데이터 전처리

#1 AI HUB 에서 받은 MSCOCO_train_val_Korean.json 을 한글 토크나이징 라이브러리인 KoNLPy 를 사용하여 (Okt, Komoran, Kkma 형태소 분석기) 한국어 문장을 형태소 단위로 나눠준다.

Ex 1. <나는 밥을 먹는다>

Kkma

'나', '는', '밥', '을', '먹', '는', '다'

Komoran

'나', '는', '밥', '을', '먹', '는다'

Okt

'나', '는', '밥', '을', '먹는다'

Ex 2. <하늘을 나는 자동차>

Kkma

'하늘', '을', '날', '는', '자동차'

Komoran

'하늘', '을', '나', '는', '자동차'

Okt

'하늘', '을', '나', '는', '자동차'

Ex 3. <큰 건물 뒤에 서있는 흰색 자전거가 지나간다.>

Kkma

'크', 'ㄴ', '건물', '뒤', '에', '서', '어', '있', '는', '흰색', '자전거', '가', '지나가', 'ㄴ다'

Komoran

'크', 'ㄴ', '건물', '뒤', '에', '서', '어', '있', '는', '흰색', '자전거', '가', '지나가', 'ㄴ다'

Okt

'큰', '건물', '뒤', '에', '서있는', '흰색', '자전거', '가', '지나간다'

#2 형태소 단위로 나눈 문장을 넣어 새로운 dataset_coco.json 파일을 만들어 준다.

Ex.


```
{ “image” : [ {
                    "filepath": "train2014",
                    "sentids": [
                            660072,
                            662265,
                            664215,
                            666405,
                            667128
                    ],
                    "filename": "COCO_train2014_000000547503.jpg",
                    "imgid": 123285,
                    "split": "train",
                    "sentences": [
                            {tokens":  [  "술집","에","앉",”아","술","을","마시","고",”있","는","한","무리","
의","남자","들"],
                                    "raw": "술집에 앉아 술을 마시고 있는 한 무리의 남자들",
                                    "imgid": 123285,
                                    "sentid": 660072
},
                            {"tokens": ["투수","앞","에","한","남자","가","앉",”아","있","다","."],
                                    "raw": "투수 앞에 한 남자가 앉아 있다.",
                                    "imgid": 123285,
                                    "sentid": 662265
                            },
```

{"tokens": ["사람","들","이","메모","를","하","며","바","에","앉","아",“있","다","."],
"raw": "사람들이 메모를 하며 바에 앉아 있다.",
"imgid": 123285,
"sentid": 664215
},
{"tokens": ["술","을","마시","며",“술집",“에","앉","아","있","는","한","무리","의","사람","들"],
"raw": "술을 마시며 술집에 앉아 있는 한 무리의 사람들",
"imgid": 123285,
"sentid": 666405
},
{"tokens": ["노부부","가","와인","잔","을","들","고",“바","에",“앉",“아","있","다","."],
"raw": "노부부가 와인 잔을 들고 바에 앉아 있다.",
"imgid": 123285,
"sentid": 667128
}
],
"cocoid": 547503
},
{ “file path” : “ ” ,
“sentid” : [ ] ,
“filename” : “ ” ,
“sentences” : [{ “tokens” : [“한”, “남자”, “가”, “우산”, “을”, “들고”, “있다.”],
“raw” : “한 남자가 우산을 들고 있다.” },
{ “tokens” : [“ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”],
“raw” : “ ” },
{ “tokens” : [“ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”],
“raw” : “ ” },
{ “tokens” : [“ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”],
“raw” : “ ”},
{ “tokens” : [“ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”],
“raw” : “ ” }],

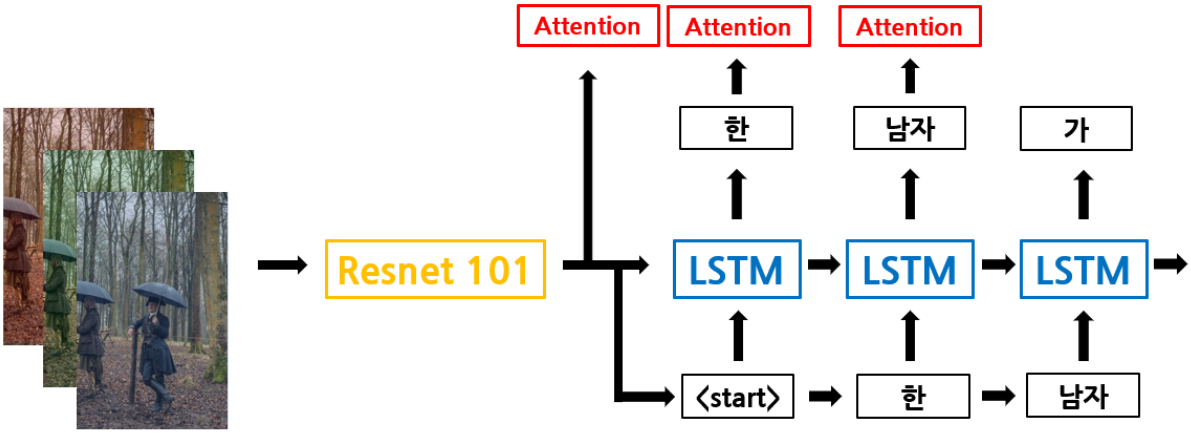
“coccoid” :
},
……]}

#3 이미지를 256 x 256 으로 resize 를 해준다.

#4 문장은 첫번째 단어 앞에 <start>가 붙고 마지막 단어 뒤에 <end> 붙는다. 이렇게 해주는 이유는 디코딩을 어디서 멈춰야하는지 알려주기 위함이다. 또한 문장의 길이를 맞춰 주기 위해 <pad>를 붙여준다.

Ex : <start>한 여자가 밥을 먹는다.<end><pad> , <start>한 남자가 우산을 들고 있다.<end>

6.2 Image Captioning



#train.py : 학습에 쓰이는 코드

```
def main():

    word_map_file = os.path.join(data_folder, 'WORDMAP_' + data_name + '.json')
    with open(word_map_file, 'r') as j: # caption 에 쓰일 한국어 word 를 불러옴
    word_map = json.load(j)

    decoder = DecoderWithAttention(attention_dim=attention_dim,
                                   embed_dim=emb_dim,
                                   decoder_dim=decoder_dim,
                                   vocab_size=len(word_map),
                                   dropout=dropout)

    decoder_optimizer = torch.optim.Adam(params=filter(lambda p: p.requires_grad,
    decoder.parameters()), lr=decoder_lr)
    encoder = Encoder()
    encoder.fine_tune(fine_tune_encoder)
```

```
encoder_optimizer = torch.optim.Adam(params=filter(lambda p: p.requires_grad,
encoder.parameters()),lr=encoder_lr) if fine_tune_encoder else None

decoder = decoder.to(device)
encoder = encoder.to(device)

# Loss 함수
criterion = nn.CrossEntropyLoss().to(device)

# 데이터 로더
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225]) # 이미지 pixel 값을 normalize
train_loader = torch.utils.data.DataLoader(CaptionDataset(data_folder, data_name,
'TRAIN', transform=transforms.Compose([normalize])),batch_size=batch_size,
        shuffle=True, num_workers=workers, pin_memory=True)
val_loader = torch.utils.data.DataLoader(CaptionDataset(data_folder, data_name,
        'VAL',    transform=transforms.Compose([normalize])),
        batch_size=batch_size,    shuffle=True, num_workers=workers,
pin_memory=True)

# Epochs
for epoch in range(start_epoch, epochs):
        if epochs_since_improvement == 20:
                break
        if epochs_since_improvement > 0 and epochs_since_improvement % 8 == 0:
                adjust_learning_rate(decoder_optimizer, 0.8) # learning rate decay :
        학습이 진행되면서 learning rate 를 줄여줌으로써 더 안정적인 학습이 되도록 함
                if fine_tune_encoder:
                        adjust_learning_rate(encoder_optimizer, 0.8)

train(train_loader=train_loader, encoder=encoder, decoder=decoder,
criterion=criterion, encoder_optimizer=encoder_optimizer
decoder_optimizer=decoder_optimizer,      epoch=epoch)



def train(train_loader, encoder, decoder, criterion, encoder_optimizer, decoder_optimizer, epoch):
        # train mode 로 해줌으로써 dropout 과 batchnorm 적용(overfittig 을 막기 위한
        regularization)
```

```
decoder.train()
encoder.train()

for i, (imgs, caps, caplens) in enumerate(train_loader):
        imgs = imgs.to(device)
        caps = caps.to(device)
        caplens = caplens.to(device)

        # Forward prop.
        imgs = encoder(imgs)
        scores, caps_sorted, decode_lengths, alphas, sort_ind = decoder(imgs, caps,
caplens)

        # Start token <start>와 마지막 token <end>를 고려
        targets = caps_sorted[:, 1:]

        scores, _ = pack_padded_sequence(scores, decode_lengths,
batch_first=True)
        targets, _ = pack_padded_sequence(targets, decode_lengths,
batch_first=True)

        # loss 계산
        loss = criterion(scores, targets)
        loss += alpha_c * ((1. - alphas.sum(dim=1)) ** 2).mean()

        # Backpropagation : 학습을 위해 gradient 를 구함
        decoder_optimizer.zero_grad()
        if encoder_optimizer is not None:
                encoder_optimizer.zero_grad()
        loss.backward()

        # gradient cliping(더 나은 학습을 위함)
        if grad_clip is not None:
                clip_gradient(decoder_optimizer, grad_clip)
        if encoder_optimizer is not None:
                clip_gradient(encoder_optimizer, grad_clip)

        # weight update
        decoder_optimizer.step()
```

```
            if encoder_optimizer is not None:
                encoder_optimizer.step()
```

 # model.py

```
class Encoder_Resnet(nn.Module):
    # Image Encodeing
    def __init__(self, encoded_image_size=14):
        super(Encoder_Resnet, self).__init__()
        self.enc_image_size = encoded_image_size

    resnet = torchvision.models.resnet101(pretrained=True) # 다른 대량의 이미지로
        학습 된 모델을 불러와서 그걸 기반으로 추가로 학습
        modules = list(resnet.children())[:-2]
        self.resnet = nn.Sequential(*modules)

        # 고정된 이미지 사이즈를 만들기 위해 adaptive average Pooling 을 사용
        self.adaptive_pool = nn.AdaptiveAvgPool2d((encoded_image_size,
            encoded_image_size))

        self.fine_tune()

    def forward(self, images):
        out = self.resnet(images) # (batch_size, 2048, image_size/32, image_size/32)
        out = self.adaptive_pool(out) # (batch_size, 2048, encoded_image_size,
                encoded_image_size)
        out = out.permute(0, 2, 3, 1) # (batch_size, encoded_image_size,
                encoded_image_size, 2048)
        return out

    def fine_tune(self, fine_tune=True):
        for p in self.resnet.parameters():
            p.requires_grad = False

        for c in list(self.resnet.children())[5:]:
            for p in c.parameters():
                p.requires_grad = fine_tune
```

```
class Attention(nn.Module):
        # 어디에 더 집중해야 할 지를 나타내는 Attention
        # 기존엔 encoding 된 image 의 pixel 의 평균을 사용하였지만, pixel 마다 그 이미지를
        설명하는 중요도가 다르기 때문에 가중치를 다르게 줌
        def __init__(self, encoder_dim, decoder_dim, attention_dim):
                super(Attention, self).__init__()
                self.encoder_att = nn.Linear(encoder_dim, attention_dim) # encoding 된 이미지를
attention 을 씌우기 위해 linear layer 로 구성하여 변형
        self.decoder_att = nn.Linear(decoder_dim, attention_dim) # decoder output 을 만들기
        위한 linear layer
                self.full_att = nn.Linear(attention_dim, 1)
                self.relu = nn.ReLU()
                self.softmax = nn.Softmax(dim=1)

        def forward(self, encoder_out, decoder_hidden):
                # encoder_out: encoding 된 이미지 (batch_size, num_pixels, encoder_dim)
                # decoder_hidden: (batch_size, decoder_dim)
                # return: attention 으로 weighted 된 encoded image
                att1 = self.encoder_att(encoder_out) # (batch_size, num_pixels,
                                        attention_dim)
                att2 = self.decoder_att(decoder_hidden) # (batch_size, attention_dim)
                att = self.full_att(self.relu(att1 + att2.unsqueeze(1))).squeeze(2)
                alpha = self.softmax(att) # (batch_size, num_pixels)
                attention_weighted_encoding = (encoder_out *
                alpha.unsqueeze(2)).sum(dim=1) # (batch_size, encoder_dim)

                return attention_weighted_encoding, alpha


class Decoder_Attention(nn.Module):
        # Decoder + Attention

        def __init__(self, attention_dim, embed_dim, decoder_dim, vocab_size,
        encoder_dim=2048, dropout=0.5):

                super(Decoder_Attention, self).__init__()

                self.encoder_dim = encoder_dim
                self.attention_dim = attention_dim
```

```
        self.embed_dim = embed_dim
        self.decoder_dim = decoder_dim
        self.vocab_size = vocab_size
        self.dropout = dropout

        self.attention = Attention(encoder_dim, decoder_dim, attention_dim) #
        attention network

        self.embedding = nn.Embedding(vocab_size, embed_dim) # embedding layer
        self.dropout = nn.Dropout(p=self.dropout) # regularization 을 위함
        self.decode_step = nn.LSTMCell(embed_dim + encoder_dim, embed_dim +
encoder_dim, bias=True) # decoder 부분에 쓰이는 lstm
        self.decode_step_2 = nn.LSTMCell(embed_dim + encoder_dim, decoder_dim,
bias=True) # decoder 부분에 쓰이는 lstm 2
        self.init_h = nn.Linear(encoder_dim, decoder_dim) # LSTMCell 의 hidden state
의 초기값을 찾기 위한 linear layer
        self.init_c = nn.Linear(encoder_dim, decoder_dim) # LSTMCell 의 cell state 의
초기값을 찾기 위한 linear layer
        self.f_beta = nn.Linear(decoder_dim, encoder_dim)
        self.sigmoid = nn.Sigmoid()
        self.fc = nn.Linear(decoder_dim, vocab_size)
        self.init_weights() # weight initialization

def init_weights(self):
        self.embedding.weight.data.uniform_(-0.1, 0.1)
        self.fc.bias.data.fill_(0)
        self.fc.weight.data.uniform_(-0.1, 0.1)

def load_pretrained_embeddings(self, embeddings):
        # 이미 다른 대량의 데이터로 학습 된 embedding 함수를 가져옴
        self.embedding.weight = nn.Parameter(embeddings)

        def fine_tune_embeddings(self, fine_tune=True):

        for p in self.embedding.parameters():
                p.requires_grad = fine_tune

def init_hidden_state(self, encoder_out):
        # hidden_state, cell_state initialization
```

```
        mean_encoder_out = encoder_out.mean(dim=1)
        h = self.init_h(mean_encoder_out) # (batch_size, decoder_dim)
        c = self.init_c(mean_encoder_out)
        return h, c

def forward(self, encoder_out, encoded_captions, caption_lengths):

        # encoder_out: encoding 된 이미지 (batch_size, enc_image_size,
        enc_image_size, encoder_dim)
        # encoded_captions: 인코딩 된 caption(한글) (batch_size, max_caption_length)
        caption_lengths: caption 의 길이 (batch_size, 1)
        return: scores, 정렬된 encoded captions, decode 길이, weights, sort index

        batch_size = encoder_out.size(0)
        encoder_dim = encoder_out.size(-1)
        vocab_size = self.vocab_size

        encoder_out = encoder_out.view(batch_size, -1, encoder_dim) #네트워크에
image를 넣기 위해 shape을 맞춤
        num_pixels = encoder_out.size(1)

        caption_lengths, sort_ind = caption_lengths.squeeze(1).sort(dim=0,
descending=True) # input data를 길이에 따라 sorting
        encoder_out = encoder_out[sort_ind]
        encoded_captions = encoded_captions[sort_ind]

        embeddings = self.embedding(encoded_captions) # 임베딩

        h, c = self.init_hidden_state(encoder_out) # hidden state, cell state
initialization

        decode_lengths = (caption_lengths - 1).tolist() # end token인 <end>는 빼고
출력할 것

        predictions = torch.zeros(batch_size, max(decode_lengths),
        vocab_size).to(device) # score를 뽑기 위함
        alphas = torch.zeros(batch_size, max(decode_lengths),
        num_pixels).to(device)
```

```
        for t in range(max(decode_lengths)):
            batch_size_t = sum([l > t for l in decode_lengths])
            attention_weighted_encoding, alpha =
                self.attention(encoder_out[:batch_size_t], h[:batch_size_t])
            gate = self.sigmoid(self.f_beta(h[:batch_size_t]))
            attention_weighted_encoding = gate * attention_weighted_encoding
            h, c = self.decode_step(
                torch.cat([embeddings[:batch_size_t, t, :],
                    attention_weighted_encoding], dim=1),
                    (h[:batch_size_t], c[:batch_size_t])) # for LSTM1
            h, c = self.decode_step_2(
                torch.cat([embeddings[:batch_size_t, t, :],
                attention_weighted_encoding], dim=1),
                (h[:batch_size_t], c[:batch_size_t])) # for LSTM2
            preds = self.fc(self.dropout(h)) # (batch_size_t, vocab_size)
            predictions[:batch_size_t, t, :] = preds
            alphas[:batch_size_t, t, :] = alpha

        return predictions, encoded_captions, decode_lengths, alphas, sort_ind
```

 # inference.py : test 할 때 쓰이는 코드

```
# 이전에 생성된 단어가 현재 매 time step 마다 LSTM 의 input 으로 들어감
def caption_image_beam_search(encoder, decoder, image_path, word_map, beam_size=3):

    # encoder: encoder model
    # decoder: decoder model
    # image_path: image 경로
    # word_map: word map
    # beam_size: 각 decoder step마다 고려 할 sequences들의 수
    # return: caption, weights


    k = beam_size
    vocab_size = len(word_map)


    # Read image and process
    img = imread(image_path)
```

```
    if len(img.shape) == 2:
        img = img[:, :, np.newaxis]
        img = np.concatenate([img, img, img], axis=2)
    img = imresize(img, (256, 256))
    img = img.transpose(2, 0, 1)
    img = img / 255.
    img = torch.FloatTensor(img).to(device)
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225]) # coco image를 normalize
    transform = transforms.Compose([normalize])
    image = transform(img)  # (3, 256, 256)

    # Encode
    image = image.unsqueeze(0)  # encoder에 넣기 위해 image shape 맞추기
    encoder_out = encoder(image)
    enc_image_size = encoder_out.size(1)
    encoder_dim = encoder_out.size(3)

    encoder_out = encoder_out.view(1, -1, encoder_dim)
    num_pixels = encoder_out.size(1)

    encoder_out = encoder_out.expand(k, num_pixels, encoder_dim)
    # beam size만큼의 encoder output. 하나의 결과만 취할 경우 초반 결과가 틀리면 아주 이상한 결
과가 나올 수 있으니
    # beam size(top k)만큼의 후보군을 뽑음

    k_prev_words = torch.LongTensor([[word_map['<start>']]] * k).to(device) #
    seqs = k_prev_words
    top_k_scores = torch.zeros(k, 1).to(device)

    seqs_alpha = torch.ones(k, 1, enc_image_size, enc_image_size).to(device)
    complete_seqs = list()
    complete_seqs_alpha = list()
    complete_seqs_scores = list()

    step = 1
    h, c = decoder.init_hidden_state(encoder_out)

    while True:
```

```
        embeddings = decoder.embedding(k_prev_words).squeeze(1)
        awe, alpha = decoder.attention(encoder_out, h)
        alpha = alpha.view(-1, enc_image_size, enc_image_size)
        gate = decoder.sigmoid(decoder.f_beta(h))
        awe = gate * awe

        h, c = decoder.decode_step(torch.cat([embeddings, awe], dim=1), (h, c))

        scores = decoder.fc(h)
        scores = F.log_softmax(scores, dim=1)

        scores = top_k_scores.expand_as(scores) + scores

        if step == 1:
            top_k_scores, top_k_words = scores[0].topk(k, 0, True, True)  # 처음 step에서는 모두 같
은 score
        else:
            top_k_scores, top_k_words = scores.view(-1).topk(k, 0, True, True)

        prev_word_inds = top_k_words / vocab_size
        next_word_inds = top_k_words % vocab_size

        seqs = torch.cat([seqs[prev_word_inds], next_word_inds.unsqueeze(1)], dim=1)
        seqs_alpha = torch.cat([seqs_alpha[prev_word_inds], alpha[prev_word_inds].unsqueeze(1)],
                               dim=1)

        incomplete_inds = [ind for ind, next_word in enumerate(next_word_inds) if
                           next_word != word_map['<end>']]
        complete_inds = list(set(range(len(next_word_inds))) - set(incomplete_inds))

        if len(complete_inds) > 0:
            complete_seqs.extend(seqs[complete_inds].tolist())
            complete_seqs_alpha.extend(seqs_alpha[complete_inds].tolist())
            complete_seqs_scores.extend(top_k_scores[complete_inds])
        k -= len(complete_inds)

        if k == 0:
            break
```

```
        seqs = seqs[incomplete_inds]
        seqs_alpha = seqs_alpha[incomplete_inds]
        h = h[prev_word_inds[incomplete_inds]]
        c = c[prev_word_inds[incomplete_inds]]
        encoder_out = encoder_out[prev_word_inds[incomplete_inds]]
        top_k_scores = top_k_scores[incomplete_inds].unsqueeze(1)
        k_prev_words = next_word_inds[incomplete_inds].unsqueeze(1)
        if step > 50:
            break
        step += 1

    i = complete_seqs_scores.index(max(complete_seqs_scores))
    seq = complete_seqs[i]
    alphas = complete_seqs_alpha[i]

    return seq, alphas
```

6.3 Client(Android)

```
2020-06-17 21:21:18.224 13498-13498/com.example.android.camera2basic I/System.out: 10초마다 사진 찍기
2020-06-17 21:21:18.316 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:19.056 13498-13972/com.example.android.camera2basic I/System.out: 이미지 파일 크기 : 2343743
2020-06-17 21:21:19.116 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:19.560 13498-13972/com.example.android.camera2basic I/System.out: 이미지 전송 완료
2020-06-17 21:21:19.560 13498-13972/com.example.android.camera2basic I/System.out: 결과 대기중
2020-06-17 21:21:19.917 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:20.716 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:21.517 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:22.317 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:23.118 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:23.917 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:24.717 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:25.518 13498-13613/com.example.android.camera2basic I/System.out: 0.8초마다 음성 체크중
2020-06-17 21:21:26.029 13498-13972/com.example.android.camera2basic I/System.out: 받은 문장 : 나무 책상 위에 서 있 는 인형
```

6.3.1 10초마다 사진찍기

```
Handler mHandler = new Handler() {
    public void handleMessage(Message msg) {
        System.out.println("10초마다 사진 찍기");
        takePicture();          // 사진 찍기
        Log.e("Timer", "찰칵");
        mHandler.sendEmptyMessageDelayed(0,10000);
    }
```

```
};
```

6.3.2 서버와의 통신 (AsyncTask 이용) – 언어가 다르기 때문에 bytes 형태로 소켓 통신

```
try {     //서버와 소켓 연결
    Log.d("TCP", "server connecting");
    InetAddress serverAddr = InetAddress.getByName(SERV_IP);
    Socket sock = new Socket(serverAddr, PORT);
    DataInputStream dis = null;
    DataOutputStream dos = null;
    try{          //이미지 파일을 읽어와 bytes 형태로 서버에 전송
        File      file=new      File(Environment.getExternalStorageDirectory().getPath()      +
"/Android/data/com.example.android.camera2basic/files/" , "pic.jpg");
        Log.d("debug",file.getPath());
        dis=new DataInputStream(new FileInputStream(file));
        dos=new DataOutputStream(sock.getOutputStream());
        byte[]buf=new byte[1024];

        int readBytes;

        dos.write(longToBytes(file.length()),0,8);
        System.out.println("이미지 파일 크기 : "+file.length());

        while((readBytes=dis.read(buf))>0){          //길이 정해주고 서버로 전송.
            if(readBytes==0){
                dos.flush();
            }
            dos.write(buf,0,readBytes);
        }
        System.out.println("이미지 전송 완료");
    } catch (IOException e) {
        Log.d("TCP", "don't send message");
        e.printStackTrace();
    }
    try {          //서버에서 나온 결과값을 받기 위해 대기
        BufferedReader br = null;
        br = new BufferedReader(new InputStreamReader(sock.getInputStream(), "UTF-8"));

        while(true) {
            System.out.println("결과 대기중");
```

```
                sentence = br.readLine().substring(4);
                if(sentence != "") {
                    Log.w("recv", sentence);
                    System.out.println("받은 문장 : " + sentence);
                    dis.close();
                    dos.close();
                    br.close();
                    sock.close();
                    break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
} catch (UnknownHostException e) {
        e.printStackTrace();
} catch (IOException e) {
        e.printStackTrace();
}
```

6.3.3 TextToSpeech – 위와 다른 task로 0.8초마다 새로운 문장이 왔는지 체크, 새로운 문장이 왔다면 음성 출력.

```
public void testStart() {
        second = new TimerTask() {

            @Override
            public void run() {
                System.out.println("0.8초마다 음성 체크중");
                if(!sentence.equals(mainFragment.getSentence())) {
                    sentence = mainFragment.getSentence();
                    Update();
                }
            }
        };
        Timer timer = new Timer();
        timer.schedule(second, 0, 800);
}
```

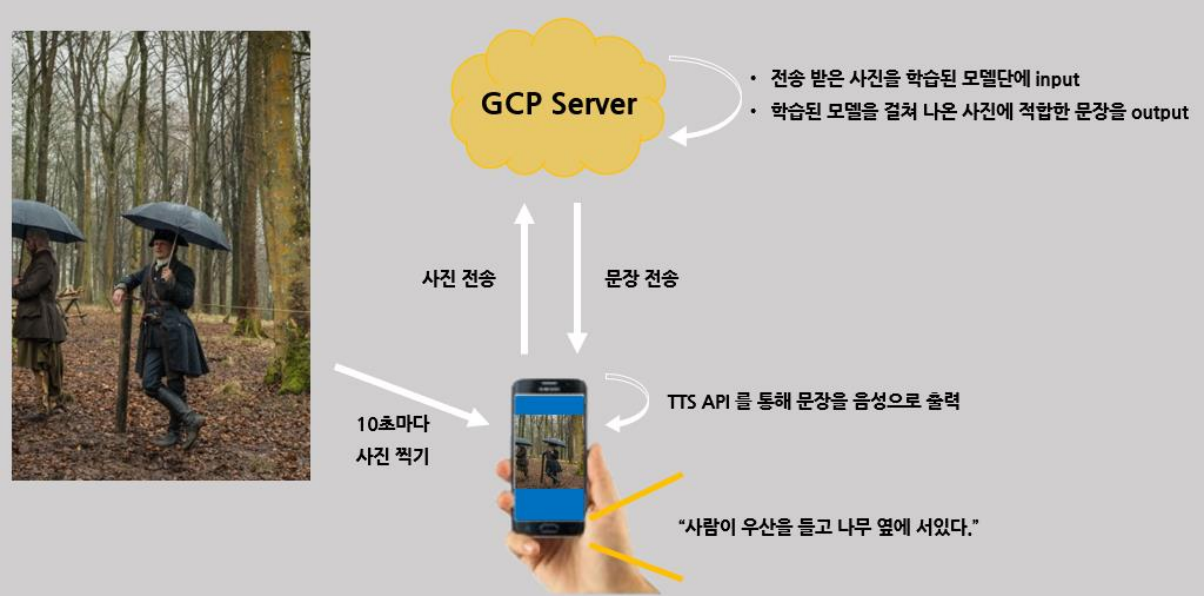
6.4 Server(Python)

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((TCP_IP, TCP_PORT))
server_socket.listen(5)
print("TCPServer Waiting for client on port 9999")
while True:
    print("대기중")
    client_socket, address = server_socket.accept()
    print("I got a connection from ", address)
    image_size = client_socket.recv(8)
    size = int.from_bytes(image_size, byteorder="big") #java는 big endian
    img_data = client_socket.recv(1024)
    data = img_data
    size -= len(img_data)
    if img_data:
        while len(img_data):
            img_data = client_socket.recv(1024)
            data += img_data
            size -= len(img_data)
if(size == 0):
                print("clear")
                break;
    print("이미지 파일 확인")
    img_file = open("blah/Client.jpg", "wb")
    print("finish img recv")
    print(sys.getsizeof(data))
    img_file.write(data)
    img_file.close()        # 소켓으로 이미지 수신 완료, ./blah 디렉토리에 저장

subprocess.run(["python","tools/eval.py","--model","log_fc/model.pth","--
infos_path","log_fc/infos_fc.pkl","--image_folder","blah","--num_images","1"]) # ./blah 디렉토리에
있는 이미지 파일을 모델에 입력, 결과값은 caption.txt에 저장
    file = open('caption.txt','r')
    s = file.read()
    s= s.encode()
    file.close()
    length = len(s)
    client_socket.sendall(length.to_bytes(4, byteorder="little")) # python은 little endian
    client_socket.sendall(s)
    print("text :" + s.decode() +" send")
```

client_socket.close()

7. 실행 (Test)



7.1 실행 결과

: 한 남자가 건물 앞에 서있다.

: 보도에 표지판이 있다.

: 자전거가 보도에 주차되어 있다.

7.2 System Test

TEST CASE	NAME	DESCRIPTION	Pass/Fail
1-1	전면부 상황 입력	스마트폰 앱에서 10초마다 사진이 찍히는지 테스트 한다.	Pass
1-2	전면부 상황 입력	새로 핸드폰 카메라로 '사람이 우산을 들고 나무 옆에 있는 사진'을 찍고 그 사진을 이미 GCP에서 학습된 weight를 갖고 있는 모델의 입력으로 넣고, 모델 입력 단에서 카메라로 찍은 사진 frame이 들어오는지 확인하는 코드를 추가한다. Input image가 잘 들어오면 'OK', 아니면 'No Image'라는 메시지를 코드단에서 출력하도록 한다.	Pass
2	딥러닝 학습을 통한 모델	Test data(train에서 사용하지 않은 MSCOCO 8000장)로 모델을 평가하였을 때, 평균적인 BLUE Score-4 기준 어절단위 0.111 의미형태소 단위 0.225 형태소 단위 0.251 이상이면 신뢰도를 갖고 있다고 판단한다. ⇨ **BLEU-4 기준 형태소 단위 0.302**	Pass
3	학습된 모델을 통한 결과 도출	위에서 정의한 '사람이 우산을 들고 나무 옆에 있는 사진' 과 그에 맞는 문장 '사람이 우산을 들고 나무 옆에 서있음' 을 준비해두고, 준비해둔 사진을 모델의 입력으로 넣었을 때, 미리 준비해둔 그 사진을 설명하는 문장 '사람이 우산을 들고 나무 옆에 서있음'과 유사한 문장이 10초 이내에 출력 되는지 확인한다. ⇨ **데모 영상을 통해 10초 이내에 나오는 것을 확인할 수 있다.**	Pass
4	TTS (Text To Speech)	결과로 나온 문장을 TTS API를 통해 음성으로 바꿔서 출력하는지를 체크한다.	Pass

Test case 3 (PASS) + Success Criteria (만족)

8. Reference

Okt
사이트 : https://openkoreantext.org/

KKMA
사이트 : http://kkma.snu.ac.kr/documents/index.jsp

Komoran
사이트 : https://docs.komoran.kr/