A Capsule Network for Traffic Speed Prediction in Complex Road Networks

최우진

2020/11/26

0. Abstract (요약)

- 교통 흐름 데이터(3D)를 이미지(2D)로 변환.
- CNN을 이용하여 교통 흐름을 예측한 것은 좋은 성능을 가진다.
- 하지만 max pooling 이라는 단점을 가지고 있다.
- max pooling은 중요한 정보를 잃게 만든다.
- 따라서 본 논문은 capsule을 사용한 neural network를 만들었고 max pooling을 dynamic routing으로 대체했다.
- 성능(MSE)는 CNN보다 13.1% 더 좋게 나왔다.

1. Inrtoduction (도입)

- Fig. 2에서 인접한 값은 반드시 인접한 센서에서 측정된 값이 아니다.

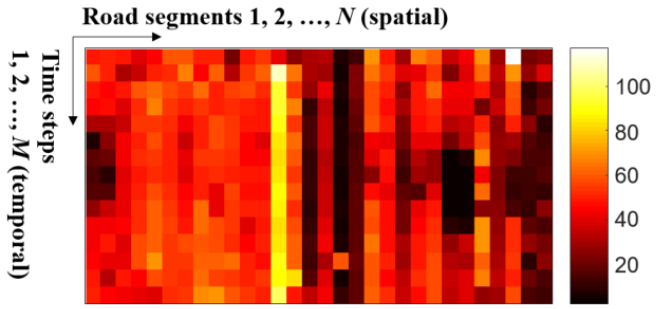


Fig. 2: Spatio-temporal image representation of traffic speed data (unit: km/h).

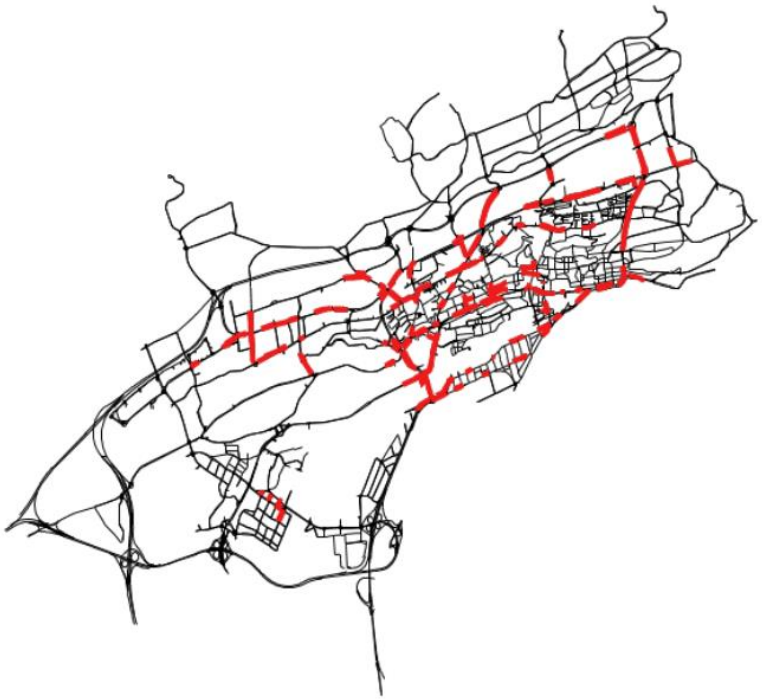


Fig. 1: Road network of central Santander city. Red lines denote road segments where the speed sensors are located.

2. Traffic Speed Prediction (교통 속도 예측)

- A. Traffic Speed Data as an Image (교통 속도 데이터(3D) -> 이미지(2D))

- Input(X)
- X.shape = [M, N]

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{M1} & \cdots & x_{MN} \end{bmatrix}$$

- Output
- Output.shape = [L x N, 1]

$$Y = \begin{bmatrix} y_1 & \cdots & y_N & y_{N+1} & \cdots & y_{LN} \end{bmatrix}$$

2. Traffic Speed Prediction (교통 속도 예측)

- B. CNN for Traffic Speed Prediction (CNN을 사용하여 교통 속도 예측)

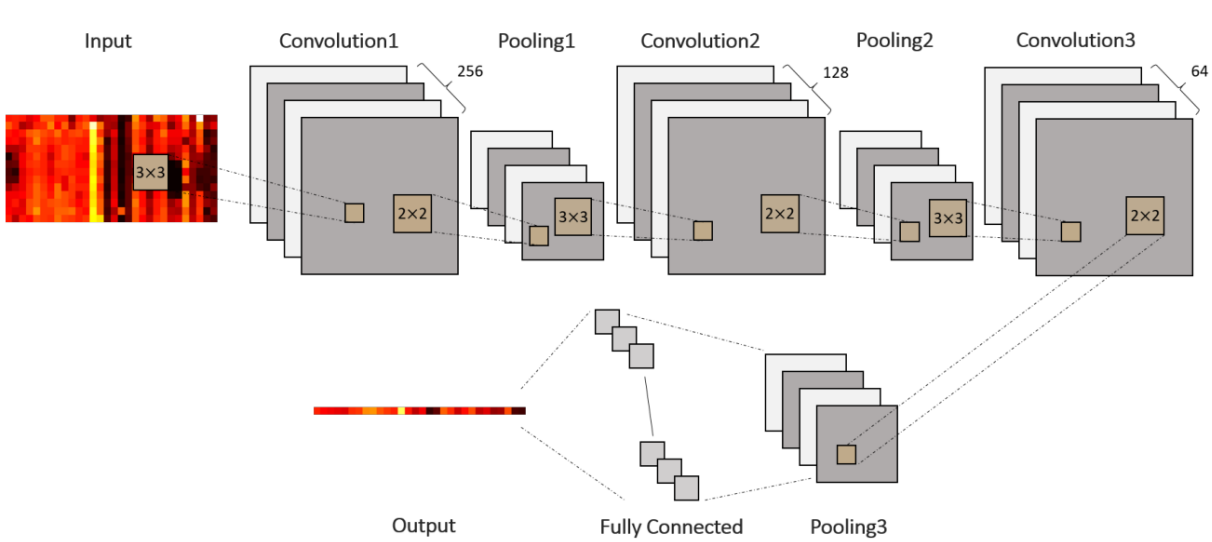


Fig. 3: Architecture of CNN for traffic speed prediction.

TABLE I: Layer parameters of CNN.

Layer	Parameter	Activation
Convolution1	(256, 3, 3)	ReLu
Pooling1	(2, 2)	-
Convolution2	(128, 3, 3)	ReLu
Pooling2	(2, 2)	-
Convolution3	(64, 3, 3)	ReLu
Pooling3	(2, 2)	-
Flattening	-	-
Fully-connected	-	-

2. Traffic Speed Prediction (교통 속도 예측)

- C. Proposed CapsNet Architecture (제안된 CapsNet 구조)

- CNN은 max pooling을 사용하여 가치 있는 정보를 잃는다.
- Capsule은 특징 검출 가능성을 출력 벡터의 길이로 인코딩하는 뉴런의 그룹이다.
- CapsNet은 max pooling(CNN의 단점)을 대신하여 dynamic routing에 의해 학습된다.
- Dynamic routing은 두 연속적인 캡슐 레이어 사이에서 낮은 레벨 캡슐이 그들의 입력을 그 입력과 일치하는 높은 레벨 캡슐로 어떻게 보낼지에 대한 가중치를 업데이트할 때 사용된다.
- 다시 말해, 그 가중치는 낮은 레벨 캡슐과 높은 레벨 캡슐의 dot product을 기반으로 정해진다. dot product는 두 벡터의 유사성을 특징으로 추출한다.

2. Traffic Speed Prediction (교통 속도 예측)

- C. Proposed CapsNet Architecture (제안된 CapsNet 구조)

- 그런 다음 각 낮은 레벨 캡슐의 가중치 합은 벡터의 방향을 유지하면서 길이를 1 이하로 줄이는 squash function을 통과한다.

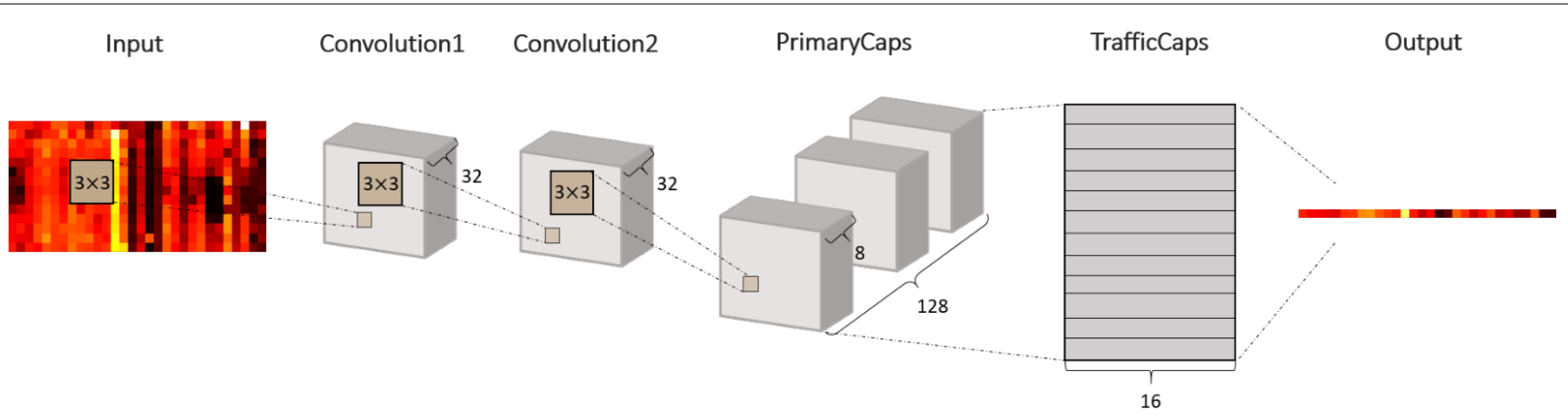
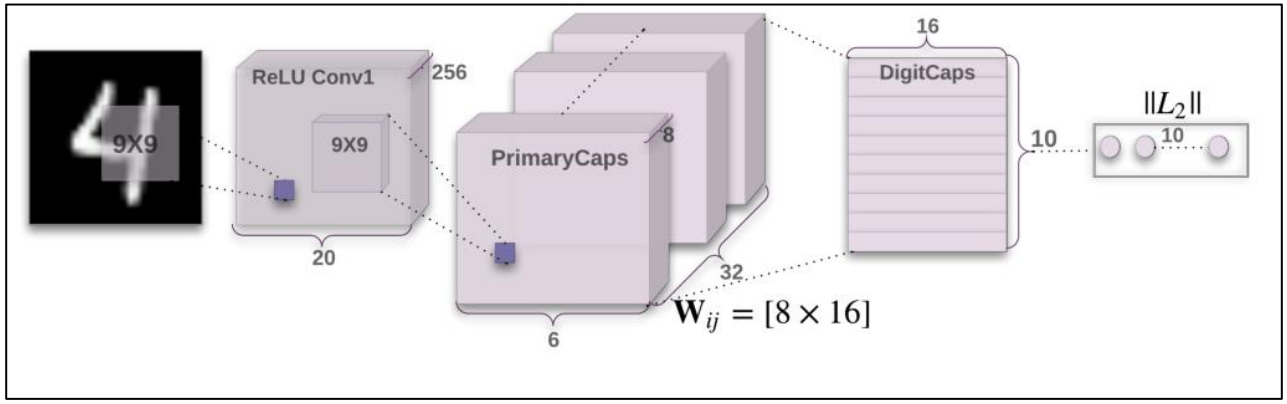


Fig. 4: Architecture of CapsNet for traffic speed prediction.

Dynamic Routing Between Capsules

Dynamic Routing Between Capsules

- CapsNet architecture



- Input: 28 x 28 x 1 (mnist data)
- Conv1: 20 x 20 x 256

Kernel: 9x 9 x 256, stride 1 + Relu

- PrimaryCaps: 6 x 6 x (8 x 32)

Kernel: 9x 9 x (32 x 8), stride 2+ Relu

- DigitCaps: 10 x 16

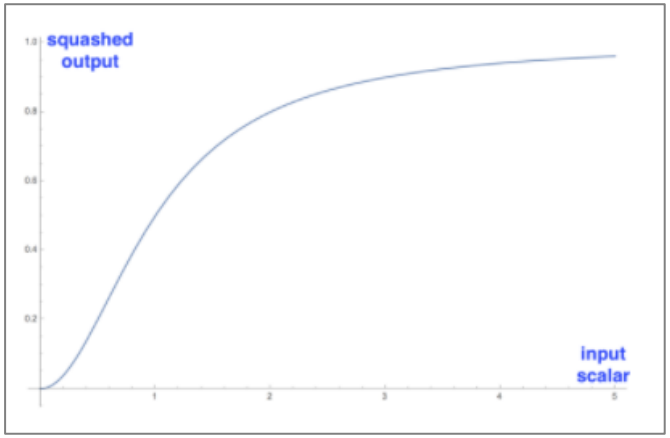
Dynamic Routing

Dynamic Routing Between Capsules

- squashing function (기존에 알던 활성화함수)
- 벡터의 방향을 유지시켜주면서 크기를 1이하로 줄이는 함수

$$\mathbf{v}_j = \underbrace{\frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}}_{\text{additional "squashing"}} \underbrace{\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}}_{\text{unit scaling}}$$

$$= squash(s_j)$$



- 출력 벡터의 길이는 캡슐에 의해서 인식되는 특정 개체의 존재 확률로 해석이 가능하다.
- 오른쪽 그림은 스칼라 값으로 그린 squashing 함수이다.

Dynamic Routing Between Capsules

- squashing function

$$\mathbf{v}_j = \frac{||\mathbf{s}_j||^2}{1 + ||\mathbf{s}_j||^2} \frac{\mathbf{s}_j}{||\mathbf{s}_j||}$$

$$\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j|i} , \qquad \hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij} \mathbf{u}_i$$

u_i: prediction vectors
s_j: total input
v_j: vector output of capsule
c_{ij}: coupling coefficient

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} = softmax(b_{ij})$$

Dynamic Routing Between Capsules

- Routing algorithm

primaryCaps (6x6x32, 8)

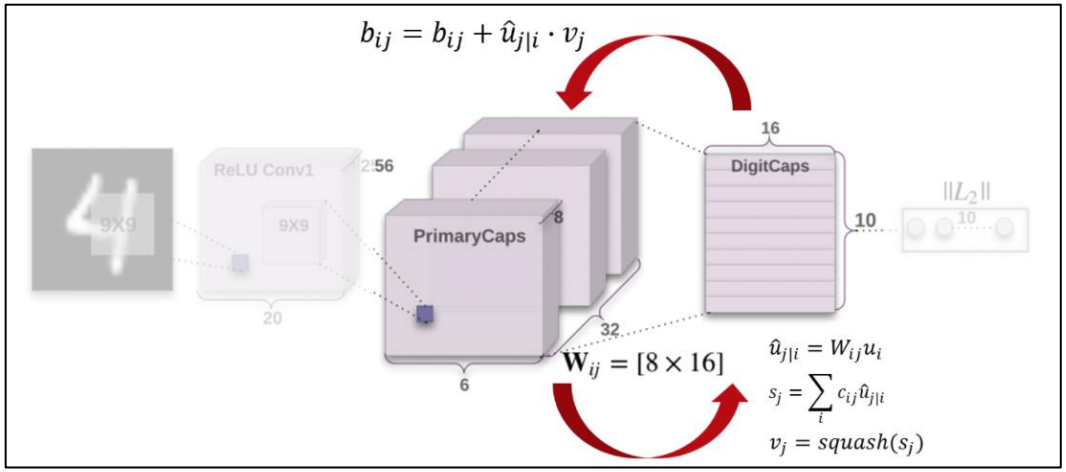
DigitCaps (10, 16)

Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

r: 하이퍼파라미터
보통 3으로 설정

w_{ij}: (8, 16)
u_i: (1, 8)
$\hat{u}_{j|i}$, v_j: (1, 16)



3. Performance Validation With Real Data (실제 데이터를 통한 성능 검증)

- 2016년 central Santander시 15분마다 측정된 데이터
- 전처리 후 Matrix 크기는 33054 x N(road segment 수)
- 희박하게 누락된 각 측정치는 다른 날들의 해당 시간에 측정한 평균 값으로 대체된다.
- Training Data는 1월 ~ 9월, Test Data는 10월 ~ 12월으로 하였다.
- 총 4개의 Task

output input

- Task 1: 15-min prediction with 150-min traffic history on 20 road segments ($L = 1, M = 10, N = 20$)
- Task 2: 30-min prediction with 150-min traffic history on 20 road segments ($L = 2, M = 10, N = 20$)
- Task 3: 15-min prediction with 210-min traffic history on 50 road segments ($L = 1, M = 14, N = 50$)
- Task 4: 30-min prediction with 210-min traffic history on 50 road segments ($L = 2, M = 14, N = 50$)

input

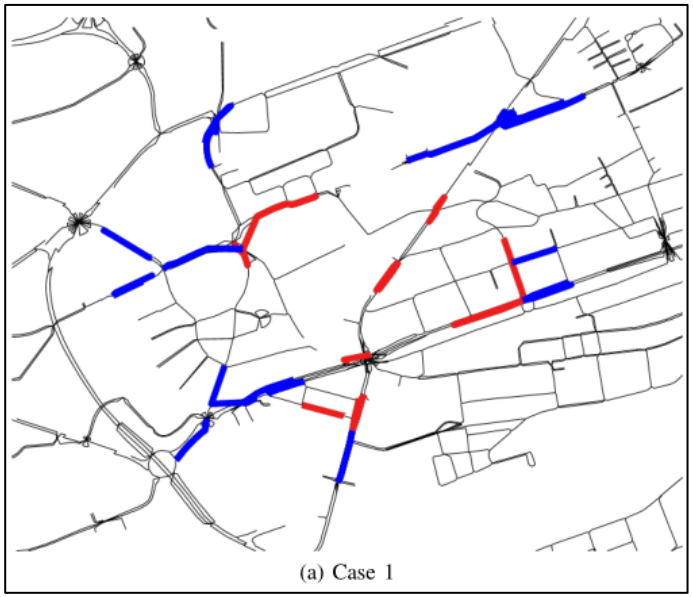
$$X = \begin{bmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{M1} & \cdots & x_{MN} \end{bmatrix}$$

output

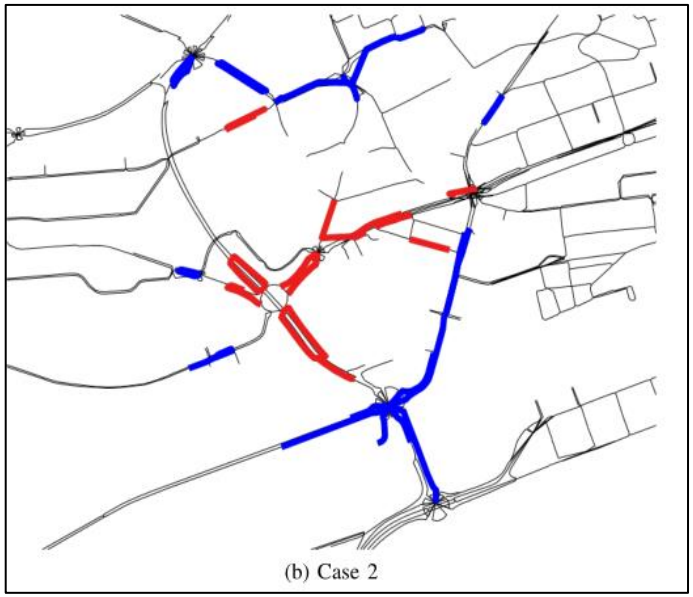
$$Y = [y_1 \quad \cdots \quad y_N \quad y_{N+1} \quad \cdots \quad y_{LN}]$$

3. Performance Validation With Real Data (실제 데이터를 통한 성능 검증)

- 총 4개의 Task



(a) Case 1



(b) Case 2

Red	• Task 1: 15-min prediction with 150-min traffic history on 20 road segments ($L = 1, M = 10, N = 20$)	• Task 2: 30-min prediction with 150-min traffic history on 20 road segments ($L = 2, M = 10, N = 20$)
Red + Blue	• Task 3: 15-min prediction with 210-min traffic history on 50 road segments ($L = 1, M = 14, N = 50$)	• Task 4: 30-min prediction with 210-min traffic history on 50 road segments ($L = 2, M = 14, N = 50$)

3. Performance Validation With Real Data (실제 데이터를 통한 성능 검증)

- Loss functio은 MSE(mean squared error)이고, Adam optimizer를 사용
- 각 속도 값을 [0, 1]으로 정규화
- 3가지 metric을 통해 검증
- MRE(mean relative error), MAE(mean absolute error), RMSE(root mean squared error)

$$MRE = \frac{\sum_{i=1}^{I} |y_i - \hat{y}_i| / y_i}{I}$$

$$MAE = \frac{\sum_{i=1}^{I} |y_i - \hat{y}_i|}{I}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{I} (y_i - \hat{y}_i)^2}{I}}$$

y_i: true value
$\hat{y}_i$: i-th speed prediction
I: 평가 세트의 속도 데이터 수

3. Performance Validation With Real Data (실제 데이터를 통한 성능 검증)

- 결과

(a) Case 1

	CNN			CapsNet		
	MRE	*MAE*	*RMSE*	*MRE*	*MAE*	*RMSE*
Task 1	5.668	6.102	10.30	0.444	5.675	8.853
Task 2	0.649	6.204	10.47	0.289	5.791	9.179
Task 3	18.14	6.323	10.68	5.146	5.790	9.257
Task 4	4.661	6.583	10.85	0.876	5.898	9.472

(b) Case 2

	CNN			CapsNet		
	MRE	*MAE*	*RMSE*	*MRE*	*MAE*	*RMSE*
Task 1	37.35	6.519	10.98	1.555	6.109	9.362
Task 2	21.41	6.667	11.19	10.97	6.240	9.718
Task 3	19.76	6.915	11.29	9.746	6.113	9.674
Task 4	2.333	6.957	11.38	4.146	6.243	9.913

- MRE는 일정한 결과를 보여주지 못한다.
- 반면, MAE와 RMSE는 input과 output size 증가처럼 증가한다.
- 두 케이스 모두 MAE와 RMSE가 CapsNet이 CNN보다 작다.
- 평균적으로 CapsNet에서 MAE와 RMSE에서 각각 8.24%, 13.1% 향상했다.

3. Performance Validation With Real Data
(실제 데이터를 통한 성능 검증)

- CapsNet의 단점
- 네트워크 학습 시간이 오래 걸린다.
- Task1에서 CNN보다 30배 오래 걸린다.
- CapsNet의 파라미터는 8.24×10^6(Task1) ~ 143×10^6(Task4)이지만,
  CNN의 파라미터는 0.374×10^6(Task1) ~ 0.410×10^6(Task4)이다.
- CapsNet에서 routing 알고리즘은 캡슐이라는 다차원 벡터들 사이의 모든 조합을
  테스트하여 full-scale 이미지 특징을 다루기 때문에 많은 학습 파라미터가 필요하다.
- 또한 input과 output의 크기가 커지면 학습 파라미터가 상당히 커진다.

4. Conclusion (결론)

- CNN보다 네트워크 학습시간은 오래 걸리지만, CNN보다 spatio-temporal 특징을 잘 학습하여 대략 13.1% 성능 향상을 했다.
- 또한 입력과 출력의 크기가 커지면 더욱 성능 향상 폭이 컸다.

5. Reference (참조)

- Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in neural information processing systems. 2017.
- Kim, Youngjoo, et al. "A capsule network for traffic speed prediction in complex road networks." 2018 Sensor Data Fusion: Trends, Solutions, Applications (SDF). IEEE, 2018.
- https://m.blog.naver.com/bootpay/221162937822
- https://jayhey.github.io/deep%20learning/2017/11/28/CapsNet_2/

A Capsule Network for Traffic Speed Prediction

in Complex Road Networks

감사합니다